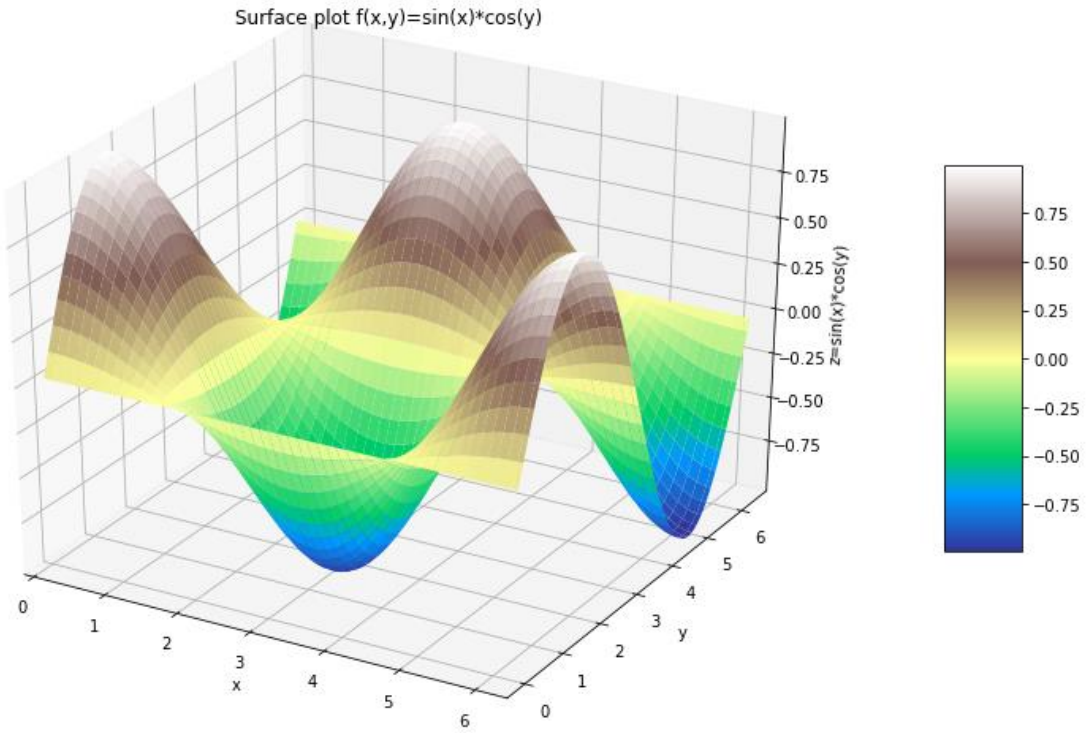
NUMERICAL MATH WITH PYTHON LANGUAGE



M. Turhan ÇOBAN

EGE University, School of Engineering, Department of Mechanical Engineering

www.turhancoban.com

version:25.01.2022

INDEX

1. **PYTHON PROGRAMMING LANGUAGE BASICS**
   - **1.1 Loading Python environment**
   - **1.2 Basic processes and variables**
   - **1.3 If statement**
   - **1.4 For and while statement, arrays and range method**
   - **1.5 Class concept in python**
   - **1.6 Abstract classes**
   - **1.7 Math library**
   - **1.8 Plotting in python**
2. **TAYLOR SERIES AND SERIES APPROXIMATIONS TO FUNCTIONS**
   - **2.1 Taylor series**
   - **2.2 Series solutions to functions**
3. **DERIVATIVES**
   - **3.1 Definitions**
   - **3.2 Numerical derivations(Differnece formulas)**
4. **INTEGRALS**
   - **4.1 Definitions**
   - **4.2 Newton-Cotes Numerical integrals**
   - **4.3 Romberg integration**
   - **4.4 Gauss-Legendre Numerical integration**
   - **4.5 Integration formulations with adjustable error**
   - **4.6 Monte-Carlo integral**
   - **4.7 Multidimensional integration**
5. **COMPLEX NUMBERS**
6. **LINEAR SYSTEM OF EQUATIONS**
   - **6.1 Gauss elimination method**
   - **6.2 Gauss-Jordan method**
   - **6.3 LU decomposition**
   - **6.4 Iterative methods (Jacobi, Gauss-Seidel and Successive relexation)**
   - **6.5 Methods for band matrices**
   - **6.6 Conjugate gradient methods**
   - **6.7 Unit matrix, Transpose matrix, Determinant and matrix inversion**
   - **6.8 Cramer's rule**
   - **6.9 Problems**
7. **ROOTS OF NONLINEAR FUNCTIONS**
   - **7.1 Bisection method**
   - **7.2 False position (Regula falsi)**
   - **7.3 Fixed position method**
   - **7.4 Newton-Raphson method**
   - **7.5 Muller's method**
   - **7.6 Inverse quadratic Lagrange interpolation**
   - **7.7 Brent method**
   - **7.8 Higher order methods**

7.9 Ridder's method
7.10 Roots of the quadratic and cubic polynomials
7.11 Roots of the polynomials by using synthetic division process(Horner method..)
7.12 Roots finding by using software packages
7.13 Finding root limits and multiple roots
7.14 Problems

8. OPTIMISATION
8.1 One dimensional, non-linear Newton's method
8.2 One dimensional non-linear Fibbonachi(Golden search) method
8.3 One dimensional non-linear Quadratic polynomials method
8.4 Multidimensional non-linear Newton-Raphson method
8.5 Multidimensional non-linear Nelder-Mead simplex method
8.6 Multidimensional non-linear Steepest-Descent method
8.7 Multidimensional non-linear Stochastic firefly algorithm
8.8 Multidimensional non-linear Stochastic Particle swarm algorithm
8.9 Problems

9. NON-LINEAR SYSTEM OF EQUATIONS
9.1 Newton's method
9.2 Homotophy or continuation methods
9.3 Solution of the non-linear system of equations by using optimisation algorithms
9.4 problems

10. CURVE FITTING
10.1 Linear curve fitting: leasts square methods
10.2 Linear curve fitting: Orthogonal polynomial least square methods
10.3 Linear curve fitting: least square methods with multivariables
10.4 Non linear curve fitting: least square methods with non-linear equations
10.5 Interpolation: Moving least squares, one and multivariables
10.6 Interpolation: Newton, Lagrange and Hermit interpolation formulas
10.7 Interpolation: Cubic spline and B spline interpolation
10.8 Interpolation: Van der Monde interpolation
10.9 Function Approximation: Pade rational approximation
10.10 Function Approximation: Chebshev rational approximation
10.11 Function Approximation: Trigonometric (Fourier) approximation
10.12 Function Approximation: Orthogonal Legendre approximation
10.13 Problems

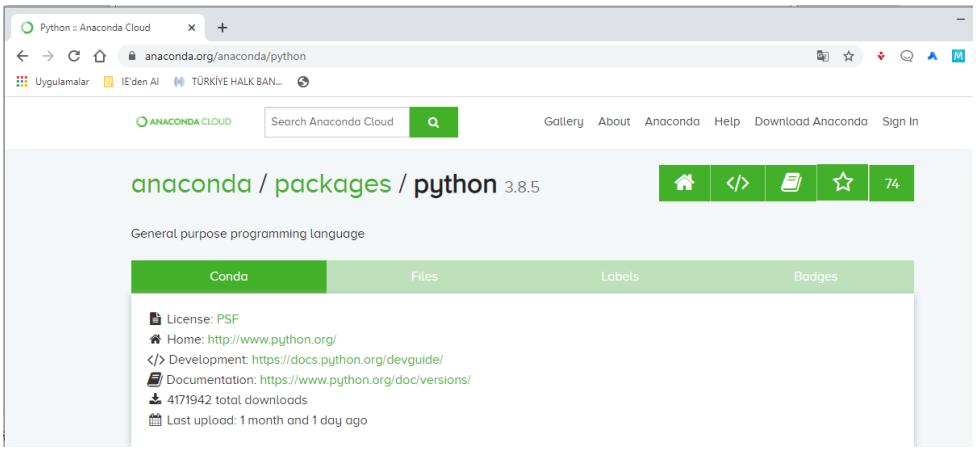
11. DIFFERENTIAL EQUATIONS
11.1 Euler method
11.2 Runge-Kutte methods
11.3 Predictor-Corrector methods (Multistep-open methods)
11.4 Solution of some differential equations
11.5 Problems

12. EIGENVALUE PROBLEM
12.1 Introduction to Eigenvalues
12.2 Jacobi method for symmetric matrices
12.3 Characteristic polynomials and Levverier method
12.4 Singular value decomposition
12.5 Problems

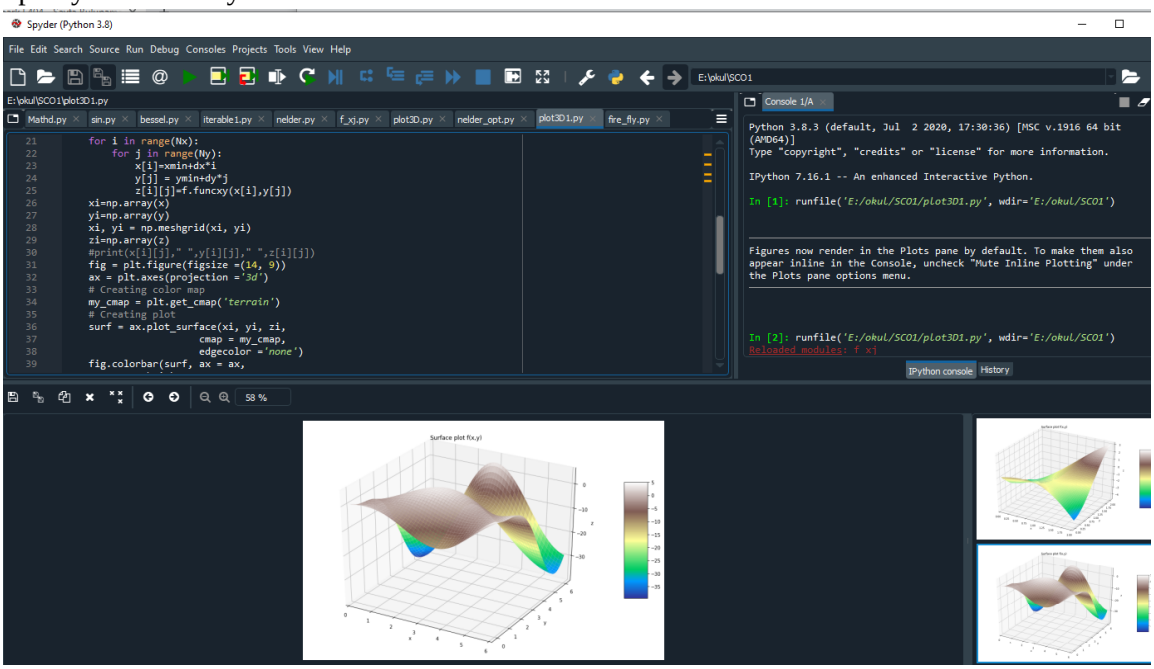
1. PYTHON PROGRAMMING LANGUAGE BASICS

1.1 LOADING PYTHON ENVIRONMENT INTO YOUR COMPUTER

We are recommending anaconda python version. You can obtain this from https://anaconda.org/anaconda/python



Adress. Furthermore package include several editors to work with. We are using spyder editor, but it is up to you to select your own.



Python is an interpreted language. An **interpreted language** is a type of language for which most of its implementations execute instructions directly, without previously compiling to machine language (binary) code instructions. The Python interpreter executes the program directly, translating each statement into macine language code. Elimination of compiling process makes running of the program simpler, but execution of the program is much slower compare to compiled programming languages such as C. Authors personel preference for complex numerical modelling and processes languages like C, C++ or java, of which C and C++ are compiled languages and java has two step procedure. In the first step it is compiled to a virtual code close to machine language, and then interpret this code into machine language.

1.2 BASIC PROCESSES AND VARIABLES

In Python language when defining the variables, method and classes variable types is not specified. When the data is loaded to a value into the variable language determine the variable type. This has both positive and negatve effects. Very beginner programmers does not have to learn variable type immadetly, but yet because the language using them, ıt is necassary to learn them and using them properly. Passaging of variables into methods(functions) and classes are a little different compare to languages with enforced variable type specifications.

You can run the python statements directly in console environment

```
In [4]: 2*sin(pi/3.0)
Out[4]: 1.7320508075688772

In [5]: x=2.3*sin(pi/4.0)

In [6]: y=x*2+3.2*x+5

In [7]: 2.3*3.4-5.5
Out[7]: 2.3199999999999994
```

But it is only practical when the codes are simple. For the complex cases programs should be written and running (through enterpreter). Programs should be saved as **name.py**. Simple examples explain variables and input andoutput will be given here.

Welcome message (String)

```
print("welcome to python programming language" )
```

```
runfile('E:/okul/SCO1/if_x_test.py', wdir='E:/okul/SCO1')
welcome to python programming language
```

Pyton supports integer (int) and floating numbers(float). Declaration of the type is not necassary in python, when you need an integer numbers declare it as x=15 and when you need a floating numbers declare it as x=15.0.

Integer numbers are sored in memory of computer a s a series of binary bits. They are called bytes. Different number of bits can be located in a byte. In python bytes of an integer can be changed acording to actual numbers stored

Table writing integer 15 as an integer variable in computer memory as binary bits

+/-	2^6	2^5	2^4	2^3	2^2	2^1	2^0
+	64	32	16	8	4	2	1
0	0	0	0	1	1	1	1

```
import sys
x=75
y=sys.getsizeof(x)
print(y)
28
w=2**75
y1=sys.getsizeof(w)
print(y1)
36
```

Sys.getsizeof(x) returns the size of integer in bits that stored in computer memory

writing 22.625x10^{-17} in a floating numbers as binary bits

+/-	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	+\-	2^4	2^3	2^2	2^1	2^0
+	16	8	4	2	1	0.5	0.25	0.125	0.0625	-	16	8	4	2	1
0	1	0	1	1	0	1	0	1	0	1	1	0	0	0	1

Python supports four different numerical types −

- int (signed integers)
- long (long integers, they can also be represented in octal and hexadecimal)
- float (floating point real values)
- complex (complex numbers)

If a string variable is consider, you use double or single quote signs
x=”Turhan” or x=’Turhan’

```
x=15
y=15.0;
z=int(15);
w='15'
q=int(w)
r=float(w)
s=float(x)
print("x=",x,"y=",y,"z=",z,"w=",w,"q=",q,"r=",r,"s=",s)
```

```
runfile('E:/okul/SCO1/number.py', wdir='E:/okul/SCO1')
x= 15 y= 15.0 z= 15 w= 15 q= 15 r= 15.0 s= 15.0
```

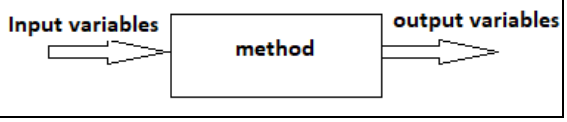
Arithmetic processes of integer + add ; - substract * multiply; / divide; % remaining in divide
Integer variable reading, calculating & printing

```
x=int(input("x="))
y=2*x+5
print("x =",x,"y = ",y)
```

```
runfile('E:/okul/SCO1/int1.py', wdir='E:/okul/SCO1')

x=5
x = 5 y =  15
```

Methods or functions are program pieces that has input variables and output variables. Structure is as follows:

```
def method_name(method input variable,method input variable,..):
    program statements
    …..
    return  method output variable
```



Integer variable reading, calculating by using method(function) & printing

```
def f(x):
   y=2*x+5
   return y;

x=int(input("x="))
print("x =",x,"y = ",f(x))
```

```
runfile('E:/okul/SCO1/int2.py', wdir='E:/okul/SCO1')

x=5
x = 5 y =  15
```

Integer variable reading, calculating by using method(function) & printing

```
def divide(x,y):
   z=x/y
   return int(z);
def remain(x,y):
   z=x%y
```

```
    return z;

x=int(input("x="))
y=int(input("y="))
print("x =",x,"y = ",y,"x/y = ",divide(x,y),"x%y = ",remain(x,y))
```

```
runfile('E:/okul/SCO1/int3.py', wdir='E:/okul/SCO1')

x=7

y=3
x = 7 y =  3 x/y =  2 x%y =  1
```

```
number1=int(input("number1 = "))
number2=int(input("number2 = "))
sum = number1+number2;
substract = number1-number2
multiply = number1*number2
divide=int(number1/number2)
remain=number1%number2
s=str(number1)+" +  "+str(number2)+" = "+str(sum)+"\n"
s+=str(number1)+" -  "+str(number2)+" = "+str(substract)+"\n"
s+=str(number1)+" x  "+str(number2)+" = "+str(multiply)+"\n"
s+=str(number1)+" /  "+str(number2)+" = "+str(divide)+"\n"
s+=str(number1)+" %  "+str(number2)+" = "+str(remain)+"\n"
print(s)
```

```
runfile('E:/okul/SCO1/string2.py', wdir='E:/okul/SCO1')

number1 = 7

number2 = 3
7 +  3 = 10
7 -  3 = 4
7 x  3 = 21
7 /  3 = 2
7 %  3 = 1
```

Note when adding strings together in tor float will be converted dto string by using str(variable) method

Float variable(real numbers) reading, calculating and printing

```
x=float(input("x="))
y=x*x-2.3*x+5.7
print("x=",x,"y=",y)
```

```
x=1.235
x= 1.235 y= 4.384725
```

Float variable(real numbers) reading, calculating by using method(function) and printing

```
def f(x):
    y=x*x-2.3*x+5.7
    return y;

x=float(input("x="))
print("x=",x,"y=",f(x))
```

```
runfile('E:/okul/SCO1/float2.py', wdir='E:/okul/SCO1')

x=1.235
x= 1.235 y= 4.384725
```

Float variable(real numers) reading, calculating by using method(function) and printing

```
def divide(x,y):
    z=x/y
    return z

x=float(input("x="))
y=float(input("y="))
print("x =",x,"y = ",y,"x/y = ",divide(x,y))
```

```
runfile('E:/okul/SCO1/float3.py', wdir='E:/okul/SCO1')

x=7

y=3
x = 7.0 y =  3.0 x/y =  2.3333333333333335
```

```
number1=float(input("number1 = "))
number2=float(input("number2 = "))
sum = number1+number2;
substract = number1-number2
multiply = number1*number2
divide=number1/number2
s=str(number1)+" +  "+str(number2)+" = "+str(sum)+"\n"
s+=str(number1)+" -  "+str(number2)+" = "+str(substract)+"\n"
s+=str(number1)+" x  "+str(number2)+" = "+str(multiply)+"\n"
s+=str(number1)+" /  "+str(number2)+" = "+str(divide)+"\n"
print(s)
```

```
runfile('E:/okul/SCO1/float4.py', wdir='E:/okul/SCO1')

number1 = 7.86

number2 = 2.31
7.86 +  2.31 = 10.17
7.86 -  2.31 = 5.550000000000001
7.86 x  2.31 = 18.1566
7.86 /  2.31 = 3.4025974025974026
```

Reading and writing string

```
name=input("what is your name =")
print("welcome to pyhon programming ",name)
```

```
runfile('E:/okul/SCO1/prog2.py', wdir='E:/okul/SCO1')

what is your name =turhan
welcome to pyhon programming  turhan
```

Reading and writing string by using method name()

```
def name():
    n=input("what is your name =")
    return n

print("welcome to pyhon programming ",name())
```

```
what is your name =turhan
welcome to pyhon programming  turhan
```

Reading and writing string by using method hello()

```
def hello():
    n=input("what is your name =")
    s="welcome to python programming "+n
    return s

print(hello())
```

```
runfile('E:/okul/SCO1/name3.py', wdir='E:/okul/SCO1')

what is your name =turhan
welcome to python programming turhan
```

Boolean variables:

Boolean variable tales values True/False. It takes only one bit binary to define a boolean number.

Boolean operators:

AND OR NOT

Boolean Logic table

First variable	**Second variable**	**AND**	**OR**
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False
First variable		**NOT**	
True		False	

False		True	

```
x=True
y=False
z=x and y
w=x or y
u=not x
print("x=",x,"y=",y,"z=",z,"w=",w,"u=",u)
```

```
runfile('E:/okul/SCO1/bool1.py', wdir='E:/okul/SCO1')
x= True y= False z= False w= True
```

```
def x_and_y(x,y):
    return x and y
def x_or_y(x,y):
    return x or y
def not_x(x):
    return not x
x=True
y=True
print("x=",x,"y=",y,"x and y=",x_and_y(x,y),"x or y=",x_or_y(x,y),"not x ",not_x(x))
x=True
y=False
print("x=",x,"y=",y,"x and y=",x_and_y(x,y),"x or y=",x_or_y(x,y),"not x ",not_x(x))
x=False
y=True
print("x=",x,"y=",y,"x and y=",x_and_y(x,y),"x or y=",x_or_y(x,y),"not x ",not_x(x))
x=False
y=False
print("x=",x,"y=",y,"x and y=",x_and_y(x,y),"x or y=",x_or_y(x,y),"not x ",not_x(x))
```

```
runfile('E:/okul/SCO1/bool2.py', wdir='E:/okul/SCO1')
x= True y= True x and y= True x or y= True not x  False
x= True y= False x and y= False x or y= True not x  False
x= False y= True x and y= False x or y= True not x  True
x= False y= False x and y= False x or y= False not x  True
```

Comparison of two numbers:

sign	**meaning**
>	**greater**
>=	**Grater and equal**
<	**smaller**
<=	**Smaller and equal**
==	**equal**
!=	**Not equal**

```
def greater(x,y):
    return x>y
def smaller(x,y):
    return x<y
def greater_and_equal(x,y):
    return x>=y
def smaller_and_equal(x,y):
    return x<=y
def equal(x,y):
    return x==y
def not_equal(x,y):
    return x!=y
x=int(input("x="))
y=int(input("x="))
print("x=",x,"y=",y,"x>y=",greater(x,y),"x<y=",smaller(x,y),"x>=y",greater_and_equal(x,y),"x<=y",smaller_and_equal(x,y),"equal",equa
l(x,y),"not equal ",not_equal(x,y))
```

```
runfile('E:/okul/SCO1/bool3.py', wdir='E:/okul/SCO1')

x=2

x=3
x= 2 y= 3 x>y= False x<y= True x>=y False x<=y True equal False not equal  True
```

```
runfile('E:/okul/SCO1/bool3.py', wdir='E:/okul/SCO1')
```

```
x=3
x=3
x= 3 y= 3 x>y= False x<y= False x>=y True x<=y True equal True not equal  False
```

Character variable type and Unicode characters

Character (char) variable type define actual characters in computers. In order to see characters in python **ISO Unicode** character standards are used. Unicode character code consist of 4 hexagonal (base 16) number. In case you are not familiar with orthogonal number system, a table of equiavalancy is given

Tablo Heksagonal(sixteen base), and 10 base and binary number systems equvalency

Heksagonal	10 base	2 base (binary)
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

```
x = int(input("Input an integer: "))
print("x ten base =",x,"x 2 base ",bin(x),"x octogonal base",oct(x),"x hexagonal base=",hex(x))
```

```
Input an integer: 15
x ten base = 15 x 2 base  0b1111 x octogonal base 0o17 x hexagonal base= 0xf
```

```
Input an integer: 12
x ten base = 12 x 2 base  0b1100 x octogonal base 0o14 x hexagonal base= 0xc
```

in ISO unicode '\u0041' defines character 'A'. or '\u03E1'. Or code '\u03E1' defines 'a'. When the first two number of unicode equals to 0, it defines ASCII characte codes. You can find very detailed list of characters in ISO unicode by looking the internet site **http:\\unicode.org**

In the small code sample below character A and 'α' is loaded to computer as char variables in python

```
A1='\u0041'
A2='A'
alpha1='\u03B1'
print(A1,A2,alpha1)
```

```
runfile('E:/okul/SCO1/char1.py', wdir='E:/okul/SCO1')
A A α
```

Table Some special control characters

Chracter sysmbol	**action**	**Unicode symbol**
\b	**Back one character**	**\u0008**
\t	**tab**	**\u0009**
\n	**ctrl-return**	**\u000A**
\"	**“ sign**	**\u0022**

\'	‘ **sign**	**\u0027**
\\	\ **sign**	**\u005C**

Some additional unicode characters are given in tables below. By using control characters several actions in print can be carried out such as back space, carriage-return, tab etc. When character variables are used, they can merge in strings, In fact strings are character arrays. When something with a character that does not available in your editor you can add that character into your String variable as char variable character.

```
b1='ü'
b2='ğ'
b3='ç'
b4='a'
b5='b'
b6='c'
s=""+b1+b2+b3+b4+b5+b6
print(s)
```

```
runfile('E:/okul/SCO1/char2.py', wdir='E:/okul/SCO1')
üğçabc
```

```
b1='\u03B1'
b2='\u03B2'
b3='\u03B3'
b4='\u03B4'
b5='\u03B5'
b6='\u03B6'
s=""+b1+b2+b3+b4+b5+b6
print(s)
```

```
runfile('E:/okul/SCO1/char3.py', wdir='E:/okul/SCO1')
αβγδεζ
```

```
b1='\u0394'
b2='\u00B2'
s=b1+"T = "+b1+"x"+b2+"+"+b1+"y"+b2
print(s)
```

```
runfile('E:/okul/SCO1/char1.py', wdir='E:/okul/SCO1')
ΔT = Δx²+Δy²
```

Table Some unicode character codes

	000	001	002	003	004	005	006	007
0	NUL 0000	DLE 0010	SP 0020	0 0030	@ 0040	P 0050	` 0060	p 0070
1	SOH 0001	DC1 0011	! 0021	1 0031	A 0041	Q 0051	a 0061	q 0071
2	STX 0002	DC2 0012	" 0022	2 0032	B 0042	R 0052	b 0062	r 0072
3	ETX 0003	DC3 0013	# 0023	3 0033	C 0043	S 0053	c 0063	s 0073
4	EOT 0004	DC4 0014	$ 0024	4 0034	D 0044	T 0054	d 0064	t 0074
5	ENQ 0005	NAK 0015	% 0025	5 0035	E 0045	U 0055	e 0065	u 0075
6	ACK 0006	SYN 0016	& 0026	6 0036	F 0046	V 0056	f 0066	v 0076
7	BEL 0007	ETB 0017	' 0027	7 0037	G 0047	W 0057	g 0067	w 0077
8	BS 0008	CAN 0018	(0028	8 0038	H 0048	X 0058	h 0068	x 0078
9	HT 0009	EM 0019	) 0029	9 0039	I 0049	Y 0059	i 0069	y 0079
A	LF 000A	SUB 001A	* 002A	: 003A	J 004A	Z 005A	j 006A	z 007A
B	VT 000B	ESC 001B	+ 002B	; 003B	K 004B	[005B	k 006B	{ 007B
C	FF 000C	FS 001C	, 002C	< 003C	L 004C	\ 005C	l 006C	| 007C
D	CR 000D	GS 001D	- 002D	= 003D	M 004D	] 005D	m 006D	} 007D
E	SO 000E	RS 001E	. 002E	>>003E	N 004E	^ 005E	n 006E	~ 007E
F	SI 000F	US 001F	/ 002F	? 003F	O 004F	_ 005F	o 006F	DEL 007F

	008	009	00A	00B	00C	00D	00E	00F
0	XXX 0080	DCS 0090	NB SP 00A0	° 00B0	À 00C0	Ð 00D0	à 00E0	ð 00F0
1	XXX 0081	PU1 0091	¡ 00A1	± 00B1	Á 00C1	Ñ 00D1	á 00E1	ñ 00F1
2	BPH 0082	PU2 0092	¢ 00A2	2 00B2	Â 00C2	Ò 00D2	â 00E2	ò 00F2
3	NBH 0083	STS 0093	£ 00A3	3 00B3	Ã 00C3	Ó 00D3	ã 00E3	ó 00F3
4	IND 0084	CCH 0094	¤ 00A4	´ 00B4	Ä 00C4	Ô 00D4	ä 00E4	ô 00F4
5	NEL 0085	MW 0095	¥ 00A5	µ 00B5	Å 00C5	Õ 00D5	å 00E5	õ 00F5
6	SSA 0086	SPA 0096	¦ 00A6	¶ 00B6	Æ 00C6	Ö 00D6	æ 00E6	ö 00F6
7	ESA 0087	EPA 0097	§ 00A7	· 00B7	Ç 00C7	× 00D7	ç 00E7	÷ 00F7
8	HTS 0088	SOS 0098	¨ 00A8	¸ 00B8	È 00C8	Ø 00D8	è 00E8	ø 00F8
9	HTJ 0089	XXX 0099	© 00A9	1 00B9	É 00C9	Ù 00D9	é 00E9	ù 00F9
A	VTS 008A	SCI 009A	ª 00AA	º 00BA	Ê 00CA	Ú 00DA	ê 00EA	ú 00FA
B	PLD 008B	CSI 009B	« 00AB	» 00BB	Ë 00CB	Û 00DB	ë 00EB	û 00FB
C	PLU 008C	ST 009C	¬ 00AC	¼ 00BC	Ì 00CC	Ü 00DC	ì 00EC	ü 00FC
D	RI 008D	OSC 009D	SHY 00AD	½ 00BD	Í 00CD	Ý 00DD	í 00ED	ý 00FD
E	SS2 008E	PM 009E	® 00AE	¾ 00BE	Î 00CE	Þ 00DE	î 00EE	þ 00FE
F	SS3 008F	APC 009F	¯ 00AF	¿ 00BF	Ï 00CF	ß 00DF	ï 00EF	ÿ 00FF

0180 **Latin Extended-B** **024**

	018	019	01A	01B	01C	01D	01E	01F	020	021	022	023	024
0	ƀ 0180	Ɛ 0190	Ơ 01A0	ư 01B0	ǀ 01C0	ǐ 01D0	Ǎ 01E0	ǰ 01F0	Ȁ 0200	Ȑ 0210	Ȓ 0220	Ȍ 0230	ȝ 0240
1	Ɓ 0181	Ƒ 0191	ơ 01A1	Ʊ 01B1	ǁ 01C1	Ǒ 01D1	ǡ 01E1	DZ 01F1	ȁ 0201	ȑ 0211	ȓ 0221	ȍ 0231	ȞȞ 0241
2	Ƃ 0182	ƒ 0192	Ƣ 01A2	Ʋ 01B2	ǂ 01C2	ǒ 01D2	Ǣ 01E2	Dz 01F2	Ȃ 0202	Ȓ 0212	Ȣ 0222	Ȏ 0232	ȟ 0242
3	ƃ 0183	Ɠ 0193	ƣ 01A3	Ʒ 01B3	! 01C3	Ǔ 01D3	ǣ 01E3	dz 01F3	ȃ 0203	ȓ 0213	ȣ 0223	ȏ 0233	Ƀ 0243
4	Ƅ 0184	Ɣ 0194	Ƥ 01A4	ʒ 01B4	DŽ 01C4	ǔ 01D4	Ǥ 01E4	Ǵ 01F4	Ȅ 0204	Ȕ 0214	Ȥ 0224	ȐȐ 0234	Ʉ 0244
5	ƅ 0185	ƕ 0195	ƥ 01A5	Ƶ 01B5	Dž 01C5	Ǖ 01D5	ǥ 01E5	ǵ 01F5	ȅ 0205	ȕ 0215	ȥ 0225	ȑȑ 0235	Ʌ 0245
6	Ɔ 0186	ƚ 0196	Ʀ 01A6	ƶ 01B6	dž 01C6	ǖ 01D6	Ǧ 01E6	Ƕ 01F6	Ȇ 0206	Ȗ 0216	Ȧ 0226	ȓȓ 0236	Ɇ 0246
7	Ƈ 0187	ƛ 0197	Ƨ 01A7	Ʒ 01B7	LJ 01C7	Ǘ 01D7	ǧ 01E7	Ƿ 01F7	ȇ 0207	ȗ 0217	ȧ 0227	ȝȝ 0237	ɇ 0247
8	ƈ 0188	Ƙ 0198	ƨ 01A8	Ʒ 01B8	Lj 01C8	ǘ 01D8	Ǩ 01E8	Ǹ 01F8	Ȉ 0208	Ș 0218	Ȩ 0228	ȣȣ 0238	Ɉ 0248
9	Ɖ 0189	ƙ 0199	Ʃ 01A9	ƹ 01B9	lj 01C9	Ǚ 01D9	ǩ 01E9	ǹ 01F9	ȉ 0209	ș 0219	ȩ 0229	ȥȥ 0239	ɉ 0249
A	Ɗ 018A	ƚ 019A	ƺ 01AA	ƺ 01BA	NJ 01CA	ǚ 01DA	Ǫ 01EA	Ǻ 01FA	Ȋ 020A	Ț 021A	Ȫ 022A	Ȧȧ 023A	ɋ 024A
B	Ƌ 018B	ƛ 019B	ƫ 01AB	ƻ 01BB	Nj 01CB	Ǔǔ 01DB	ǫ 01EB	ǻ 01FB	ȋ 020B	ț 021B	ȫ 022B	Ȣȣ 023B	ɋ 024B
C	ƌ 018C	Ɯ 019C	Ƭ 01AC	ƽ 01BC	nj 01CC	ǜ 01DC	Ǫǫ 01EC	Ǽ 01FC	Ȍ 020C	ȝ 021C	Ȭ 022C	ȣ 023C	Ʀ 024C
D	ƍ 018D	Ɲ 019D	ƭ 01AD	ƾ 01BD	Ǎ 01CD	ǝ 01DD	ǫǫ 01ED	ǽ 01FD	ȍ 020D	ȝ 021D	ȭ 022D	Ł 023D	ɍ 024D
E	Ǝ 018E	ƞ 019E	Ʈ 01AE	ƿ 01BE	ǎ 01CE	Ǎǎ 01DE	Ǯ 01EE	Ǿ 01FE	Ȏ 020E	Ȟ 021E	Ȯ 022E	ƮƮ 023E	ʏ 024E
F	Ə 018F	Ɵ 019F	Ư 01AF	ƿ 01BF	Ǐ 01CF	ǎǎ 01DF	ǯ 01EF	ǿ 01FF	ȏ 020F	ȟ 021F	ȯ 022F	ȣ 023F	ɏ 024F

0370 **Greek and Coptic** **03FF**

	037	038	039	03A	03B	03C	03D	03E	03F
0	Ͱ 0370		ΐ 0390	Π 03A0	ΰ 03B0	π 03C0	ϐ 03D0	Ϣ 03E0	ϰ 03F0
1	ͱ 0371		Α 0391	Ρ 03A1	α 03B1	ρ 03C1	ϑ 03D1	ϣ 03E1	ϱ 03F1
2	Ͳ 0372		Β 0392		β 03B2	ς 03C2	ϒ 03D2	Ϥ 03E2	ϲ 03F2
3	ͳ 0373		Γ 0393	Σ 03A3	γ 03B3	σ 03C3	ϒ 03D3	ϥ 03E3	ϳ 03F3
4	ʹ 0374	΄ 0384	Δ 0394	Τ 03A4	δ 03B4	τ 03C4	ϓ 03D4	Ϧ 03E4	ϴ 03F4
5	͵ 0375	΅ 0385	Ε 0395	Υ 03A5	ε 03B5	υ 03C5	ϕ 03D5	ϥ 03E5	ϵ 03F5
6	Ͷ 0376	Ά 0386	Ζ 0396	Φ 03A6	ζ 03B6	φ 03C6	ϖ 03D6	ϧ 03E6	϶ 03F6
7	ͷ 0377	· 0387	Η 0397	Χ 03A7	η 03B7	χ 03C7	ϗ 03D7	ϩ 03E7	Ϸ 03F7
8		Έ 0388	Θ 0398	Ψ 03A8	θ 03B8	ψ 03C8	Ϙ 03D8	Ϫ 03E8	ϸ 03F8
9		Ή 0389	Ι 0399	Ω 03A9	ι 03B9	ω 03C9	ϙ 03D9	ϫ 03E9	Ϲ 03F9
A	ͺ 037A	Ί 038A	Κ 039A	Ϊ 03AA	κ 03BA	ϊ 03CA	Ϛ 03DA	Ϭ 03EA	Ϻ 03FA
B	ͻ 037B		Λ 039B	Ϋ 03AB	λ 03BB	ϋ 03CB	ϛ 03DB	ϭ 03EB	ϻ 03FB
C	ͼ 037C	Ό 038C	Μ 039C	ά 03AC	μ 03BC	ό 03CC	Ϝ 03DC	Ϯ 03EC	ϼ 03FC
D	ͽ 037D		Ν 039D	έ 03AD	ν 03BD	ύ 03CD	ϝ 03DD	ϯ 03ED	Ͻ 03FD
E	; 037E	Ύ 038E	Ξ 039E	ή 03AE	ξ 03BE	ώ 03CE	Ϟ 03DE	ϰ 03EE	Ͼ 03FE
F		Ώ 038F	Ο 039F	ί 03AF	ο 03BF	Ϗ 03CF	ϟ 03DF	ϱ 03EF	Ͽ 03FF

	220	221	222	223	224	225	226	227	228	229	22A	22B	22C	22D	22E	22F
0	∀ 2200	∐ 2210	∠ 2220	∭ 2230	≀ 2240	≐ 2250	≠ 2260	≮ 2270	≺ 2280	⊐ 2290	⊠ 22A0	≻ 22B0	∧ 22C0	⋐ 22D0	≰ 22E0	⋰ 22F0
1	∁ 2201	∑ 2211	∡ 2221	∮ 2231	≁ 2241	≑ 2251	≡ 2261	≯ 2271	≻ 2281	⊏ 2291	⊡ 22A1	≽ 22B1	∨ 22C1	⋑ 22D1	≱ 22E1	⋱ 22F1
2	∂ 2202	− 2212	∢ 2222	∯ 2232	≂ 2242	≒ 2252	≢ 2262	≲ 2272	⊂ 2282	⊒ 2292	⊢ 22A2	⊲ 22B2	∩ 22C2	⋒ 22D2	⋢ 22E2	∈ 22F2
3	∃ 2203	∓ 2213	∣ 2223	∰ 2233	≃ 2243	≓ 2253	≡ 2263	≳ 2273	⊃ 2283	⊓ 2293	⊣ 22A3	⊳ 22B3	∪ 22C3	⋓ 22D3	⋣ 22E3	∉ 22F3
4	∄ 2204	∔ 2214	∤ 2224	∴ 2234	≠ 2244	≔ 2254	≤ 2264	≴ 2274	⊄ 2284	⊔ 2294	⊤ 22A4	⊴ 22B4	⋄ 22C4	⋔ 22D4	⋤ 22E4	∈ 22F4
5	∅ 2205	∕ 2215	∥ 2225	∵ 2235	≅ 2245	≕ 2255	≥ 2265	≵ 2275	⊅ 2285	⊕ 2295	⊥ 22A5	⊵ 22B5	⋅ 22C5	⋕ 22D5	⋥ 22E5	∈̇ 22F5
6	∆ 2206	∖ 2216	∦ 2226	∶ 2236	≆ 2246	≖ 2256	≦ 2266	≶ 2276	⊆ 2286	⊖ 2296	⊢ 22A6	⊶ 22B6	⋆ 22C6	⋖ 22D6	≤ 22E6	∈̄ 22F6
7	∇ 2207	∗ 2217	∧ 2227	∷ 2237	≇ 2247	≙ 2257	≧ 2267	≷ 2277	⊇ 2287	⊗ 2297	⊨ 22A7	⊷ 22B7	⋇ 22C7	⋗ 22D7	≥ 22E7	∈̄ 22F7
8	∈ 2208	∘ 2218	∨ 2228	∸ 2238	≈ 2248	≚ 2258	≨ 2268	≸ 2278	⊈ 2288	⊘ 2298	⊨ 22A8	⊸ 22B8	⋈ 22C8	⋘ 22D8	≤ 22E8	∈ 22F8
9	∉ 2209	∙ 2219	∩ 2229	∺ 2239	≉ 2249	≜ 2259	≩ 2269	≹ 2279	⊉ 2289	⊙ 2299	⊩ 22A9	⊹ 22B9	⋉ 22C9	⋙ 22D9	≥ 22E9	∈ 22F9
A	∈ 220A	√ 221A	∪ 222A	∻ 223A	≊ 224A	≟ 225A	≪ 226A	≺ 227A	⊊ 228A	⊚ 229A	⊪ 22AA	⊤ 22BA	⋊ 22CA	≦ 22DA	⋪ 22EA	∋ 22FA
B	∋ 220B	∛ 221B	∫ 222B	÷ 223B	≋ 224B	≛ 225B	≫ 226B	≻ 227B	⊋ 228B	⊛ 229B	⊫ 22AB	⊻ 22BB	≻ 22CB	≧ 22DB	⋫ 22EB	⋑ 22FB
C	∌ 220C	∜ 221C	∬ 222C	∼ 223C	≌ 224C	≜ 225C	⧫ 226C	≼ 227C	⊎ 228C	⊜ 229C	⊬ 22AC	⊼ 22BC	≺ 22CC	≤ 22DC	⋬ 22EC	∋ 22FC
D	∋ 220D	∝ 221D	∭ 222D	∽ 223D	≍ 224D	def ≝ 225D	≠ 226D	≽ 227D	⊎ 228D	⊖ 229D	⊭ 22AD	⊽ 22BD	≃ 22CD	≥ 22DD	⋭ 22ED	⋑ 22FD
E	■ 220E	∞ 221E	∮ 222E	≈ 223E	≎ 224E	m ≞ 225E	≮ 226E	≾ 227E	⊎ 228E	⊞ 229E	⊯ 22AE	⊾ 22BE	∨ 22CE	≤ 22DE	⋮ 22EE	∋ 22FE
F	∏ 220F	∟ 221F	∯ 222F	∿ 223F	≏ 224F	? ≟ 225F	≯ 226F	≿ 227F	⊏ 228F	⊟ 229F	⊮ 22AF	⊿ 22BF	∧ 22CF	≥ 22DF	⋯ 22EF	⊏ 22FF

Binary , octagonal and hexagonal integers

Integer numbers can be expressed either in 10 digit form we are familiar with or binary form which is what computer understand or octagonal form and hexagonal form which basically derived from binary bit combinations (octagonal=8 digit=2^3, hexagonal=16 digit=2^4) If you want to output a 10 base integer into binary, octagonal or hexagonal form use bin,oct and hex method

```
x=256
print("x=",x,"binary ",bin(x),"octagonal ",oct(x),"hexagonal ",hex(x))
```

```
runfile('E:/okul/SCO1/greek_char.py', wdir='E:/okul/SCO1')
x= 256 binary  0b100000000 octagonal  0o400 hexagonal  0x100
```

We can enter a binary integer by using 0b in front, for example 0b001101, 0o for octogonal number, for example 0o134, 0x for hexagonal number for example 0x12af

```
x=134
x1=0b001101
x2=0o134
x3=0x12af
print("x=",x,"binary ",bin(x),"octagonal ",oct(x),"hexagonal ",hex(x))
print("x1=",x1,"binary ",bin(x1),"octagonal ",oct(x1),"hexagonal ",hex(x1))
print("x12",x2,"binary ",bin(x2),"octagonal ",oct(x2),"hexagonal ",hex(x2))
print("x3=",x3,"binary ",bin(x3),"octagonal ",oct(x3),"hexagonal ",hex(x3))
```

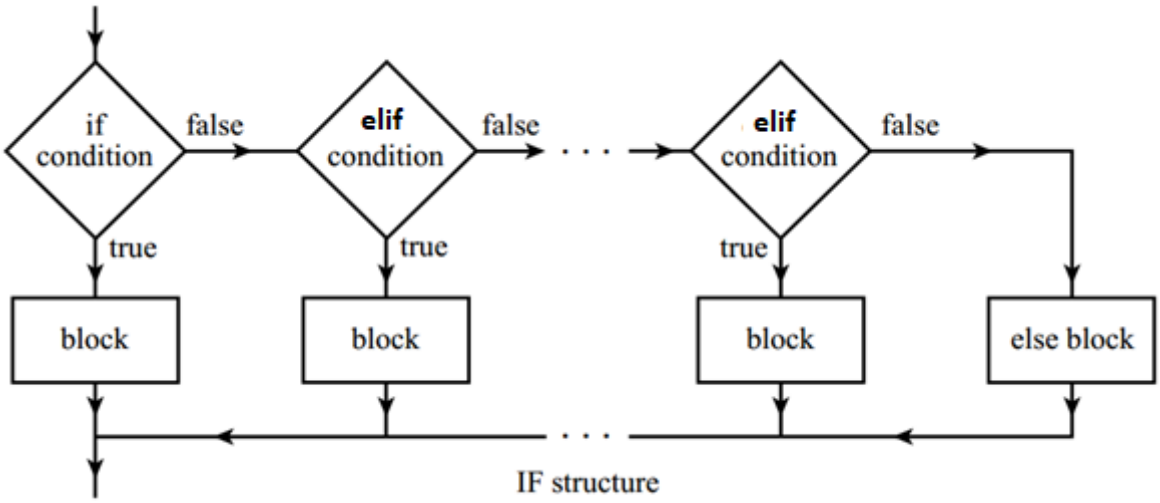
```
runfile('E:/okul/SCO1/greek_char.py', wdir='E:/okul/SCO1')
x= 134 binary  0b10000110 octagonal  0o206 hexagonal  0x86
x1= 13 binary  0b1101 octagonal  0o15 hexagonal  0xd
x12 92 binary  0b1011100 octagonal  0o134 hexagonal  0x5c
```

1.3 IF STATEMENT

When a decision should be made by a computer program, if control structure used. If structure has the following form :

an example of boolean variable

if boolean variable 1:
    executes statements here if boolean variable 1 is true
elif boolean variable 2:
  executes statements here if boolean variable 2 is true
elif (boolean variable 3)
   executes statements here if boolean variable 3 is true
……. (more else if statements are possible)
Else:
  executes statements here if none of the above boolean variables are true



IF structure

In our first example program name and grade of a students read and process the grade according to number grade.

```
name=input("Enter student name=")
grade=float(input("enter student\'s grade="))
s1="Grade of "+name+" is "
if grade>=90: s1=s1+"A"
elif grade>=75: s1=s1+"B"
elif grade>=60: s1=s1+"C"
elif grade>=50: s1=s1+"D"
elif grade>=40: s1=s1+"E"
else:veli
78  s1=s1+"F"
print(s1);
```

```
runfile('E:/okul/SCO1/if5.py', wdir='E:/okul/SCO1')

Enter student name=ali

enter student's grade=16
Grade of ali is F

runfile('E:/okul/SCO1/if5.py', wdir='E:/okul/SCO1')

Enter student name=veli

enter student's grade=78
Grade of veli is B
```

```
x=float(input("x="))
if x<2: y=x*x-2
elif x>=2 & x<4: y=x+5.5
else:   y=3
print("x=",x,"y=",y)
```

```
runfile('E:/okul/SCO1/if1.py', wdir='E:/okul/SCO1')

x=1.5
x= 1.5 y= 0.25
```

```
wheather=input("how is weather to day?")
x1=wheather=="rainy"
x2=wheather=="sunny"
x3=wheather=="cloudy"
if x1: action="stay at home"
elif x2: action="go to beach"
elif x3: action="study"
else: action="do nothing"
print(action)
```

```
runfile('E:/okul/SCO1/wheather.py', wdir='E:/okul/SCO1')

how is weather to day?sunny
go to beach
```

```
def compare(x,y):
    if x<y: s=str(x)+" is smaller than "+str(y)
    elif x>y: s=str(x)+" is bigger than "+str(y)
    elif x==y: s=str(x)+" is equal to "+str(y)
    return s

x=float(input("x="))
y=float(input("x="))
print(compare(x,y))
```

```
runfile('E:/okul/SCO1/bool3.py', wdir='E:/okul/SCO1')

x=7.35

x=2.23
7.35 is bigger than 2.23
```

```
def guess_color(s):
    b1=s=="red"
    if b1: s1="Correct guess color you entered is red"
    else:  s1="Color you entered is npt red, it is "+s
    return s1

x=input("guess the color =")
print(guess_color(x))
```

```
runfile('E:/okul/SCO1/bool4.py', wdir='E:/okul/SCO1')

guess the color =red
Correct guess color you entered is red
```

Nested if

```
x=float(input("x = "))
y=float(input("y = "))
if x>5:
    if y>5: s="x and y are greater than 5"
    elif y==5: s="x is greater that 5 and y is equal to 5"
    else: s="x is greater that 5 and y is smaller than 5"
else:
    if y>5: s="x is smaller or equal to 5 and  y is greater than 5"
    elif y==5: s="x is smaller or equal to 5 and y is equal to 5"
    else: s="x is smaller or equal to 5 and y is smaller than 5"
print(s)
```

```
runfile('E:/okul/SCO1/bool5.py', wdir='E:/okul/SCO1')

x = 1.23

y = 2.35
x is smaller or equal to 5 and y is smaller than 5
```

Composite if

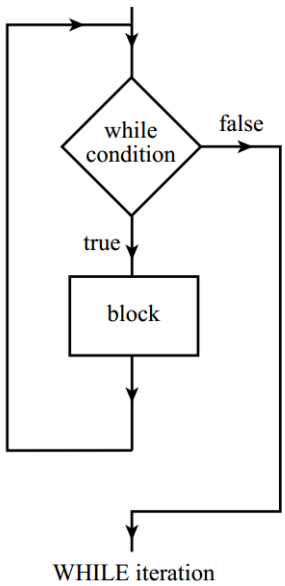
```
x=float(input("x = "))
y=float(input("y = "))
if x>5 and y>5: s="x and y are greater than 5"
elif x>5 and y==5: s="x is greater that 5 and y is equal to 5"
elif x>5 and y<5: s="x is greater that 5 and y is smaller than 5"
elif x<=5 and y>5: s="x is smaller or equal to 5 and  y is greater than 5"
elif x<=5 and y==5: s="x is smaller or equal to 5 and y is equal to 5"
elif x<=5 and y>5: s="x is smaller or equal to 5 and y is smaller than 5"
print(s)
```

```
runfile('E:/okul/SCO1/bool6.py', wdir='E:/okul/SCO1')

x = 3
y = 2
x is smaller or equal to 5 and y is smaller than 5
```

1.4 FOR AND WHILE STATEMENTS

In the if structure of the previous section, it is seen that if the given boolean value is true, computer carries out instruction inside of the if section. While structure is similar to if structure, with one difference: In the if statement inside of the statement structure instructions will be carried out once. In while statements, it will go inside of while statement as repeatedly, as long as the boolean is true.

while boolean variable:
processes while boolean variable is true




```
def sum(n):
    total=0;
```

```
    i=1
    while i<(n+1):
        total=total+i;
        i+=1
    return total;

n=int(input("n = "));
r=sum(n);
print("total=",r);
```

```
n = 100
total= 5050
```

For loop structure is also similar to while structure. Except initial condition and incrimental conditions are also given in the structure

for initial variable **in** a vector
  body

A set of numbers (or any type of varibles) is called lists. Lists are given within [] and seperated with ,
A set of numbers(or any type of variables) is called tupples . Tupples are given within () and seperated with ,
Basic difference of list and tuple is: elements of list can be changed. Elements of tuple can not be changed define only ones and then used as is. In math, List and tuple is represent a **vector**.

```
list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]

print(list)          # Prints complete list
print(list[0])       # Prints first element of the list
print(list[1:3])     # Prints elements starting from 2nd till 3rd
print(list[2:])      # Prints elements starting from 3rd element
```

```
runfile('E:/okul/SCO1/list.py', wdir='E:/okul/SCO1')
['abcd', 786, 2.23, 'john', 70.2]
abcd
[786, 2.23]
[2.23, 'john', 70.2]
```

```
tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )

print(tuple)          # Prints complete list
print(tuple[0])       # Prints first element of the list
print(tuple[1:3])     # Prints elements starting from 2nd till 3rd
print(tuple[2:])      # Prints elements starting from 3rd element
```

```
runfile('E:/okul/SCO1/list.py', wdir='E:/okul/SCO1')
('abcd', 786, 2.23, 'john', 70.2)
abcd
(786, 2.23)
(2.23, 'john', 70.2)
```

There are buit in functions to be used with the lists

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position

pop()	Removes the element at the specified position
remove()	Removes the first item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

```
list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]

print(list)          # Prints complete list
print(list[0])       # Prints first element of the list
print(list[1:3])     # Prints elements starting from 2nd till 3rd
print(list[2:])      # Prints elements starting from 3rd element
list.append("xxx")
print(list)
list1=[1,2,3]
list.extend(list1)
print(list)
list.pop(5)
print(list)
```

```
runfile('E:/okul/SCO1/list_test1.py', wdir='E:/okul/SCO1')
['abcd', 786, 2.23, 'john', 70.2]
abcd
[786, 2.23]
[2.23, 'john', 70.2]
['abcd', 786, 2.23, 'john', 70.2, 'xxx']
['abcd', 786, 2.23, 'john', 70.2, 'xxx', 1, 2, 3]
['abcd', 786, 2.23, 'john', 70.2, 1, 2, 3]
```

vector can be defined y using range method or given directly

```
y=[1,2,3,5,7,9,11]
i=0
for x in y:
   print("i=",i,"x=",x)
   i=i+1
```

```
runfile('E:/okul/SCO1/for3.py', wdir='E:/okul/SCO1')
i= 0 x= 1
i= 1 x= 2
i= 2 x= 3
i= 3 x= 5
i= 4 x= 7
i= 5 x= 9
i= 6 x= 11
```

```
y=(1,2,3,5,7,9,11)
i=0
for x in y:
   print("i=",i,"x=",x)
   i=i+1
```

```
runfile('E:/okul/SCO1/list.py', wdir='E:/okul/SCO1')
i= 0 x= 1
i= 1 x= 2
i= 2 x= 3
i= 3 x= 5
i= 4 x= 7
i= 5 x= 9
i= 6 x= 11
```

```
y=["What's"," Up"," Docs"]
i=0
for x in y:
    print("i=",i,"x=",x)
    i=i+1
```

```
runfile('E:/okul/SCO1/for4.py', wdir='E:/okul/SCO1')
i= 0 x= What's
i= 1 x=  Up
i= 2 x=  Docs
```

Range method can defined vectors

range(3) defines [0,1,2]
range(1,4) defines [1,2,3]
range(0,2,10) defines [0,2,4,6,8]
range(5,-1,0) defines [5,4,3,2,1]

```
i=0
for x in range(5):
    print("i=",i,"x=",x)
    i=i+1
```

```
runfile('E:/okul/SCO1/for5.py', wdir='E:/okul/SCO1')
i= 0 x= 0
i= 1 x= 1
i= 2 x= 2
i= 3 x= 3
i= 4 x= 4
```

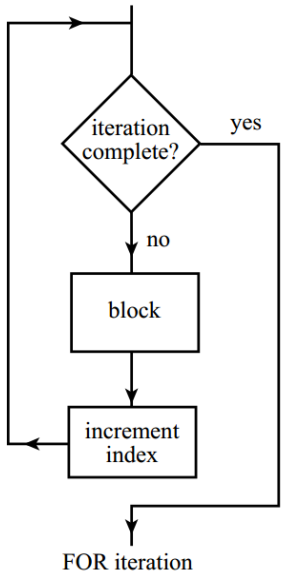
```
i=0
y=range(5)
for x in y:
    print("i=",i,"x=",x)
    i=i+1
```

```
runfile('E:/okul/SCO1/for5.py', wdir='E:/okul/SCO1')
i= 0 x= 0
i= 1 x= 1
i= 2 x= 2
i= 3 x= 3
i= 4 x= 4
```

A matrix(double array) example

```
m=3
n=5
x= [[0 for j in range(n)] for i in range(m)]
s=""
s1=""
s2=""
for i in range(m):
    for j in range(n):
        s="("+str(i)+str(j)+") "
        s1=str(i)+str(j)
        s2=s2+s
        x[i][j]=int(s1)
    s2=s2+"\n"
print("x = ")
print(x)
print("s2 = ")
print(s2)
print(x)
```

```
x =
[[0, 1, 2, 3, 4], [10, 11, 12, 13, 14], [20, 21, 22, 23, 24]]
s2 =
(00) (01) (02) (03) (04)
(10) (11) (12) (13) (14)
(20) (21) (22) (23) (24)

[[0, 1, 2, 3, 4], [10, 11, 12, 13, 14], [20, 21, 22, 23, 24]]
```




```
 from math import *

def sum(n):
    total=0;
    for i in range(1,(n+1)):
        total=total+i;
    return total;

n=int(input("n = "));
r=sum(n);
print("total=",r);
```

```
from math import *

def average(x):
    n=len(x);
    total=0;
    i=1;
    for i in range(0,n):
        total=total+x[i];
    return total/n;

x=[1.1179,2.21,3.31,4.41,5.51,6.61];
r=average(x);
print ("average=",r);
```

```
a=["ali","veli","49","elli"]
for i in a:
    print("i =",i)
```

```
runfile('E:/okul/SCO1/sum.py', wdir='E:/okul/SCO1')
i = ali
i = veli
i = 49
i = elli
```

```
a=[1.0,3.5,7.5,9.1,10.5]
sum=0
for i in a:
    sum+=i
    print("i =",i,"sum=",sum)
```

```
runfile('E:/okul/SCO1/sum.py', wdir='E:/okul/SCO1')
i = 1.0 sum= 1.0
i = 3.5 sum= 4.5
```

```
i = 7.5 sum= 12.0
i = 9.1 sum= 21.1
i = 10.5 sum= 31.6
```

List of greek characters

```
c1=ord('\u03B1')
c2=ord('\u03D2')
s=""
for i in range(c1,c2):
    s=s+" "+chr(i)
print(s)
```

```
runfile('E:/okul/SCO1/greek_char.py', wdir='E:/okul/SCO1')
 α β γ δ ε ζ η θ ι κ λ μ ν ξ ο π ρ ς σ τ υ φ χ ψ ω ϊ ϋ ό ύ ώ Ϗ ϐ ϑ
```

ASCII characters

```
c1=ord('\u0021')
c2=ord('\u007E')
s=""
for i in range(c1,c2):
    s=s+" i: "+str(i)+" "+chr(i)
print(s)
```

```
runfile('E:/okul/SCO1/greek_char.py', wdir='E:/okul/SCO1')
 i: 33 ! i: 34 " i: 35 # i: 36 $ i: 37 % i: 38 & i: 39 ' i: 40 ( i: 41 ) i: 42 * i: 43 + i: 44 , i: 45 - i: 46 . i: 47 / i: 48 0 i: 49 1 i: 50 2 i: 51 3 i: 52 4 i: 53 5 i: 54 6 i: 55 7 i: 56 8 i: 57 9 i: 58 : i: 59 ; i: 60 < i: 61 = i: 62 > i: 63 ? i: 64 @ i: 65 A i: 66 B i: 67 C i: 68 D i: 69 E i: 70 F i: 71 G i: 72 H i: 73 I i: 74 J i: 75 K i: 76 L i: 77 M i: 78 N i: 79 O i: 80 P i: 81 Q i: 82 R i: 83 S i: 84 T i: 85 U i: 86 V i: 87 W i: 88 X i: 89 Y i: 90 Z i: 91 [ i: 92 \ i: 93 ] i: 94 ^ i: 95 _ i: 96 ` i: 97 a i: 98 b i: 99 c i: 100 d i: 101 e i: 102 f i: 103 g i: 104 h i: 105 i i: 106 j i: 107 k i: 108 l i: 109 m i: 110 n i: 111 o i: 112 p i: 113 q i: 114 r i: 115 s i: 116 t i: 117 u i: 118 v i: 119 w i: 120 x i: 121 y i: 122 z i: 123 { i: 124 | i: 125 }
```

1.5 CLASS CONCEPT IN PYTHON

```
class Person:
  def __init__(self, name, age,ssno):
    self.name = name
    self.age = age
    self.ssno=ssno

  def print(self):
      print("name = ",self.name,"age=",self.age,"social security number = ",self.ssno)

p1 = Person("John", 36,20321)
p2=Person("Daphne",18,65655)
p1.print()
p2.print()
```

```
runfile('E:/okul/SCO1/Person.py', wdir='E:/okul/SCO1')
name =  John age= 36 social security number =  20321
name =  Daphne age= 18 social security number =  65655
```

```
class Box:
  def __init__(self, length,width,height,color):
    self.length = length
    self.width = width
    self.height=height
    self.color=color
  def volume(self):
      return self.length*self.width*self.height
  def area(self):
      return 2.0*(self.length*self.width+self.length*self.height+self.width*self.height)

  def print(self):
      print("length = ",self.length,"width=",self.width,"height = ",self.height,"color = ",self.color,"volume =
",self.volume(),"area=",self.area())

p1 = Box(1.0,1.2,0.4,"blue")
p2=Box(1.2,1.2,1.0,"yellow")
p1.print()
p2.print()
```

```
runfile('E:/okul/SCO1/Box.py', wdir='E:/okul/SCO1')
length =  1.0 width= 1.2 height =  0.4 color =  blue volume =  0.48 area= 4.16
length =  1.2 width= 1.2 height =  1.0 color =  yellow volume =  1.44 area= 7.68
```

```
class redBox:
```

```
  def __init__(self, length,width,height):
    self.p = Box(length,width,height,"red")

  def print(self):
      print("length = ",self.p.length,"width=",self.p.width,"height = ",self.p.height,"color = ",self.p.color,"volume =
",self.p.volume(),"area=",self.p.area())

p1 = redBox(1.0,1.2,0.4)
p2=redBox(1.2,1.2,1.0)
p1.print()
p2.print()
```

```
runfile('E:/okul/SCO1/redBox.py', wdir='E:/okul/SCO1')
length =  1.0 width= 1.2 height =  0.4 color =  red volume =  0.48 area= 4.16
length =  1.2 width= 1.2 height =  1.0 color =  red volume =  1.44 area= 7.68
```

```
from math import *
class robot:
  def __init__(self, name):
    self.name = name
    self.R=0.0
    self.theta=0.0
  def turn_north(self):
      self.theta=pi/2.0
  def turn_south(self):
      self.theta=3.0*pi/2.0
  def turn_west(self):
      self.theta=pi
  def turn_east(self):
      self.theta=0.0
  def turn(self,angle):
      self.theta=angle
  def forward(self):
      self.R=self.R+1.0
  def forward(self,Ri):
      self.R=self.R+Ri
  def backward(self):
      self.R=self.R-1.0
  def backward(self,Ri):
      self.R=self.R-Ri
  def print(self):
      print("name = ",self.name,"R=",self.R,"theta=",self.theta)

p1 = robot("R2D2")
p1.turn_north()
p1.forward(2.5)
p1.print()
```

```
runfile('E:/okul/SCO1/robot.py', wdir='E:/okul/SCO1')
name =  R2D2 R= 2.5 theta= 1.5707963267948966
```

Class inheritance:

```
class yellowBox(Box):
  def __init__(self, length,width,height):
    self.length=length
    self.width=width
    self.height=height
    self.color="yellow"

p1 = yellowBox(1.0,1.2,0.4)
p2 = yellowBox(1.2,1.2,1.0)
p1.print()
p2.print()
```

```
runfile('E:/okul/SCO1/yellowBox.py', wdir='E:/okul/SCO1')
length =  1.0 width= 1.2 height =  0.4 color =  yellow volume =  0.48 area= 4.16
length =  1.2 width= 1.2 height =  1.0 color =  yellow volume =  1.44 area= 7.68
```

1.6 ABSTRACT CLASS

An abstract class or interface is an empty class for reference purposes, although they can have additional methods. Most of the modern third generation languages such as java, C++, python has abstract class and/or interface buid into the language. Lambda variables a short-cut definition to define a new class derived from an interface. This concepts are very useful for us in numerical methods, because function definitions in our

numerical codes become independent of actual function. As an input function an object of abstract class or interface can be defined, and when we will use the numerial analysis codes (suc as root finding, integration etc.) actual function can be defined as derived form abstract/class and/or interface. Abstract class/interface examples will be given here for simple function f(x). Please note that inside of these classes some additional codes for derivatives, root finding methods and integrals might exist. In order to see details of these codes please refer to related chapters in this book.

```
# Python program showing
# abstract base class work

from abc import ABC, abstractmethod

class Polygon(ABC):

    @abstractmethod
    def noofsides(self):
        pass

class Triangle(Polygon):

    # overriding abstract method
    def noofsides(self):
        print("I have 3 sides")

class Pentagon(Polygon):

    # overriding abstract method
    def noofsides(self):
        print("I have 5 sides")

class Hexagon(Polygon):

    # overriding abstract method
    def noofsides(self):
        print("I have 6 sides")

R = Triangle()
R.noofsides()

R = Pentagon()
R.noofsides()

K = Hexagon()
K.noofsides()
```

```
runfile('E:/okul/SCO1/Polygon.py', wdir='E:/okul/SCO1')
I have 3 sides
I have 5 sides
I have 6 sides
```

Abstract Class if_x (for mathematical function f(x))

```
from abc import ABC, abstractmethod
from math import *

class if_x(ABC):

    @abstractmethod
    def func(self,x):
        pass;

    def dfunc(self,x,n):
        dx=0.001;
        if n==0: df=self.func(x);
        elif n==1: df=(3.0*self.func(x-4.0*dx)-32.0*self.func(x-3.0*dx)+168.0*self.func(x-2.0*dx)-672.0*self.func(x-
dx)+672.0*self.func(x+dx)-168.0*self.func(x+2.0*dx)+32.0*self.func(x+3.0*dx)-3.0*self.func(x+4.0*dx))/840.0/dx;
        elif n==2: df=(-14350.0*self.func(x)-9.0*self.func(x-4.0*dx)+128.0*self.func(x-3.0*dx)-1008.0*self.func(x-
2.0*dx)+8064.0*self.func(x-dx)+8064.0*self.func(x+dx)-1008.0*self.func(x+2.0*dx)+128.0*self.func(x+3.0*dx)-
9.0*self.func(x+4.0*dx))/5040.0/dx/dx;
        elif n==3: df=(-7.0*self.func(x-4.0*dx)+72.0*self.func(x-3.0*dx)-338.0*self.func(x-2.0*dx)+488.0*self.func(x-dx)-
488.0*self.func(x+dx)+338.0*self.func(x+2.0*dx)-72.0*self.func(x+3.0*dx)+7.0*self.func(x+4.0*dx))/240.0/dx/dx/dx;
```

```
        elif n==4: df=(2730.0*self.func(x)+7.0*self.func(x-4.0*dx)-96.0*self.func(x-3.0*dx)+676.0*self.func(x-2*dx)-1952.0*self.func(x-
dx)-1952.0*self.func(x+dx)+676.0*self.func(x+2.0*dx)-96.0*self.func(x+3.0*dx)+7.0*self.func(x+4.0*dx))/240.0/dx/dx/dx/dx;
        elif n==5: df=(self.func(x-4.0*dx)-9.0*self.func(x-3.0*dx)+26.0*self.func(x-2.0*dx)-29.0*self.func(x-dx)+29.0*self.func(x+dx)-
26.0*self.func(x+2.0*dx)+9.0*self.func(x+3.0*dx)-self.func(x+4.0*dx))/6.0/dx/dx/dx/dx/dx;
        elif n==6: df=(-150.0*self.func(x)-self.func(x-4.0*dx)+12.0*self.func(x-3.0*dx)-52.0*self.func(x-2.0*dx)+116.0*self.func(x-
dx)+116.0*self.func(x+dx)-52.0*self.func(x+2.0*dx)+12.0*self.func(x+3.0*dx)-self.func(x+4.0*dx))/4.0/dx/dx/dx/dx/dx/dx;
        elif n==7: df=(-self.func(x-4.0*dx)+6.0*self.func(x-3.0*dx)-14.0*self.func(x-2.0*dx)+14.0*self.func(x-dx)-
14.0*self.func(x+dx)+14.0*self.func(x+2.0*dx)-6.0*self.func(x+3.0*dx)+self.func(x+4.0*dx))/2.0/dx/dx/dx/dx/dx/dx/dx;
        elif n==8: df=(70.0*self.func(x)+self.func(x-4.0*dx)-8.0*self.func(x-3.0*dx)+28.0*self.func(x-2.0*dx)-56.0*self.func(x-dx)-
56.0*self.func(x+dx)+28.0*self.func(x+2.0*dx)-8.0*self.func(x+3.0*dx)+self.func(x+4.0*dx))/dx/dx/dx/dx/dx/dx/dx/dx;
        else: df=0;
        return df;

    def gauss_legendre_coefficients(self,x1,x2,n):
        #calculates legendre gauss-coefficients as coefficients of the integral
        #for n terms
        eps=3e-15
        m=int((n+1.0)/2.0)
        xm=0.5*(x2+x1)
        xl=0.5*(x2-x1)
        nmax=100
        a=[[0.0 for i in range(n)] for j in range(2)]
        for i in range(1,m+1):
            z=cos(pi*((i-0.25)/(n+0.5)))
            for ii in range(nmax):
                #while abs(z-z1) > eps:
                p1=1.0;
                p2=0.0;
                for j in range(1,n+1):
                    p3=p2
                    p2=p1
                    p1=(float)(((2.0*j-1.0)*z*p2-(j-1.0)*p3)/j)
                pp=(float)(n*(z*p1-p2)/(z*z-1.0))
                z1=z;
                z=z1-p1/pp
                if abs(z-z1) < eps: break
            a[0][i-1]=xm-xl*z
            a[0][n-i]=xm+xl*z
            a[1][i-1]=2.0*xl/((1.0-z*z)*pp*pp)
            a[1][n-i]=a[1][i-1]
        return a

    def integral(self,x1,x2,n):
        #integral func(x)dx
        #integral of a function by using gauss-legendre quadrature
        #between x1 and x2
        a=self.gauss_legendre_coefficients(x1,x2,n)
        z=0
        for i in range(n):
            z=z+a[1][i]*self.func(a[0][i])
        return z
```

```
from math import *
from if_x import *;

class f1(if_x):
  def func(self,x):
      y=x*x-2;
      return y;

x=[1.2,2.3,3.4,4.5,5.6,6.0,5.7,5.1,4.6]
n=len(x)
f = f1()
for i in range(n):
    print("i = ",i," x = ",x[i],"y=f(x) = ",f.func(x[i]))
```

```
runfile('E:/okul/SCO1/if_x_test.py', wdir='E:/okul/SCO1')
Reloaded modules: if_x
i =  0  x =  1.2 y=f(x) =  -0.56
i =  1  x =  2.3 y=f(x) =  3.289999999999999
i =  2  x =  3.4 y=f(x) =  9.559999999999999
i =  3  x =  4.5 y=f(x) =  18.25
i =  4  x =  5.6 y=f(x) =  29.359999999999996
i =  5  x =  6.0 y=f(x) =  34.0
i =  6  x =  5.7 y=f(x) =  30.490000000000002
i =  7  x =  5.1 y=f(x) =  24.009999999999998
```

```
i =  8  x =  4.6 y=f(x) =  19.159999999999997
```

Lambda version of class f1

```
from math import *
from if_x import *;

x=[1.2,2.3,3.4,4.5,5.6,6.0,5.7,5.1,4.6]
n=len(x)
class f1(if_x):func=lambda self,x: x*x
f = f1()
for i in range(n):
    print("i = ",i," x = ",x[i],"y=f(x) = ",f.func(x[i]))
```

```
runfile('E:/okul/SCO1/if_x_test.py', wdir='E:/okul/SCO1')
Reloaded modules: if_x
i =  0  x =  1.2 y=f(x) =  1.44
i =  1  x =  2.3 y=f(x) =  5.289999999999999
i =  2  x =  3.4 y=f(x) =  11.559999999999999
i =  3  x =  4.5 y=f(x) =  20.25
i =  4  x =  5.6 y=f(x) =  31.359999999999996
i =  5  x =  6.0 y=f(x) =  36.0
i =  6  x =  5.7 y=f(x) =  32.49
i =  7  x =  5.1 y=f(x) =  26.009999999999998
i =  8  x =  4.6 y=f(x) =  21.159999999999997
```

The same function can be defined with **lambda variable method(function**) as:

```
x=[1.2,2.3,3.4,4.5,5.6,6.0,5.7,5.1,4.6]
n=len(x)
f = lambda x: x*x-2;
for i in range(n):
    print("i = ",i," x = ",x[i],"y=f(x) = ",f(x[i]))
```

```
runfile('E:/okul/SCO1/if_x_test1.py', wdir='E:/okul/SCO1')
i =  0  x =  1.2 y=f(x) =  -0.56
i =  1  x =  2.3 y=f(x) =  3.289999999999999
i =  2  x =  3.4 y=f(x) =  9.559999999999999
i =  3  x =  4.5 y=f(x) =  18.25
i =  4  x =  5.6 y=f(x) =  29.359999999999996
i =  5  x =  6.0 y=f(x) =  34.0
i =  6  x =  5.7 y=f(x) =  30.490000000000002
i =  7  x =  5.1 y=f(x) =  24.009999999999998
i =  8  x =  4.6 y=f(x) =  19.159999999999997
```

Abstract class f_xj (for multi independent function f(x0,x1,x2..))

```
from abc import ABC, abstractmethod
from math import *

class f_xj(ABC):


  # multifunction multi independent variable
  # vector of dependent variables are returned
  # example f[0]=x[0]+sin(x[1])
  #         f[1]=x[0]*x[1]-x[1]
  # func(x) returns the value of f[0] and f[1]
  # as a two dimensional vector

  @abstractmethod
  def func(self,x):
    pass;

  def funcxy(self,x1,y1):
    n=2;
    x=[0 for i in range(n)];
    x[0]=x1;
    x[1]=y1;
    return self.func(x);

  def dfunc_(self,x,x_ref):
     #derivative of the function with respect to x_ref
     h0=0.015675863;
     m=7;
     n=len(x);
     x1=[0 for i in range(n)];
     x2=[0 for i in range(n)];
```

```
    for i in range(n):
        x1[i]=x[i];
        x2[i]=x[i];
    #derivative of a simple function
    T=[[0 for i2 in range(m)] for j2 in range(m)];
    h=[0 for i in range(m)];
    for i in range(m):
        h[i]=0;
        for j in range(m):
            T[i][j]=0;
    h[0]=h0;
    r=0.5;
    for i in range(m):
        h[i]=h0*pow(r,i);
    for i in range(m):
        x1[x_ref]+=h[i];
        x2[x_ref]-=h[i];
        f1=self.func(x1);
        f2=self.func(x2);
        T[i][0]=( f1 - f2)/(2.0*h[i]);
        x1[x_ref]=x[x_ref];
        x2[x_ref]=x[x_ref];
    for k in range(1,m):
         for i in range(0,m-k):
             T[i][k]=(h[i]*h[i]*T[i+1][k-1] - h[i+k]*h[i+k]*T[i][k-1])/(h[i]*h[i]
             - h[i+k]*h[i+k]);
    xx=T[0][n-1];
    return xx;

def d2func_(self,x,x_ref):
    #second derivative of the function with respect to x_ref
    h0=0.015675863;
    m=7;
    n=len(x);
    x1=[0 for i in range(n)];
    x2=[0 for i in range(n)];
    for i in range(n):
        x1[i]=x[i];
        x2[i]=x[i];
    #derivative of a simple function
    T=[[0 for i2 in range(m)] for j2 in range(m)];
    h=[0 for i in range(m)];
    for i in range(m):
        h[i]=0;
        for j in range(m):
            T[i][j]=0;
    h[0]=h0;
    r=0.5;
    for i in range(m):
        h[i]=h0*pow(r,i);
    for i in range(m):
        x1[x_ref[1]]+=h[i];
        x2[x_ref[1]]-=h[i];
        f1=self.dfunc_(x1,x_ref[0]);
        f2=self.dfunc_(x2,x_ref[0]);
        T[i][0]=( f1 - f2)/(2.0*h[i]);
        x1[x_ref[1]]=x[x_ref[1]];
        x2[x_ref[1]]=x[x_ref[1]];
    for k in range(1,m):
         for i in range(0,m-k):
             T[i][k]=(h[i]*h[i]*T[i+1][k-1] - h[i+k]*h[i+k]*T[i][k-1])/(h[i]*h[i]
             - h[i+k]*h[i+k]);
    xx=T[0][n-1];
    return xx;

def dfunc(self,x):
    n=len(x);
    c=[0 for i in range(n)];
    for i in range(n):
        c[i]=self.dfunc_(x,i);
    return c;

def d2func(self,x):
    n=len(x);
    x_ref=[0 for i2 in range(2)];
    c=[[0 for i2 in range(n)] for j2 in range(n)];
```

```
    for i in range(n):
        for j in range(n):
            x_ref[0]=i;
            x_ref[1]=j;
            c[i][j]=self.d2func_(x,x_ref);
    return c;
```

Separate class version

```
# -*- coding: utf-8 -*-
"""
Created on Fri Aug 24 16:46:48 2018

@author: Mustafa Turhan Çoban
"""
from math import *
from f_xj import *

class f2(f_xj):
  def func(self,x):
      y=sin(x[0])*x[0]+sin(x[1])*x[1];
      return y;

f=f2();
x=[1,1];
df=f.dfunc(x);
print("df",df);
d2f=f.d2func(x);
print("d2f",d2f);
```

```
runfile('E:/okul/SCO1/f_xj.py', wdir='E:/okul/SCO1')

runfile('E:/okul/SCO1/f_xjtest.py', wdir='E:/okul/SCO1')
df [1.3817732900787727, 1.3817732900787727]
d2f [[0.23913362632367666, -3.682833747686372e-13], [-3.682833747686372e-13, 0.23913362632367666]]
```

Lambda version:

```
from math import *
from f_xj import *

class f1(f_xj):func=lambda self,x: sin(x[0])*x[0]+sin(x[1])*x[1]
f=f1()
x=[1,1]
df=f.dfunc(x)
print("df",df)
d2f=f.d2func(x)
```

```
runfile('E:/okul/SCO1/f_xjtest1.py', wdir='E:/okul/SCO1')
Reloaded modules: f_xj
df [1.3817732900787727, 1.3817732900787727]
d2f [[0.23913362632367666, -3.682833747686372e-13], [-3.682833747686372e-13, 0.23913362632367666]]
```

Abstract class fi_xi (for multi independent function-vector function)

f_1(x0,x1,x2..)

f_2(x0,x1,x2..)

…

```
from abc import ABC, abstractmethod
from math import *

class fi_xi(ABC):

  # multifunction multi independent variable
  # vector of dependent variables are returned
  # example f[0]=x[0]+sin(x[1])
  #         f[1]=x[0]*x[1]-x[1]
  # func(x) returns the value of f[0] and f[1]
  # as a two dimensional vector

  @abstractmethod
  def func(self,x):
     pass;

  def xp(self,x,j,m):
      k=len(x);
      h=1.0e-6
      xx=[0 for i in range(k)];
```

```
    for i in range(k):
        if i==m:
            xx[i]=x[i]+j*h;
        else: xx[i]=x[i];
    return xx;

 def dfunc(self,x,n):
    h=1.0e-6;
    k=len(x);
    df=[[0 for i in range(k)] for j in range(k)];
    for i in range(k):
        for j in range(k):
            df[i][j]=self.dfun(x,n,i,j);
    return df;

 def dfun(self,x,n,j,m):
    k=len(x);
    ff=[[0 for i2 in range(k)] for j2 in range(9)];
    xp1=[[0 for i2 in range(k)] for j2 in range(9)];
    h=1.0e-6;
    hh=1/h;
    x1=x[m];
    for jj in range(-4,5):
       j1=jj+4;
       xp1[j1]=self.xp(x,jj,m);
       ff[j1]=self.func(xp1[j1]);
    if n==0: df=ff[4][j];
    elif n==1:
        df=(3.0*ff[0][j]-32.0*ff[1][j]+168.0*ff[2][j]-672.0*ff[3][j]+672.0*ff[5][j]-168.0*ff[6][j]+32.0*ff[7][j]-3.0*ff[8][j])/840.0*hh;
    elif n==2:
        df=(-14350.0*ff[4][j]-9.0*ff[0][j]+128*ff[1][j]-1008*ff[2][j]+8064*ff[3][j]+8064.0*ff[5][j]-1008.0*ff[6][j]+128.0*ff[7][j]-
9.0*ff[8][j])/5040.0*hh*hh;
    elif n==3:
        df=(-7.0*ff[0][j]+72.0*ff[1][j]-338.0*ff[2][j]+488.0*ff[3][j]-488.0*ff[5][j]+338.0*ff[6][j]-
72.0*ff[7][j]+7.0*ff[8][j])/240.0*hh*hh*hh;
    elif n==4:
        df=(2730.0*ff[4][j]+7.0*ff[0][j]-96.0*ff[1][j]+676.0*ff[2][j]-1952.0*ff[3][j]-1952.0*ff[5][j]+676.0*ff[6][j]-
96.0*ff[7][j]+7.0*ff[8][j])/240.0*hh*hh*hh*hh;
    elif n==5:
        df=(ff[0][j]-9.0*ff[1][j]+26.0*ff[2][j]-29.0*ff[3][j]+29.0*ff[5][j]-26.0*ff[6][j]+9.0*ff[7][j]-ff[8][j])/6.0*hh*hh*hh*hh*hh;
    elif n==6:
        df=(-150.0*ff[4][j]-ff[0][j]+12.0*ff[1][j]-52.0*ff[2][j]+116.0*ff[3][j]+116.0*ff[5][j]-52.0*ff[6][j]+12.0*ff[7][j]-
ff[8][j])/4.0*hh*hh*hh*hh*hh*hh;
    elif n==7:
        df=(-ff[0][j]+6.0*ff[1][j]-14.0*ff[2][j]+14.0*ff[3][j]-14.0*ff[5][j]+14.0*ff[6][j]-6.0*ff[7][j]+ff[8][j])/2.0*hh*hh*hh*hh*hh*hh*hh;
    elif n==8:
        df=(70.0*ff[4][j]+ff[0][j]-8.0*ff[1][j]+28.0*ff[2][j]-56.0*ff[3][j]-56.0*ff[5][j]+28.0*ff[6][j]-
8.0*ff[7][j]+ff[8][j])*hh*hh*hh*hh*hh*hh*hh*hh;
    else: df=0;
    return df
```

```
from math import *
from fi_xi import *

class f2(fi_xi):
  def func(self,x):
    y=[0 for i2 in range(2)]
    y[0]=sin(x[0])*x[0]+sin(x[1])*x[1]
    y[1]=x[0]*x[0]-x[1]
    return y;

f=f2();
x=[1.0,1.0]
ff=f.func(x)
print("f=\n",ff)
df=f.dfunc(x,1)
print("df=\n",df)
```

```
runfile('E:/okul/SCO1/fi_xjtest.py', wdir='E:/okul/SCO1')
Reloaded modules: fi_xi
f=
 [1.682941969615793, 0.0]
df=
 [[1.3817732906328843, 1.3817732906328843], [2.000000000016142, -0.999999999954873]]
```

Abstract class fij_xj (for multi independent function-matrix function)

f_{11}(x0,x1,x2..) f_{12}(x0,x1,x2..) …
f_{21}(x0,x1,x2..) f_{22}(x0,x1,x2..) …
…..
f_{n1}(x0,x1,x2..) f_{n2}(x0,x1,x2..) …

```
from abc import ABC, abstractmethod
from math import *

class fij_xi(ABC):

  # multifunction multi independent variable
  # for n independent variable n*n matrix of
  # dependent variables are returned
  # example f[0][0]=x[0]+Math.sin(x[1])   f[0][1]=x[0]-x[1]
  #         f[1][0]=x[0]*x[0]-x[1]        f[1][1]=Math.exp(x[0]+x[1]*x[1]
  # func(x) returns the value of
  # multifunction multi independent variable
  # for n independent variable n*n matrix of
  # dependent variables are returned
  # example f[0][0]=x[0]+Math.sin(x[1])   f[0][1]=x[0]-x[1]
  #         f[1][0]=x[0]*x[0]-x[1]        f[1][1]=Math.exp(x[0]+x[1]*x[1]
  # func(x) returns the value of
  #         f[0][0], f[0][1]
  #         f[1][0], f[1][1]
  # as a two dimensional matrix

  @abstractmethod
  def func(self,x):
     pass;
```

```
from math import *
from fij_xi import *

class f2(fij_xi):
  def func(self,x):
      y=[[0 for j in range(2)] for i in range(2)]
      y[0][0]=2.0*x[0]+x[1];
      y[0][1]=x[0];
      y[1][0]=3.0*x[1]*x[1];
      y[1][1]=1.0+6.0*x[0]*x[1];
      return y;

f=f2();
x=[1.0,1.0]
ff=f.func(x)
print("f=\n",ff)
```

```
runfile('E:/okul/SCO1/fij_xitest.py', wdir='E:/okul/SCO1')
Reloaded modules: fij_xi
f=
 [[3.0, 1.0], [3.0, 7.0]]
```

Iterable in Python:
An iteratable is a Python object that can be used as a sequence. You can go to the next item of the sequence using the next() method.
You can loop over an iterable, but you cannot access individual elements directly.
It's a container object: it can only return one of its element at the time.

```
d = { "one": 1, "two": 2, "three": 3, "four": 4, "five": 5 }
iterable = d.keys()
for item in iterable:
   print(item)
```

```
runfile('E:/okul/SCO1/iterable1.py', wdir='E:/okul/SCO1')
one
two
three
four
five
```

1.7 MATH LIBRARY

Several mathematical functions(methods) are defined in math library of python. Detail information can be obtained from https://docs.python.org/3/library/math.html internet site. A list of methods and constants are given below as a table

ceil(x)	Return the ceiling of x, the smallest integer greater than or equal to x. If x is not a float,
copysign(x, y)	Return a float with the magnitude (absolute value) of x but the sign of y.
comb(n, k)	Return the number of ways to choose k items from n items without repetition
fabs(x)	Return the absolute value of x.
factorial(x)	Return x factorial as an integer.
floor(x)	Return the floor of x, the largest integer less than or equal to x.
fmod(x, y)	Return fmod(x, y)
frexp(x)	Return the mantissa and exponent of x as the pair (m, e). m is a float and e is an integer
fsum(iterable)	Return an accurate floating point sum of values in the iterable.
gcd(a, b)	Return the greatest common divisor of the integers a and b. If either a or b is nonzero
modf(x)	Return the fractional and integer parts of x.
prod(iterable, *, start=1)	Calculate the product of all the elements in the input iterable. The default start value for the product is 1.
remainder(x, y)	Return the IEEE 754-style remainder of x with respect to y.
trunc(x)	Return the Real value x truncated to an integer
exp(x)	Return e raised to the power x, where e = 2.718281… is the base of natural logarithms.
expm1(x)	Return e raised to the power x, minus 1.
log10(x)	Return the base-10 logarithm of x.
log(x[, base])	With one argument, return the natural logarithm of x (to base e).
pow(x, y)	Return x raised to the power y.
sqrt(x)	Return the square root of x.
acos(x)	Return the arc cosine of x, in radians.
asin(x)	Return the arc sine of x, in radians.
atan(x)	Return the arc tangent of x, in radians.
atan2(y, x)	Return atan(y / x), in radians. The result is between -pi and pi. T
cos(x)	Return the cosine of x radians.
dist(p, q)	Return the Euclidean distance between two points p and q,
hypot(*coordinates)	Return the Euclidean norm, sqrt(sum(x**2 for x in coordinates)).
sin(x)	Return the sine of x radians.
tan(x)¶	Return the tangent of x radians.
degrees(x)	Convert angle x from radians to degrees.
radians(x)	Convert angle x from degrees to radians.
acosh(x)	Return the inverse hyperbolic cosine of x.
asinh(x)	Return the inverse hyperbolic sine of x
atanh(x)	Return the inverse hyperbolic tangent of x.
cosh(x)	Return the hyperbolic cosine of x.
sinh(x)	Return the hyperbolic sine of x
tanh(x)	Return the hyperbolic tangent of x.
erf(x)	Return the error function at x.
phi(x)	Cumulative distribution function for standard normal distribution
erfc(x)	Return the complementary error function at x.
lgamma(x)	Return the natural logarithm of the absolute value of the Gamma function at x.
CONSTANTS	
pi	The mathematical constant π = 3.141592…, to available precision.
e	The mathematical constant e = 2.718281…, to available precision.
tau	The mathematical constant τ = 6.283185…, to available precision. Tau is a circle constant equal to 2π,
inf	A floating-point positive infinity.
nan	A floating-point "not a number" (NaN) value.

```
x = [0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1]
y=fsum(x)
print("y=",y)
```

```
runfile('E:/okul/SCO1/iterable1.py', wdir='E:/okul/SCO1')
y= 1.0
```

Note cmath library is available for similar functions for complex variables. Another important function library in pyython is scripy.special A very rich collection of mathematical functions are available in this library (**https://docs.scipy.org/doc/scipy/reference/special.html**). For full list please refer to above internet site. Some of the functions in the library is listed in here:

Elliptic functions and integrals	
ellipj(u, m)	Jacobian elliptic functions

ellipk(m)	Complete elliptic integral of the first kind.	
ellipkm1(p)	Complete elliptic integral of the first kind around $m = 1$	
ellipkinc(phi, m)	Incomplete elliptic integral of the first kind	
ellipe(m)	Complete elliptic integral of the second kind	
ellipeinc(phi, m)	Incomplete elliptic integral of the second kind	
Bessel functions		
jv(v, z)	Bessel function of the first kind of real order and complex	
jve(v, z)	Exponentially scaled Bessel function of order *v*.	
yn(n, x)	Bessel function of the second kind of integer order and real	
yv(v, z)	Bessel function of the second kind of real order and complex	
yve(v, z)	Exponentially scaled Bessel function of the second kind of real	
kn(n, x)	Modified Bessel function of the second kind of integer order *n*	
kv(v, z)	Modified Bessel function of the second kind of real order *v*	
kve(v, z)	Exponentially scaled modified Bessel function of the second kind.	
iv(v, z)	Modified Bessel function of the first kind of real order.	
ive(v, z)	Exponentially scaled modified Bessel function of the first kind	
hankel1(v, z)	Hankel function of the first kind	
hankel1e(v, z)	Exponentially scaled Hankel function of the first kind	
hankel2(v, z)	Hankel function of the second kind	
hankel2e(v, z)	Exponentially scaled Hankel function of the second kind	
wright_bessel(a, b, x)	Wright's generalized Bessel function.	
lmbda(v, x)	Jahnke-Emden Lambda function, Lambdav(x).	
Zeros of Bessel functions		
jnjnp_zeros(nt)	Compute zeros of integer-order Bessel functions Jn and Jn'.	
jnyn_zeros(n, nt)	Compute nt zeros of Bessel functions Jn(x), Jn'(x), Yn(x), and	
jn_zeros(n, nt)	Compute zeros of integer-order Bessel functions Jn.	
jnp_zeros(n, nt)	Compute zeros of integer-order Bessel function derivatives Jn'.	
yn_zeros(n, nt)	Compute zeros of integer-order Bessel function Yn(x).	
ynp_zeros(n, nt)	Compute zeros of integer-order Bessel function derivatives	
y0_zeros(nt[, complex])	Compute nt zeros of Bessel function Y0(z), and derivative at each	
y1_zeros(nt[, complex])	Compute nt zeros of Bessel function Y1(z), and derivative at each	
y1p_zeros(nt[, complex])	Compute nt zeros of Bessel derivative Y1'(z), and value at each	
Faster versions of common Bessel functions		
j0(x)	Bessel function of the first kind of order 0.	
j1(x)	Bessel function of the first kind of order 1.	
y0(x)	Bessel function of the second kind of order 0.	
y1(x)	Bessel function of the second kind of order 1.	
i0(x)	Modified Bessel function of order 0.	
i0e(x)	Exponentially scaled modified Bessel function of order 0.	
i1(x)	Modified Bessel function of order 1.	
i1e(x)	Exponentially scaled modified Bessel function of order 1.	
k0(x)	Modified Bessel function of the second kind of order 0, K0.	
k0e(x)	Exponentially scaled modified Bessel function K of order 0	
k1(x)	Modified Bessel function of the second kind of order 1, K1(x).	
k1e(x)	Exponentially scaled modified Bessel function K of order 1	
Integrals of Bessel functions		
itj0y0(x[, out])	Integrals of Bessel functions of the first kind of order 0.	
it2j0y0(x[, out])	Integrals related to Bessel functions of the first kind of order 0.	
iti0k0(x[, out])	Integrals of modified Bessel functions of order 0.	
it2i0k0(x[, out])	Integrals related to modified Bessel functions of order 0.	
besselpoly(a, lmb, nu[, out])	Weighted integral of the Bessel function of the first kind.	
Derivatives of Bessel functions		
jvp(v, z[, n])	Compute derivatives of Bessel functions of the first kind.	
yvp(v, z[, n])	Compute derivatives of Bessel functions of the second kind.	
kvp(v, z[, n])	Compute nth derivative of real-order modified Bessel function	
ivp(v, z[, n])	Compute derivatives of modified Bessel functions of the first	
h1vp(v, z[, n])	Compute nth derivative of Hankel function H1v(z) with respect	
h2vp(v, z[, n])	Compute nth derivative of Hankel function H2v(z) with respect	
Spherical Bessel functions		
spherical_jn(n, z[, derivative])	Spherical Bessel function of the first kind or its derivative.	

spherical_yn(n, z[, derivative])	Spherical Bessel function of the second kind or its derivative.	
spherical_in(n, z[, derivative])	Modified spherical Bessel function of the first kind or its	
spherical_kn(n, z[, derivative])	Modified spherical Bessel function of the second kind or its	
Gamma and related functions		
gamma(z)	gamma function.	
gammaln(x[, out])	Logarithm of the absolute value of the gamma function.	
loggamma(z[, out])	Principal branch of the logarithm of the gamma function.	
gammasgn(x)	Sign of the gamma function.	
gammainc(a, x)	Regularized lower incomplete gamma function.	
gammaincinv(a, y)	Inverse to the lower incomplete gamma function with respect to *x*.	
gammaincc(a, x)	Regularized upper incomplete gamma function.	
gammainccinv(a, y)	Inverse of the upper incomplete gamma function with respect to *x*	
beta(a, b[, out])	Beta function.	
betaln(a, b)	Natural logarithm of absolute value of beta function.	
betainc(a, b, x[, out])	Incomplete beta function.	
betaincinv(a, b, y[, out])	Inverse of the incomplete beta function.	
psi(z[, out])	The digamma function.	
rgamma(z[, out])	Reciprocal of the gamma function.	
polygamma(n, x)	Polygamma functions.	
multigammaln(a, d)	Returns the log of multivariate gamma, also sometimes called the	
digamma(z[, out])	The digamma function.	
poch(z, m)	Pochhammer symbol.	
Error function and Fresnel integrals		
erf(z)	Returns the error function of complex argument.	
erfc(x[, out])	Complementary error function, 1 - erf(x).	
erfcx(x[, out])	Scaled complementary error function, exp(x**2) * erfc(x).	
erfi(z[, out])	Imaginary error function, -i erf(i z).	
erfinv(x, /[, out, where, casting, order, …])	Inverse of the error function.	
erfcinv(x, /[, out, where, casting, order, …])	Inverse of the complementary error function.	
wofz(z)	Faddeeva function	
dawsn(x)	Dawson's integral.	
fresnel(z[, out])	Fresnel integrals.	
fresnel_zeros(nt)	Compute nt complex zeros of sine and cosine Fresnel integrals	
modfresnelp(x)	Modified Fresnel positive integrals	
modfresnelm(x)	Modified Fresnel negative integrals	
voigt_profile(x, sigma, gamma[, out])	Voigt profile.	
erf_zeros(nt)	Compute the first nt zero in the first quadrant, ordered by absolute	
fresnelc_zeros(nt)	Compute nt complex zeros of cosine Fresnel integral C(z).	
fresnels_zeros(nt)	Compute nt complex zeros of sine Fresnel integral S(z).	
Legendre functions		
lpmv(m, v, x)	Associated Legendre function of integer order and real degree.	
sph_harm(m, n, theta, phi)	Compute spherical harmonics.	
clpmn(m, n, z[, type])	Associated Legendre function of the first kind for complex	
lpn(n, z)	Legendre function of the first kind.	
lqn(n, z)	Legendre function of the second kind.	
lpmn(m, n, z)	Sequence of associated Legendre functions of the first kind.	
lqmn(m, n, z)	Sequence of associated Legendre functions of the second kind.	
Ellipsoidal harmonics		
ellip_harm(h2, k2, n, p, s[, signm, signn])	Ellipsoidal harmonic functions E^p_n(l)	
ellip_harm_2(h2, k2, n, p, s)	Ellipsoidal harmonic functions F^p_n(l)	
ellip_normal(h2, k2, n, p)	Ellipsoidal harmonic normalization constants gamma^p_n	
Orthogonal polynomials		
The following functions evaluate values of orthogonal		
assoc_laguerre(x, n[, k])	Compute the generalized (associated) Laguerre polynomial of	
eval_legendre(n, x[, out])	Evaluate Legendre polynomial at a point.	
eval_chebyt(n, x[, out])	Evaluate Chebyshev polynomial of the first kind at a point.	
eval_chebyu(n, x[, out])	Evaluate Chebyshev polynomial of the second kind at a point.	
eval_chebyc(n, x[, out])	Evaluate Chebyshev polynomial of the first kind on [-2, 2] at a	
eval_chebys(n, x[, out])	Evaluate Chebyshev polynomial of the second kind on [-2, 2] at a	
eval_jacobi(n, alpha, beta, x[, out])	Evaluate Jacobi polynomial at a point.	
eval_laguerre(n, x[, out])	Evaluate Laguerre polynomial at a point.	

eval_genlaguerre(n, alpha, x[, out])	Evaluate generalized Laguerre polynomial at a point.	
eval_hermite(n, x[, out])	Evaluate physicist's Hermite polynomial at a point.	
eval_hermitenorm(n, x[, out])	Evaluate probabilist's (normalized) Hermite polynomial at a	
eval_gegenbauer(n, alpha, x[, out])	Evaluate Gegenbauer polynomial at a point.	
eval_sh_legendre(n, x[, out])	Evaluate shifted Legendre polynomial at a point.	
eval_sh_chebyt(n, x[, out])	Evaluate shifted Chebyshev polynomial of the first kind at a point.	
eval_sh_chebyu(n, x[, out])	Evaluate shifted Chebyshev polynomial of the second kind at a	
eval_sh_jacobi(n, p, q, x[, out])	Evaluate shifted Jacobi polynomial at a point.	
The following functions compute roots and quadrature		
roots_legendre(n[, mu])	Gauss-Legendre quadrature.	
roots_chebyt(n[, mu])	Gauss-Chebyshev (first kind) quadrature.	
roots_chebyu(n[, mu])	Gauss-Chebyshev (second kind) quadrature.	
roots_chebyc(n[, mu])	Gauss-Chebyshev (first kind) quadrature.	
roots_chebys(n[, mu])	Gauss-Chebyshev (second kind) quadrature.	
roots_jacobi(n, alpha, beta[, mu])	Gauss-Jacobi quadrature.	
roots_laguerre(n[, mu])	Gauss-Laguerre quadrature.	
roots_genlaguerre(n, alpha[, mu])	Gauss-generalized Laguerre quadrature.	
roots_hermite(n[, mu])	Gauss-Hermite (physicist's) quadrature.	
roots_hermitenorm(n[, mu])	Gauss-Hermite (statistician's) quadrature.	
roots_gegenbauer(n, alpha[, mu])	Gauss-Gegenbauer quadrature.	
roots_sh_legendre(n[, mu])	Gauss-Legendre (shifted) quadrature.	
roots_sh_chebyt(n[, mu])	Gauss-Chebyshev (first kind, shifted) quadrature.	
roots_sh_chebyu(n[, mu])	Gauss-Chebyshev (second kind, shifted) quadrature.	
roots_sh_jacobi(n, p1, q1[, mu])	Gauss-Jacobi (shifted) quadrature.	
The functions below, in turn, return the polynomial		
legendre(n[, monic])	Legendre polynomial.	
chebyt(n[, monic])	Chebyshev polynomial of the first kind.	
chebyu(n[, monic])	Chebyshev polynomial of the second kind.	
chebyc(n[, monic])	Chebyshev polynomial of the first kind on [−2,2].	
chebys(n[, monic])	Chebyshev polynomial of the second kind on [−2,2].	
jacobi(n, alpha, beta[, monic])	Jacobi polynomial.	
laguerre(n[, monic])	Laguerre polynomial.	
genlaguerre(n, alpha[, monic])	Generalized (associated) Laguerre polynomial.	
hermite(n[, monic])	Physicist's Hermite polynomial.	
hermitenorm(n[, monic])	Normalized (probabilist's) Hermite polynomial.	
gegenbauer(n, alpha[, monic])	Gegenbauer (ultraspherical) polynomial.	
sh_legendre(n[, monic])	Shifted Legendre polynomial.	
sh_chebyt(n[, monic])	Shifted Chebyshev polynomial of the first kind.	
sh_chebyu(n[, monic])	Shifted Chebyshev polynomial of the second kind.	
sh_jacobi(n, p, q[, monic])	Shifted Jacobi polynomial.	
Hypergeometric functions		
hyp2f1(a, b, c, z)	Gauss hypergeometric function 2F1(a, b; c; z)	
hyp1f1(a, b, x[, out])	Confluent hypergeometric function 1F1.	
hyperu(a, b, x[, out])	Confluent hypergeometric function U	
hyp0f1(v, z[, out])	Confluent hypergeometric limit function 0F1.	
Parabolic cylinder functions		
pbdv(v, x)	Parabolic cylinder function D	
pbvv(v, x)	Parabolic cylinder function V	
pbwa(a, x)	Parabolic cylinder function W.	
These are not universal functions:		
pbdv_seq(v, x)	Parabolic cylinder functions Dv(x) and derivatives.	
pbvv_seq(v, x)	Parabolic cylinder functions Vv(x) and derivatives.	
pbdn_seq(n, z)	Parabolic cylinder functions Dn(z) and derivatives.	
Mathieu and related functions		
mathieu_a(m, q)	Characteristic value of even Mathieu functions	
mathieu_b(m, q)	Characteristic value of odd Mathieu functions	
These are not universal functions:		
mathieu_even_coef(m, q)	Fourier coefficients for even Mathieu and modified Mathieu	
mathieu_odd_coef(m, q)	Fourier coefficients for even Mathieu and modified Mathieu	

The following return both function and first derivative:		
mathieu_cem(m, q, x)	Even Mathieu function and its derivative	
mathieu_sem(m, q, x)	Odd Mathieu function and its derivative	
mathieu_modcem1(m, q, x)	Even modified Mathieu function of the first kind and its derivative	
mathieu_modcem2(m, q, x)	Even modified Mathieu function of the second kind and its	
mathieu_modsem1(m, q, x)	Odd modified Mathieu function of the first kind and its derivative	
mathieu_modsem2(m, q, x)	Odd modified Mathieu function of the second kind and its	
Spheroidal wave functions		
pro_ang1(m, n, c, x)	Prolate spheroidal angular function of the first kind and its	
pro_rad1(m, n, c, x)	Prolate spheroidal radial function of the first kind and its	
pro_rad2(m, n, c, x)	Prolate spheroidal radial function of the second kind and its	
obl_ang1(m, n, c, x)	Oblate spheroidal angular function of the first kind and its	
obl_rad1(m, n, c, x)	Oblate spheroidal radial function of the first kind and its	
obl_rad2(m, n, c, x)	Oblate spheroidal radial function of the second kind and its	
pro_cv(m, n, c)	Characteristic value of prolate spheroidal function	
obl_cv(m, n, c)	Characteristic value of oblate spheroidal function	
pro_cv_seq(m, n, c)	Characteristic values for prolate spheroidal wave functions.	
obl_cv_seq(m, n, c)	Characteristic values for oblate spheroidal wave functions.	
The following functions require pre-computed		
pro_ang1_cv(m, n, c, cv, x)	Prolate spheroidal angular function pro_ang1 for precomputed	
pro_rad1_cv(m, n, c, cv, x)	Prolate spheroidal radial function pro_rad1 for precomputed	
pro_rad2_cv(m, n, c, cv, x)	Prolate spheroidal radial function pro_rad2 for precomputed	
obl_ang1_cv(m, n, c, cv, x)	Oblate spheroidal angular function obl_ang1 for precomputed	
obl_rad1_cv(m, n, c, cv, x)	Oblate spheroidal radial function obl_rad1 for precomputed	
obl_rad2_cv(m, n, c, cv, x)	Oblate spheroidal radial function obl_rad2 for precomputed	

```
from scipy.special import *
from math import *
from matplotlib import pyplot as plt
import numpy as np


n=10
dx=1.0/10.0
y=[0 for z in range(n)];
x=[0 for z in range(n)];
for i in range(n):
    x[i]=0.00001+dx*i
    y[i]=erf(x[i])
    print(x[i]," ",y[i])
```

```
runfile('D:/okul/SCO1/erf2.py', wdir='D:/okul/SCO1')
1e-05   1.1283791670579e-05
0.10001   0.11247408752318092
0.20001000000000002   0.2227134305366665
0.30001000000000005   0.3286370720372856
0.40001000000000003   0.4284019704211951
0.50001   0.5205086655948966
0.60001   0.6038639632350085
0.70001   0.6778081065376335
0.80001   0.7421069145179242
0.90001   0.7969132320633975
```

1.8 PLOTTING IN PYTHON

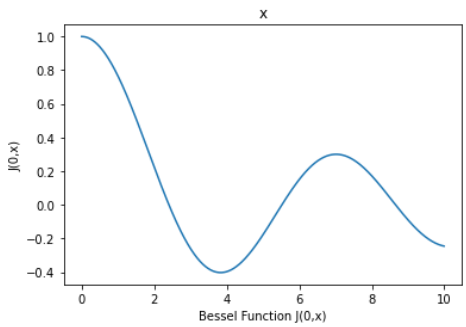
Python has the ability to create graphs by using the matplotlib library. It has numerous packages and functions which generate a wide variety of graphs and plots. It is also very simple to use. It along with numpy and other python built-in functions achieves the goal. In this article we will see some of the different kinds of graphs it can generate.

Matplotlib library documentation:

https://matplotlib.org/3.3.2/contents.html#

```
from math import *
from matplotlib import pyplot as plt
import numpy as np
```

```
def J(v,x):
    JJ=1e-6
    a=1.0
    fact=1
    b=0.0
    if x <=20:
        x5=1
        lna=0.0;
        x1=v*log(0.5*x)
        lna1=log(0.25*x*x)
        for k in range(0,100):
            x2=lna
            x3=lgamma(k+1)
            x4=lgamma(v+k+1)
            b=x1+x2-x3-x4
            b=x5*exp(b)
            lna=lna+lna1
            x5=-1*x5
            JJ=JJ+b
    elif (v==0 and x==0):
        JJ=1
    else:
        JJ=sqrt(2.0/PI/x)*cos(x-0.5*v*PI-0.25*PI)
    return JJ


n=1000
dx=1.0/100.0
y=[0 for z in range(n)];
x=[0 for z in range(n)];
for i in range(n):
    x[i]=0.00001+dx*i
    y[i]=J(0.0,x[i])
    print(x[i]," ",y[i])

plt.plot(x,y)
plt.xlabel("Bessel Function J(0,x)")
plt.ylabel("J(0,x)")
plt.title('x')
plt.show()
```



Bessel function can be obtained from scripy.special function library

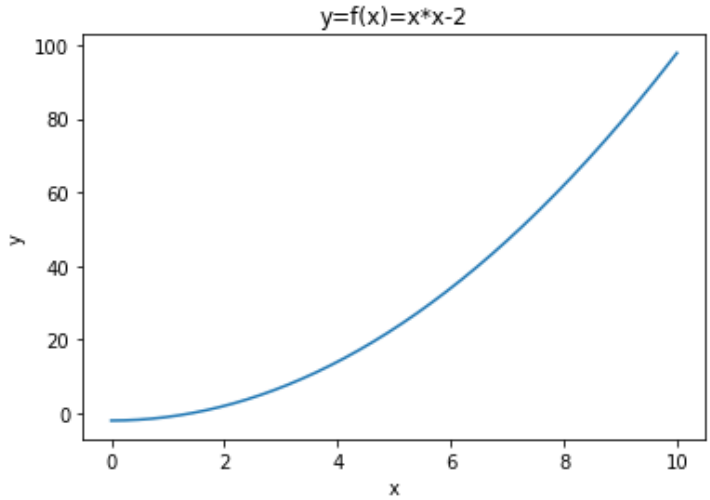
```
from scipy.special import *
from math import *
from matplotlib import pyplot as plt
import numpy as np

n=1000
dx=1.0/100.0
y=[0 for z in range(n)];
x=[0 for z in range(n)];
for i in range(n):
    x[i]=0.00001+dx*i
    y[i]=jv(0.0,x[i])
    print(x[i]," ",y[i])

plt.plot(x,y)
plt.xlabel("Bessel Function J(0,x)")
plt.ylabel("J(0,x)")
plt.title('x')
```

```
plt.show()
```

Bessel function is also available in Mathd library giving as a part of this text

```
from Mathd import *
from matplotlib import pyplot as plt
import numpy as np

n=1000
dx=1.0/100.0
y=[0 for z in range(n)];
x=[0 for z in range(n)];
for i in range(n):
    x[i]=0.00001+dx*i
    y[i]=Mathd.J(0.0,x[i])
    print(x[i]," ",y[i])

plt.plot(x,y)
plt.xlabel("Bessel Function J(0,x)")
plt.ylabel("J(0,x)")
plt.title('x')
plt.show()
```

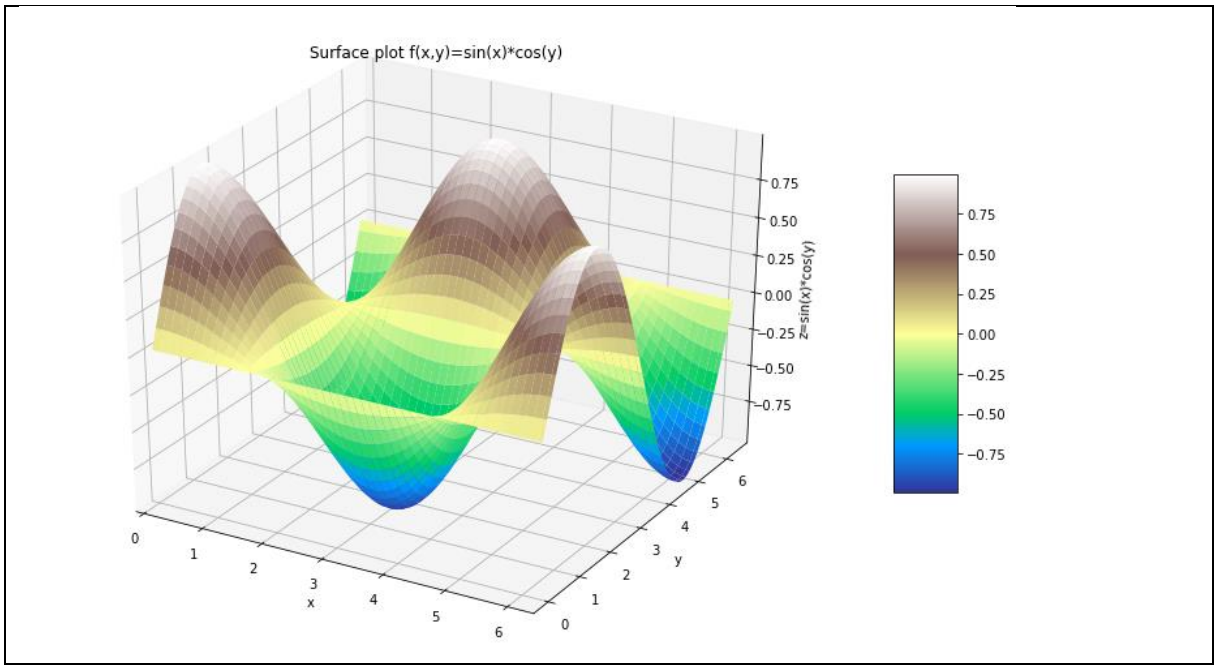
```
from math import *
from matplotlib import pyplot as plt
import numpy as np

f = lambda x: x*x-2;
n=1000
dx=1.0/100.0
y=[0 for z in range(n)];
x=[0 for z in range(n)];
for i in range(n):
    x[i]=0.00001+dx*i
    y[i]=f(x[i])
    print(x[i]," ",y[i])

plt.plot(x,y)
plt.xlabel("x")
plt.ylabel("y")
plt.title('y=f(x)=x*x-2')
plt.show()
```



3D Plot example is given below. The function is f(x)=sin(x)*cos(y)

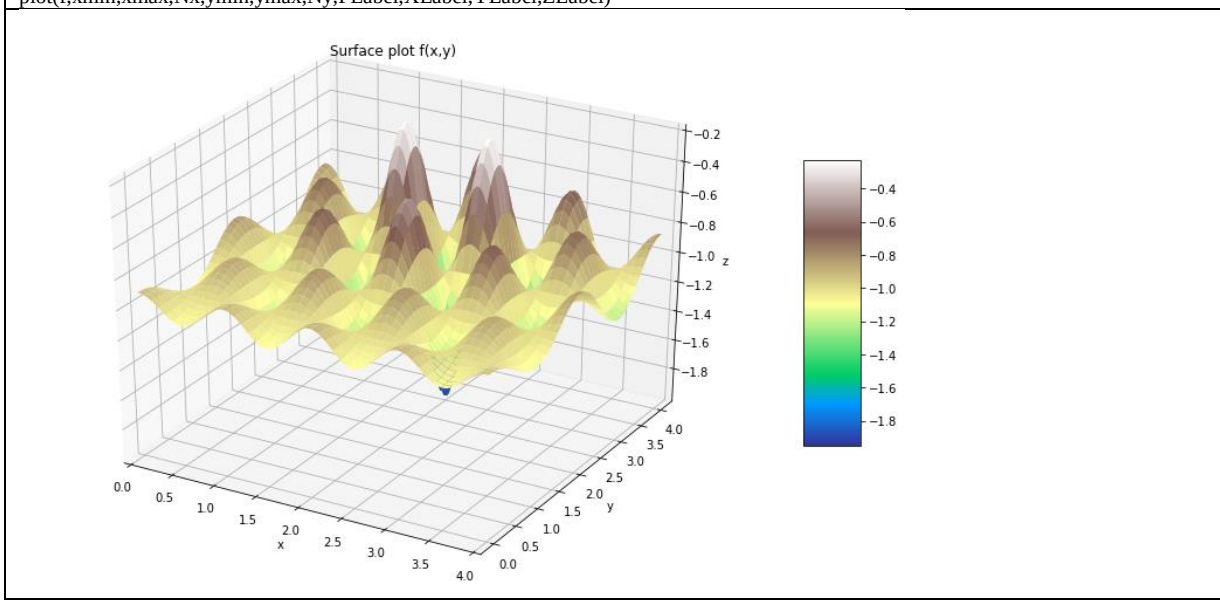
```
# Import libraries
from mpl_toolkits import mplot3d
import numpy as np
import matplotlib.pyplot as plt
from f_xj import *;

#Function to plot
```

```
class f2(f_xj):
  def func(self,x):
      y=sin(x[0])*cos(x[1])
      return y;

# Creating dataset
def plot(f,xmin,xmax,Nx,ymin,ymax,Ny,PLAbel,XLAbel,YLable,ZLabel):
    #f=f2()
    x=[0.0 for j in range(Nx)]
    y=[0.0 for j in range(Ny)]
    z=[[0.0 for j in range(Ny)] for i in range(Nx)]
    dx=(xmax-xmin)/(Nx-1)
    dy=(ymax-ymin)/(Ny-1)
    for i in range(Nx):
        for j in range(Ny):
            x[i]=xmin+dx*i
            y[j] = ymin+dy*j
            z[i][j]=f.funcxy(x[i],y[j])
    xi=np.array(x)
    yi=np.array(y)
    xi, yi = np.meshgrid(xi, yi)
    zi=np.array(z)
    #print(x[i][j]," ",y[i][j]," ",z[i][j])
    fig = plt.figure(figsize =(14, 9))
    ax = plt.axes(projection ='3d')
    # Creating color map
    my_cmap = plt.get_cmap('terrain')
    # Creating plot
    surf = ax.plot_surface(xi, yi, zi,
                    cmap = my_cmap,
                    edgecolor ='none')
    fig.colorbar(surf, ax = ax,
            shrink = 0.5,
            aspect = 5)

    ax.set_title(PLabel)
    ax.set_xlabel(XLabel)
    ax.set_xlim(xmin, xmax)
    ax.set_ylabel(YLabel)
    ax.set_xlim(ymin, ymax)
    ax.set_zlabel(ZLabel)
    ax.set_zlim(np.min(zi), np.max(zi))
    plt.show()

f=f2()
xmin=0.0
xmax=6.283185307
Nx=100
ymin=0.0
ymax=6.283185307
Ny=100
PLabel="Surface plot f(x,y)=sin(x)*cos(y)"
XLabel="x"
YLabel="y"
ZLabel="z=sin(x)*cos(y)"
plot(f,xmin,xmax,Nx,ymin,ymax,Ny,PLabel,XLabel,YLabel,ZLabel)
```



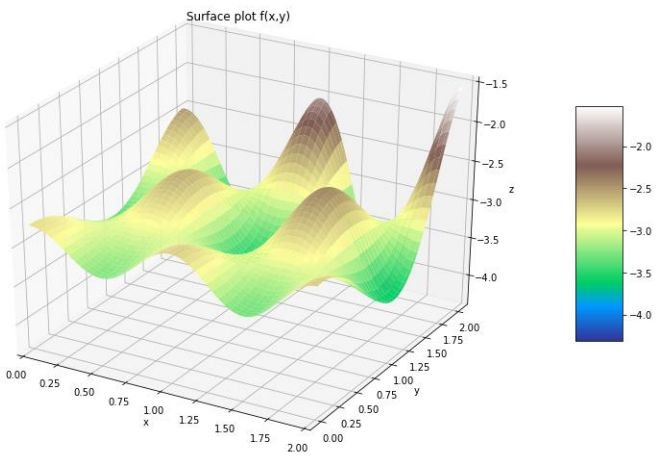
$$z(x,y) = (1 + cos(\pi(x-3) * \frac{cos(2\pi(y-2)}{1+(x-3)^2+(y-2)^2}$$

```
#Function to plot
class f2(f_xj):
  def func(self,x):
      z=-(1.0+cos(pi*(x[0]-3.0))*cos(2.0*pi*(x[1]-2.0))/(1+(x[0]-3.0)*(x[0]-3.0)+(x[1]-2.0)*(x[1]-2.0)))
      return z;

f=f2()
xmin=0.0
xmax=4
Nx=100
ymin=0.0
ymax=4
Ny=100
PLabel="Surface plot f(x,y)"
XLabel="x"
YLabel="y"
ZLabel="z"
plot(f,xmin,xmax,Nx,ymin,ymax,Ny,PLabel,XLabel,YLabel,ZLabel)
```



```
from math import *
from f_xj import *
import  plot3D1 as P3D

class f1(f_xj):func=lambda self,x:-3.0*(1.0+cos(pi*(x[0]-3.0))*cos(2.0*pi*(x[1]-
2.0))/(1+(x[0]-3.0)*(x[0]-3.0)+(x[1]-2.0)*(x[1]-2.0)))
f=f1()
a=[-1.0,-1.0]
b=[1.0,1.0]
f=f1()
xmin=0.0
xmax=2.0
Nx=100
ymin=0.0
ymax=2.0
Ny=100
PLabel="Surface plot f(x,y)"
XLabel="x"
YLabel="y"
ZLabel="z"
P3D.plot(f,xmin,xmax,Nx,ymin,ymax,Ny,PLabel,XLabel,YLabel,ZLabel)
```

```
runfile('E:/okul/SCO1/f_xjtest1.py', wdir='E:/okul/SCO1')
Reloaded modules: f_xj, plot3D1
```

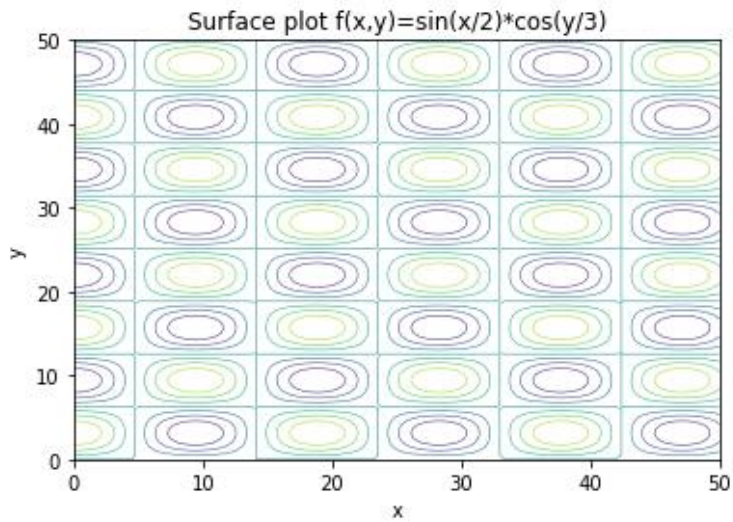


Plotting contour_plot

$$z(x,y) = sin\left(\frac{x}{2}\right) cos(\frac{y}{3})$$

```
from mpl_toolkits import mplot3d
import numpy as np
import matplotlib.pyplot as plt
from f_xj import *

#Function to plot
class f2(f_xj):
  def func(self,x):
      y=sin(x[0]/2.0)*cos(x[1]/3.0)
      return y;

# Creating dataset
def contour_plot(f,xmin,xmax,Nx,ymin,ymax,Ny,PLabel,XLabel,YLable,ZLabel):
    #f=f2()
    x=[0.0 for j in range(Nx)]
    y=[0.0 for j in range(Ny)]
    z=[[0.0 for j in range(Ny)] for i in range(Nx)]
    dx=(xmax-xmin)/(Nx-1)
    dy=(ymax-ymin)/(Ny-1)
    for i in range(Nx):
        for j in range(Ny):
            x[i]=xmin+dx*i
            y[j] = ymin+dy*j
```

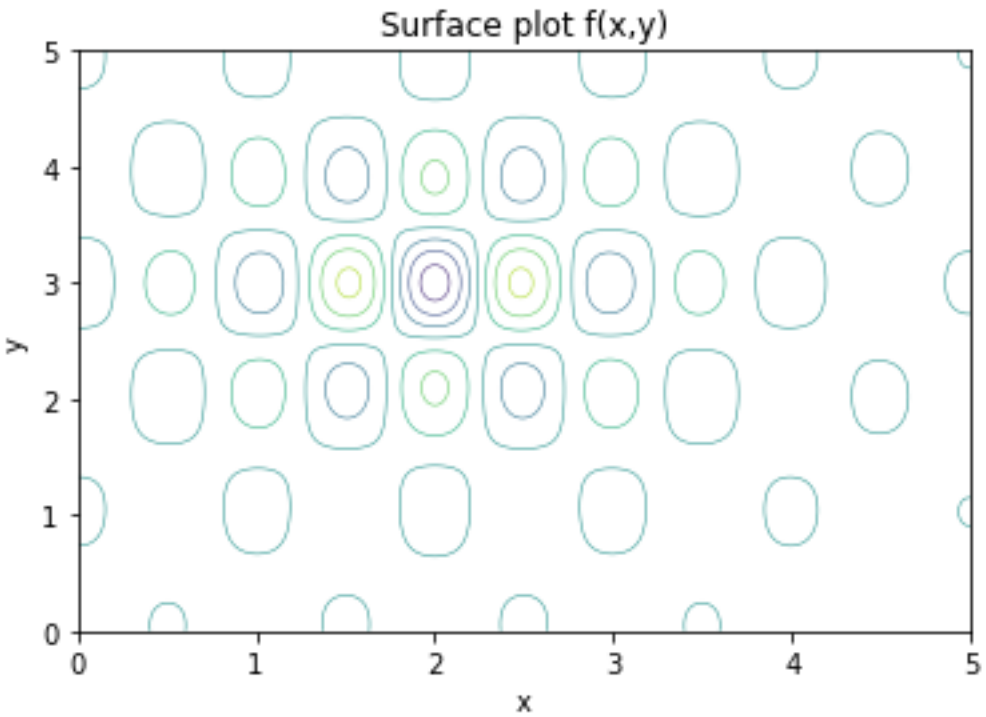
```
            z[i][j]=f.funcxy(x[i],y[j])
    xi=np.array(x)
    yi=np.array(y)
    xi, yi = np.meshgrid(xi, yi)
    zi=np.array(z)
    fig, ax = plt.subplots(1, 1)
    # plots contour lines
    ax.contour(xi, yi, zi,linewidths=0.5)
    ax.set_title(PLabel)
    ax.set_xlabel(XLabel)
    ax.set_xlim(xmin, xmax)
    ax.set_ylabel(YLabel)
    ax.set_xlim(ymin, ymax)
    plt.show()

f=f2()
xmin=0.0
xmax=50.0
Nx=100
ymin=0.0
ymax=50.0
Ny=100
PLabel="Surface plot f(x,y)=sin(x/2)*cos(y/3)"
XLabel="x"
YLabel="y"
ZLabel="z=sin(x)*cos(y)"
contour_plot(f,xmin,xmax,Nx,ymin,ymax,Ny,PLabel,XLabel,YLabel,ZLabel)
```



$$z(x,y) = (1 + cos(\pi(x-3) * \frac{cos(2\pi(y-2)}{1+(x-3)^2+(y-2)^2}$$

```
 from mpl_toolkits import mplot3d
import numpy as np
import matplotlib.pyplot as plt
from f_xj import *

# Creating dataset
def contour_plot(f,xmin,xmax,Nx,ymin,ymax,Ny,PLabel,XLabel,YLable,ZLabel):
    #f=f2()
    x=[0.0 for j in range(Nx)]
    y=[0.0 for j in range(Ny)]
    z=[[0.0 for j in range(Ny)] for i in range(Nx)]
    dx=(xmax-xmin)/(Nx-1)
    dy=(ymax-ymin)/(Ny-1)
    for i in range(Nx):
        for j in range(Ny):
            x[i]=xmin+dx*i
            y[j] = ymin+dy*j
            z[i][j]=f.funcxy(x[i],y[j])
    xi=np.array(x)
    yi=np.array(y)
    xi, yi = np.meshgrid(xi, yi)
    zi=np.array(z)
    fig, ax = plt.subplots(1,1)
```

```
    # plots contour lines
    ax.contour(xi, yi, zi,linewidths=0.5)
    ax.set_title(PLabel)
    ax.set_xlabel(XLabel)
    ax.set_xlim(xmin, xmax)
    ax.set_ylabel(YLabel)
    ax.set_xlim(ymin, ymax)
    plt.show()

class f1(f_xj):func=lambda self,x:-3.0*(1.0+cos(pi*(x[0]-3.0))*cos(2.0*pi*(x[1]-2.0))/(1+(x[0]-3.0)*(x[0]-3.0)+(x[1]-
2.0)*(x[1]-2.0)))
f=f1()
xmin=0.0
xmax=5.0
Nx=500
ymin=0.0
ymax=5.0
Ny=500
PLabel="Surface plot f(x,y)"
XLabel="x"
YLabel="y"
ZLabel="z=sin(x)*cos(y)"
contour_plot(f,xmin,xmax,Nx,ymin,ymax,Ny,PLabel,XLabel,YLabel,ZLabel)
```



1.9 READING AND WRITING DATA FROM A SEQUENTIAL FILE

In order to read/write data into a sequaential file open statement will be used. This statement has the following form:
File_object=open("filename","mode")
Where "filename" is the name of input/output file and mode will take the values of

- '***r***' – Read mode which is used when the file is only being read
- '***w***' – Write mode which is used to edit and write new information to the file (any existing files with the same name will be erased when this mode is activated)
- '***a***' – Append mode, which is used to add new data to the end of the file; that is new information is automatically amended to the end
- '***r+***' – Special read and write mode, which is used to handle both actions when working with a file

```
path = 'e:/okul/SCO1/b.txt'
file1 = open(path,'r')
days = file1.read()
print(days)
```

```
runfile('E:/okul/SCO1/read1.py', wdir='E:/okul/SCO1')

1.234 3.24 5.67 7.78
```

```
path = 'e:/okul/SCO1/d.txt'
file1 = open(path,'w')
s="once upon a time, in a country far far away"
file1.write(s)
file1.close()
file1 = open(path,'r')
s1 = file1.read()
print(s1)
```

```
runfile('E:/okul/SCO1/write1.py', wdir='E:/okul/SCO1')
once upon a time, in a country far far away
```

Reading file b.txt (first two varibles as x,y in each line)

File b.txt

```
1.234 3.24 5.67 7.78
1.345 5.25 5.67 7.78
3.748 6.38 5.67 7.78
2.175 5.24 5.67 7.78
1.222 3.45 5.67 7.78
```

```
path = 'b.txt'
file1 = open(path,'r')
file1.seek(0,2)  #jumps to the end
endLocation=file1.tell()
file1.seek(0) #jumps to the start
nn=0
while True:
   line=file1.readline()
   nn+=1
   if  file1.tell()==endLocation:
      break
x=[0.0 for i in range(nn)]
y=[0.0 for i in range(nn)]
file1.seek(0) #jumps to the start
for i in range(nn):
   line=file1.readline()
   ee=line.split()
   x[i]=float(ee[0])
   y[i]=float(ee[1])
   print(x[i]," ",y[i])
file1.close()
```

```
runfile('E:/okul/SCO1/read2.py', wdir='E:/okul/SCO1')
1.234   3.24
1.345   5.25
```

```
3.748   6.38
2.175   5.24
1.222   3.45
```

```
def read_array2D(d):
    path = d
    file1 = open(path,'r')
    file1.seek(0,2)  #jumps to the end
    endLocation=file1.tell()
    file1.seek(0) #jumps to the start
    nn=0
    m=0
    while True:
        line=file1.readline()
        ee=line.split()
        nn+=1
        if  file1.tell()==endLocation:
            break
    m=len(ee)
    print("m=",m)
    a=[[0.0 for i in range(nn)] for j in range(m)]
    file1.seek(0) #jumps to the start
    for i in range(nn):
        line=file1.readline()
        ee=line.split()
        for j in range(m):
            a[j][i]=float(ee[j])
    file1.close()
    return a

a=read_array2D('b.txt')
print(a)
```

```
runfile('E:/okul/SCO1/read3.py', wdir='E:/okul/SCO1')
m= 4
[[1.234, 1.345, 3.748, 2.175, 1.222], [3.24, 5.25, 6.38, 5.24, 3.45], [5.67, 5.67, 5.67, 5.67, 5.67], [7.78, 7.78, 7.78, 7.78, 7.78]]
```

2. TAYLOR SERIES AND SERIES APPROXIMATION TO FUNCTIONS

2.1 TAYLOR SERIES

Taylor equation can be considered the roots of many numerical analysis methods. It will give us serial. **Taylor series** is a representation of a function as an infinite sum of terms calculated from the values of its derivatives at a single point. It is named after the English mathematician Brook Taylor. If the series is centered at zero, the series is also called a **Maclaurin series**, named after the Scottish mathematician Colin Maclaurin. It is common practice to use a finite number of terms of the series to approximate a function. Taylor series around a point a can be written as

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f^{(3)}(a)}{3!}(x-a)^3 + \cdots$$

Or if it is written in summation notation

$$f(x) = \sum_{n=0}^{\propto} \frac{f^{(n)}(a)}{n!}(x-a)^n$$

Or if a is replaces with x^k and x is replaced with x^{k+1} equation becomes

$$f(x^{k+1}) = \sum_{n=0}^{\propto} \frac{f^{(n)}(x^k)}{n!}(x^{k+1}-x^k)^n$$

+..

Computers could not calculate anything more complex than addition, the rest usually accomplish through Taylor series approximations. For example function $f(x)=e^x$ can be approximated as

$$f(x) = e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots = 1 + \mathrm{x} + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} + \frac{x^5}{120}$$

Some of the formulation used in Numerical analysis is direct derivation from the Taylor series. For example for the Newton root finding equation, above linear terms of the equation is ignorede and function value is replaced with 0

$f(x^{k+1}) = 0 = f(x^k) + f'(x^k)(x^{k+1} - x^k)$ Will yield to

$$x^{k+1} = x^k - \frac{f(x^k)}{f'(x^k)}$$

Taylor series can also be used to estimate error. The series usually used up to a certain term and error will be proportinal to one upper term in the series. For example If exponetial function is taken as

$$f(x) = e^x = 1 + \mathrm{x} + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24}$$

The error term will be proportional to the next term of the series

$$Error \cong \frac{x^5}{120}$$

2.2 SERİES SOLUTIONS OF FUNCTIONS

Digital computers can only calculate addition process, therefore most of the mathematical expressions are calculated by using series. Series solutions of a few functions will be listed in here.

1. $\ln(x) = 2\sum_{k=1}^{\infty} \frac{1}{2k-1}\left(\frac{x-1}{x+1}\right)^{2k-1} \quad [0 < x]$

2. $\exp(\mathrm{x}) = \sum_{k=0}^{\infty} \frac{x^k}{k!}$

3. $\exp(-\mathrm{x}) = \sum_{k=1}^{\infty} (-1)^k \frac{x^k}{k!}$

4. $\exp(-\mathrm{x}^2) = \sum_{k=1}^{\infty}(-1)^k \frac{x^{2k}}{k!}$

5. $\exp(\mathrm{x})(1+x) = \sum_{k=1}^{\infty} \frac{x^k(k+1)}{k!}$

6. $a^x = e^{(xln(a))} = \sum_{k=1}^{\infty} \frac{[xln(a)]^k}{k!}$

7. $\sum_{k=0}^{\infty} a^{kx} = \frac{1}{1-a^x}$ [a>1 and x<0 or 0<a<1 x>0]

8. $\sin(\mathrm{x}) = \sum_{k=0}^{\infty}(-1)^k \frac{x^{2k+1}}{(2k+1)!}$

9. $\sin h(\mathrm{x}) = \sum_{k=0}^{\infty} \frac{x^{2k+1}}{(2k+1)!}$

10. $\cos(\mathrm{x}) = \sum_{k=0}^{\infty}(-1)^k \frac{x^{2k}}{(2k)!}$

11. $\cos \mathrm{h}(\mathrm{x}) = \sum_{k=0}^{\infty} \frac{x^{2k}}{(2k)!}$

12. $\tan(\mathrm{x}) = \frac{\sin(x)}{\cos(x)}$

13. $\tanh(\mathrm{x}) = \frac{\sinh(x)}{\cos \mathrm{h}(x)}$

14. $\arcsin(\mathrm{x}) = \sum_{k=0}^{\infty} \frac{(2k)!x^{2k+1}}{2^{2k}(k!)^2(2k+1)}$ $\quad x^2 \leq 1$

15. $\operatorname{arcsinh}(\mathrm{x}) = \sum_{k=0}^{\infty}(-1)^k \frac{(2k)!x^{2k+1}}{2^{2k}(k!)^2(2k+1)}$ $\quad x^2 \leq 1$

16. $\arctan(\mathrm{x}) = \sum_{k=0}^{\infty}(-1)^k \frac{x^{2k+1}}{(2k+1)}$ $\quad x^2 \leq 1$

17. $\operatorname{arctanh}(\mathrm{x}) = \sum_{k=0}^{\infty} \frac{x^{2k+1}}{(2k+1)}$ $\quad x^2 \leq 1$

18. $\operatorname{erf}(\mathrm{x}) = \frac{2}{\sqrt{\pi}} \sum_{n=0}^{\infty}(-1)^n \frac{x^{2n+1}}{n!(2n+1)}$

19. $\pi = \sum_{k=0}^{\infty} \frac{1}{16^k}\left(\frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{2}{8k+6}\right)$ Bailey, Borwein Plouffe(BBP) formula[90]

20. $J_n(z) = \left(\frac{1}{2z}\right) \sum_{k=1}^{\infty}(-1)^k \frac{\left(\frac{1}{2z}\right)^{2k}}{k!\,\Gamma(n+1)}$ Bessel function

21. $\Gamma(z) = \int_{z=0}^{\infty} t^{z-1}e^{-t}dt = \frac{e^{-\gamma z}}{z} \prod_{n=1}^{\infty}\left(1+\frac{z}{n}\right)^{-1} e^{z/n}$ where

$\gamma = 0.57721566490153286\ldots$ is the Euler-Mascheroni constant

22. $Y_n\,(z) = \frac{J_n(z)\cos(n\pi) - J_{-n}(z)}{\sin(n\pi)}$ Bessel function

23. $\gamma = \lim_{n\to\infty}\left(-log(n) + \sum_{k=1}^{n} \frac{1}{k}\right)$ Euler-Mascheroni constant

$$\gamma = 0.57721566490153286060651209008240243104215933593992\ldots$$

24. $K = \frac{3}{64}\sum_{k=0}^{\infty}\frac{(-1)^k}{64^k}\left(\frac{32}{(12k+1)^2} - \frac{32}{(12k+2)^2} - \frac{32}{(12k+3)^2} - \frac{8}{(12k+5)^2} - \frac{16}{(12k+6)^2} - \frac{4}{(12k+7)^2} - \frac{4}{(12k+9)^2} - \frac{2}{(12k+10)^2} + \frac{1}{(12k+11)^2}\right)$ Catalan's constant (K=0.915965594177)

25. $\frac{1}{\pi} = \frac{2\sqrt{2}}{9801}\sum_{k=0}^{\infty}\frac{(4k)!(1103+26390k)}{(k!)^4(396^{4k})}$ Srinavasa Ramanujan π calculation series

26. $\frac{1}{\pi} = \frac{12}{640320^{3/2}}\sum_{k=0}^{\infty}\frac{(6k)!(13591409+545140134k)}{(3k)!(k!)^3(-640320)^{3k}}$ Chudnovsky formula π calculation series

27. $\pi = \sum_{k=0}^{\infty}\frac{1}{16^k}\left(\frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{1}{8k+6}\right)$ Bailey-Borvein-Plouffe formula π calculation series

Exponential function $\exp(\mathrm{x}) = \sum_{k=1}^{\infty}\frac{x^k}{k!}$

```
from math import *
def exp(x):
    total=0;
    factorial=1;
    power=1;
    for i in range(1,100):
      total=total+power/factorial;
      power=power*x;
      factorial=factorial*i;
    return total;
r=exp(1.0);
print("r=",r);
```

```
runfile('E:/okul/SCO1/exp.py', wdir='E:/okul/SCO1')
r= 2.7182818284590455
```

Natural logarithm $\ln(x) = 2\sum_{k=1}^{\infty}\frac{1}{2k-1}\left(\frac{x-1}{x+1}\right)^{2k-1} \quad [0 < x]$

```
from math import *

def ln(x):
   total=0;
   y=(x-1.0)/(x+1.0);
   power=y;
   for k in range(1,100):
      total=total+power/(2.0*k-1.0);
      power=power*y*y;
   return 2*total;
e=r=2.7182818284590455;
r=ln(e);
print("r=",r);
```

```
runfile('E:/okul/SCO1/func2.py', wdir='E:/okul/SCO1')
r= 0.9999999999999999
```

ln(x)

k	x=	2,7182818285
	y=(x-1)/(x+1)	0,4621171573
	power	total
0	0,4621171573	0
1	0,0986861666	0,9242343145
2	0,0210746546	0,9900250922
3	0,0045005403	0,9984549541
4	0,0009611006	0,9997408227
5	0,0002052452	0,9999544006
6	4,38306E-005	0,9999917179
7	9,36012E-006	0,9999984611
8	1,99887E-006	0,9999997091
9	4,26864E-007	0,9999999443
10	9,11578E-008	0,9999999892
11	1,94670E-008	0,9999999979
12	4,15721E-009	0,9999999996
13	8,87782E-010	0,9999999999
14	1,89588E-010	1
15	4,04869E-011	1
16	8,64608E-012	1
17	1,84639E-012	1
18	3,94301E-013	1
19	8,42038E-014	1
20	1,79819E-014	1

$$\sin(\mathrm{x}) = \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k+1}}{(2k+1)!}$$

```
import math;

def sin(x):
    total=0;
    factorial=1;
    power=x;
    plusminus=1;
    i=0;
    for k in range(1,80):
        total=total+power/factorial*plusminus;
        power=power*x*x;
        i=2*k+1;
        factorial=factorial*(i-1)*i;
        plusminus*=-1;
    return total;

pi=math.pi;
x=pi/3.0;
print ("sin="+str(sin(x))+"math.sin = "+str(math.sin(x)));
```

```
runfile('E:/okul/SCO1/sin.py', wdir='E:/okul/SCO1')
sin=0.8660254037844385math.sin = 0.8660254037844386
```

$$\arcsin(\mathrm{x}) = \sum_{k=0}^{\infty} \frac{(2k)!\, x^{2k+1}}{2^{2k}(k!)^2(2k+1)} \qquad x^2 \leq 1$$

```
import math;

def asin(x):
    total=0;
    factorial1=1;
    factorial2=1;
    power=x;
    power2=1;
    i=0;
    for k in range(1,80):
        total=total+power*factorial1/(power2*factorial2*factorial2)/(2.0*(k-1)+1.0);
        power*=x*x;
        power2*=4;
        i=2*k;
        factorial1=factorial1*(i-1)*i;
```

```
        factorial2=factorial2*k;
    return total;

pi=math.pi;
x1=pi/3.0;
x2=math.sin(x1);
print ("asin="+str(asin(x2))+" math.asin = "+str(math.asin(x2)));
```

```
runfile('E:/okul/SCO1/asin.py', wdir='E:/okul/SCO1')
asin=1.0471975511964204 math.asin = 1.0471975511965976
```

$$a^x = e^{xln(a)}$$

```
from math import *

def ln(x):
    total=0
    y=(x-1.0)/(x+1.0)
    power=y
    for k in range(1,100):
        total=total+power/(2.0*k-1.0)
        power=power*y*y
    return 2*total;
def exp(x):
      total=0;
      factorial=1;
      power=1;
      for i in range(1,100):
        total=total+power/factorial;
        power=power*x;
        factorial=factorial*i;
      return total;
def pow(a,x):
    return exp(x*log(a));

r=pow(2.0,3.0);
print("r=",r);
```

```
runfile('E:/okul/SCO1/pow.py', wdir='E:/okul/SCO1')
r= 7.999999999999999
```

1. $J_n(z) = \left(\frac{1}{2z}\right) \sum_{k=1}^{\infty} (-1)^k \frac{\left(\frac{1}{2z}\right)^{2k}}{k!\,\Gamma(n+1)}$ Bessel function

Note : Γ(x) gamma function existed in python math library as math.gamma(x) and natural logarithm of gamma function as math.lgamma(x)

```
from math import *

def J(v,x):
    JJ=0
    a=1.0
    fact=1
    b=0.0
    if x <=20:
        x5=1
        lna=0.0;
        x1=v*log(0.5*x)
        lna1=log(0.25*x*x)
        for k in range(0,100):
            x2=lna
            x3=lgamma(k+1)
            x4=lgamma(v+k+1)
            b=x1+x2-x3-x4
            b=x5*exp(b)
            lna=lna+lna1
            x5=-1*x5
            JJ=JJ+b
    elif (v==0 and x==0):
```

```
        JJ=1
    else:
        JJ=sqrt(2.0/PI/x)*cos(x-0.5*v*PI-0.25*PI)
    return JJ
x=0.1
y=J(0.0,x)
print("J(0.0,",x,") = ",y)
```

```
runfile('E:/okul/SCO1/bessel.py', wdir='E:/okul/SCO1')
J(0.0, 0.1 ) =  0.9975015620660401
```

$$\mathrm{erf(x)} = \frac{2}{\sqrt{\pi}} \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{n!\,(2n+1)}$$

```
import  math

def erf(x):
   A=2.0/math.sqrt(math.pi)
   total=0.0
   plusminus=1
   power=x
   factorial=1.0
   for n in range(1,200):
        total=total+power*plusminus/factorial/(2.0*(n-1)+1.0)
        power=power*x*x
        factorial=factorial*n
        plusminus*=plusminus*(-1);
   return A*total

x=0.1
y1=erf(x)
y=math.erf(x)
print("x=",x,"y1=",y,"y=",y)
```

```
runfile('E:/okul/SCO1/erf1.py', wdir='E:/okul/SCO1')
x= 0.1 y1= 0.1124629160182849 y= 0.1124629160182849
```

Some of the algorithms might not be as clear as a directly given function. For example let us look at the Russian peasant multiplication problem:

Take two integer for example 21 and 37 then double the first one(left hand side) and divide by two the second one(right side) ignore remaining in division process and stop when the right hand side reached to 1 and then cross each line where the right side number is even add up remaining left hand side numbers for example. If the given integer numbers to be multiplied is 21 and 17, the process will follow the path given below:

21 37
~~42 18~~ cross out (18 is even)
84 9
~~168 4~~ cross out(4 is even)
~~336 2~~ cross out(2 is even)
672 1
then 21+84+672=777
21*37=777

Now let us convert this to an algorithmic structure:

```
 from math import *

def isOdd(x):
  return bool(x %2 != 0 );

def multiply(x1,x2):
  y2=x2;
  y1=x1;
  b2=0;
  total=0;
  while y2>=1:
```

```
    b2=isOdd(y2);
    if y2 %2 != 0 :
      total=total+y1;
    print("total=",total,"y1=",y1,"y2=",y2);
    y1=y1*2;
    y2=(int)(y2/2);
  return total;

x1=21;
x2=37;
x=multiply(x1,x2);
print("Russian multiplication :\nx = ",x)
```

```
runfile('E:/okul/SCO1/russian.py', wdir='E:/okul/SCO1')
total= 21 y1= 21 y2= 37
total= 21 y1= 42 y2= 18
total= 105 y1= 84 y2= 9
total= 105 y1= 168 y2= 4
total= 105 y1= 336 y2= 2
total= 777 y1= 672 y2= 1
Russian multiplication :
x =  777
```

Euler-Mascheroni constant $\gamma = 0.57721566490153286060651209008240243104215933593992\ldots$

$$\gamma = \lim_{n\to\infty}\left(-log(n) + \sum_{k=1}^{n}\frac{1}{k}\right)$$

```
#Euler-Mascheroni constant
from math import *
from f_x import *

def euler_contant():
    n=100000
    total=0.0
    for k in range(1,n):
        total+=1.0/k
    return -log(n)+total

print(euler_contant())
```

```
runfile('D:/okul/SCO1/euler_contant.py', wdir='D:/okul/SCO1')
Reloaded modules: f_x
0.5772106648931068
```

$\pi = \sum_{k=0}^{\infty}\frac{1}{16^k}\left(\frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{1}{8k+6}\right)$ Bailey-Borvein-Plouffe formula π calculation series

```
#             Bailey-Borvein-Plouffe formula π calculation series
from math import *
from f_x import *

def pi():
    n=1000
    total=4.0-2.0/4.0-1.0/5.0-1.0/6.0
    pow=1.0;
    for k in range(1,n):
        pow*=16
        k8=8.0*k
        total+=1.0/pow*(4.0/(k8+1.0)-2.0/(k8+4.0)-1.0/(k8+5.0)-1.0/(k8+6.0))
    return total

print("pi=",pi())
```

```
runfile('D:/okul/SCO1/pi_Plouffe.py', wdir='D:/okul/SCO1')
Reloaded modules: f_x
pi= 3.141592653589793
```

	Bailey-Borvein-Plouffe formula	π
k	pow	total

0	1	3.13333333333333000000000E+00
1	16	3.14142246642247000000000E+00
2	256	3.14158739034658000000000E+00
3	4096	3.14159245756744000000000E+00
4	65536	3.14159264546034000000000E+00
5	1048576	3.14159265322809000000000E+00
6	16777216	3.14159265357288000000000E+00
7	2.68E+08	3.14159265358897000000000E+00
8	4.29E+09	3.14159265358975000000000E+00
9	6.87E+10	3.14159265358979000000000E+00
10	1.1E+12	3.14159265358979000000000E+00

Further mathematical functions are defined as class Mathd. Some of the methods defined here can be used in numerical math.

```
from math import *
from cmath import *
from random import *
import matplotlib.pyplot as plt

class Mathd():
  def __init__(self):
    self.name = 'mathd'

  # First derivative
  def df(f,x):
    h=0.001
    return (-f(x+2.0*h)+8.0*f(x+h)-8.0*f(x-h)+f(x-2.0*h))/(12.0*h);

  # second derivative
  def d2f(f,x):
    dx=0.001
    return (-1.0/12.0*f(x-2.0*dx)+4.0/3.0*f(x-dx)-5.0/2.0*f(x)+4.0/3.0*f(x+dx)-1.0/12.0*f(x+2.0*dx))/dx

  # nth derivative
  def dnf(f,x,n):
    dx=0.001
    dxh=1.0/dx
    df=0.0;
    if n==0:  df=f(x);
    elif n==1: df=(3.0*f(x-4.0*dx)-32.0*f(x-3.0*dx)+168.0*f(x-2.0*dx)-672.0*f(x-dx)+672.0*f(x+dx)-
168.0*f(x+2.0*dx)+32.0*f(x+3.0*dx)-3.0*f(x+4.0*dx))/840.0*dxh;
    elif n==2: df=(-14350.0*f(x)-9.0*f(x-4.0*dx)+128*f(x-3.0*dx)-1008*f(x-2.0*dx)+8064*f(x-dx)+8064.0*f(x+dx)-
1008.0*f(x+2.0*dx)+128.0*f(x+3.0*dx)-9.0*f(x+4.0*dx))/5040.0*dxh*dxh;
    elif n==3:
      df=(-7.0*f(x-4.0*dx)+72.0*f(x-3.0*dx)-338.0*f(x-2.0*dx)+488.0*f(x-dx)-488.0*f(x+dx)+338.0*f(x+2.0*dx)-
72.0*f(x+3.0*dx)+7.0*f(x+4.0*dx))/240.0*dxh*dxh*dxh;
    elif n==4:
      df=(2730.0*f(x)+7.0*f(x-4.0*dx)-96.0*f(x-3.0*dx)+676.0*f(x-2*dx)-1952.0*f(x-dx)-1952.0*f(x+dx)+676.0*f(x+2.0*dx)-
96.0*f(x+3.0*dx)+7.0*f(x+4.0*dx))/240.0*dxh*dxh*dxh*dxh;
    elif n==5:
      df=(f(x-4.0*dx)-9.0*f(x-3.0*dx)+26.0*f(x-2.0*dx)-29.0*f(x-dx)+29.0*f(x+dx)-26.0*f(x+2.0*dx)+9.0*f(x+3.0*dx)-
f(x+4.0*dx))/6.0*dxh*dxh*dxh*dxh*dxh;
    return df;
#polynomial function
# c1[0]+c1[1]*x+c1[2]*x^2+..
  def f_poly(c1,x):
    # this function calculates the value of
    # c least square curve fitting function
    n=len(c1);
    if n!=0.0:
      ff=c1[n-1];
      for i in range(n-2,-1,-1):
        ff=ff*x+c1[i];
    else:
      ff=0,0;
    return ff;
#derivative of polynomial function
# c1[0]+c1[1]*x+c1[2]*x^2+..
  def df_poly(c1,x):
    h=0.001
```

```
    return (-Mathd.f_poly(c1,(x+2.0*h))+8.0*Mathd.f_poly(c1,(x+h))-8.0*Mathd.f_poly(c1,(x-h))+Mathd.f_poly(c1,(x-
2.0*h)))/(12.0*h);
#second derivative of polynomial function
# c1[0]+c1[1]*x+c1[2]*x^2+..
  def d2f_poly(c1,x):
    dx=0.001
    y1=-1.0/12.0*Mathd.f_poly(c1,(x-2.0*dx))+4.0/3.0*Mathd.f_poly(c1,(x-dx));
    y2=-5.0/2.0*Mathd.f_poly(c1,x)+4.0/3.0*Mathd.f_poly(c1,(x+dx))-1.0/12.0*Mathd.f_poly(c1,(x+2.0*dx));
    return (y1+y2)/dx;
#Leguerre polynomial root finding method
  def LeGuerre(c1):
    # LeGuerre root finding method of polynomials
    maxit=500;
    iter=0;
    es=0.0000001;
    ea=1.1*es;
    n=len(c1)-1;
    x=complex(1.0,0.0);
    while ea>es and iter<maxit:
      dfx=Mathd.df_poly(c1,x);
      fx=Mathd.f_poly(c1,x);
      d2fx=Mathd.d2f_poly(c1,x);
      G=dfx/fx;
      H=G*G-d2fx/fx;
      y=sqrt((n*H-G*G)*(n-1.0));
      z1=G+y;
      z2=G-y;
      zbool=z1.real*z1.real+z1.imag*z1.imag> z2.real*z2.real+z2.imag*z2.imag;
      if zbool: z=z1;
      else: z=z2;
      aa=n/z;
      x=x-aa;
      iter=iter+1;
      if abs(fx)<1e-15: return x;
    if iter>=maxit: print("Maximum number of iteration is exceeded \n result might not be valid");
    return x;

#False position root finding method
  def false_position(f,xl,xu):
    # false position root finding method
    maxit=100
    iter=0
    es=0.0000001
    ea=1.1*es
    s=""
    fxl= f(xl)
    fxu= f(xu)
    xr=xl
    while ea>es and iter<maxit:
      xold=xr
      xr=xu-fxu*(xl-xu)/(fxl-fxu);
      fxr= f(xr);
      iter=iter+1;
      if xr!=0:
        ea=abs((xr-xold)/xr)*100
      test= fxl*fxr;
      if test==0.0 :      ea=0;
      elif test<0.0: xu=xr;fxu=fxr;
      elif test>0:    xl=xr;fxl=fxr;
      else :          ea=0;
      if iter>=maxit: print("Maximum number of iteration is exceeded \n result might not be valid");
    return xr
#bisection root finding method
  def bisection(f,xl,xu):
    # bisection root finding method
    maxit=100
    iter=0
    es=0.0000001
    ea=1.1*es
    s=""
    fxl= f(xl)
    fxu= f(xu)
    xr=xl
    while ea>es and iter<maxit:
      xold=xr
      xr=(xl+xu)/2.0;
```

```
        fxr= f(xr);
        iter=iter+1;
        if xr!=0:
          ea=abs((xr-xold)/xr)*100
        test= fxl*fxr;
        if test==0.0 :      ea=0;
        elif test<0.0: xu=xr;fxu=fxr;
        elif test>0:    xl=xr;fxl=fxr;
        else :          ea=0;
        if iter>=maxit: print("Maximum number of iteration is exceeded \n result might not be valid")
    return xr
#secant root finding method
 def secant(f,x):
    # secant root finding method
    nmax=100
    tolerance=1.0e-10
    for i in range(0,nmax):
        fx= f(x)
        dfx=Mathd.df(f,x)
        x=x-fx/dfx
        if abs(fx)<tolerance: return x
    return x
#secant optimisation method
 def secant_opt(f,x):
    # secant optimisation method
    nmax=100
    tolerance=1.0e-10
    for i in range(0,nmax):
        fx= Mathd.df(x)
        dfx=Mathd.d2f(f,x)
        x=x-fx/dfx
        if abs(fx)<tolerance: return x
    return x
 #newton-Raphson root finding method
 def newton(f,df,x):
    # Newton-Raphson root finding method
    nmax=100
    tolerance=1.0e-10
    for i in range(0,nmax):
        fx= f(x);
        dfx=df(x)
        x=x-fx/dfx
        if abs(fx)<tolerance: return x
    return x
 #muller root finding method
 def muller(f,x0,x1,x2):
    # Finding real roots by using Muller method
    iter=0;
    es1=0.001;
    es=es1;
    ea=1.1*es;
    maxit = 100;
    xr=x0;
    for iter in range(0,maxit):
      fx0=f(x0);
      fx1=f(x1);
      fx2=f(x2);
      h0=x1-x0;
      h1=x2-x1;
      d0=(fx1-fx0)/h0;
      d1=(fx2-fx1)/h1;
      a=(d1-d0)/(h1+h0);
      b=a*h1+d1;
      c=fx2;
      determinant=b*b-4.0*a*c;
      if determinant<=0:  determinant=0;
      else: determinant=sqrt(determinant);
      if abs(b+determinant)>  abs(b-determinant): den=b+determinant;
      else: den=b-determinant;
      hr=-2*c/den;
      xr=x2+hr;
      ea=abs((xr-x2)/xr)*100;
      x0=x1;
      x1=x2;
      x2=xr;
      if ea<es: return xr
```

```
    if iter>=maxit:  print('Warning : Maximum number of iteration is exceeded result may not be valis');
    return xr;
# Hasan root finding method
 def hasan(f,x):
     # finding real roots by using hasan method third order
     nmax=500;
     r=3;
     tolerance=1.0e-15;
     for iter in range(0,nmax):
         fx=f(x);
         dfx=Mathd.df(f,x);
         d2fx=Mathd.dnf(f,x,2);
         x=x-fx/(dfx+1.0/r*dfx*log((dfx*dfx-fx*d2fx)/(dfx*dfx)));
         if abs(fx)<tolerance: return x;
     return x;
 # inverse interpolation (Lagrange interpolation)  root finding method
 def inverseinter2Droot(f,x1,x2,x3):
    #Inverse quadratic Lagrange interpolation
    #root finding method
    p=1;
    maxit=100
    iter=0;
    tol=1.0e-10;
    es=0.00000001;
    ea=0.001;
    f1=f(x1);
    f2=f(x2);
    f3=f(x3);
    if f1==0: p=x1;
    elif f2==0: p=x2;
    elif f3==0: p=x3;
    if f1*f3>0: print("Root is not existed in the given region");
    while (ea>es) and (iter<maxit) and f1!=0 and f2!=0 and f3!=0:
       p=-(f2*f3*x1*(f2-f3)+f3*f1*x2*(f3-f1)+f1*f2*x3*(f1-f2))/((f1-f2)*(f2-f3)*(f3-f1));
       fp=f(p);
       ea=abs(x3-p);
       if abs(f3)<tol:  p=x3;
       if p>x3:  x1=x3;f1=f3;x3=p;f3=fp;
       elif p<x3: x2=x3;f2=f3;x3=p;f3=fp;
       iter=iter+1;
    if iter>=maxit:  print("Warning : Maximum number of iteration is exceeded");
    return p;
  #Halley qudratic order root finding methos
 def halley(f,x):
    # secant quadratic root finding method
    nmax=100
    tolerance=1.0e-10
    for i in range(0,nmax):
       fx= f(x);
       dfx=Mathd.df(f,x)
       d2fx=Mathd.d2f(f,x)
       x=x-2.0*fx*dfx/(2.0*dfx*dfx-fx*d2fx)
       print("x",x,"fx",fx,"dfx",dfx,"d2fx",d2fx);
       if abs(fx)<tolerance: return x
    return x
 #Halley optimisation method
 def halley_opt(f,x):
    # halley optimisation method
    nmax=100
    tolerance=1.0e-10
    for i in range(0,nmax):
       fx=Mathd.df(x);
       dfx=Mathd.d2f(f,x)
       d2fx=Mathd.dnf(f,x,3)
       x=x-2.0*fx*dfx/(2.0*dfx*dfx-fx*d2fx)
       if abs(fx)<tolerance: return x
    return x
 #root of cubic equations
 def cubicroot(d):
     x=[[0 for i in range(3)] for j in range(2)]
     a=d[2]/d[3]
     b=d[1]/d[3]
     c=d[0]/d[3]
     Q=(a*a-3.0*b)/9.0
     y1=2.0*a*a*a
     y2=-9*a*b
```

```
    y3=27*c
    y4=y1+y2+y3
    R=y4/54.0
    Q3=Q*Q*Q
    R2=R*R
    if R2<Q3:
      qq=-2.0*sqrt(Q);
      theta=acos(R/Math.sqrt(Q3));
      x[0][0]=qq*cos(theta/3.0)-a/3.0;
      x[0][1]=qq*cos((theta-2.0*pi)/3.0)-a/3.0;
      x[0][2]=qq*cos((theta+2.0*pi)/3.0)-a/3.0;
    else:
      A=-pow((R+sqrt(R2-Q3)),(1.0/3.0));
      if A==0:  B=0
      else   :  B=Q/A
      x[0][0]=(A+B)-a/3.0
      x[0][1]=-0.5*(A+B)-a/3
      x[1][1]=sqrt(3.0)/2.0*(A-B)
      x[0][2]=-0.5*(A+B)-a/3
      x[1][2]=-sqrt(3.0)/2.0*(A-B)
    y=[complex(0,0) for i in range(3)];
    y[0]=complex(x[0][0],x[1][0]);
    y[1]=complex(x[0][1],x[1][1]);
    y[2]=complex(x[0][2],x[1][2]);
    return y
#root of quadratic equations
def squareroot(d):
  # square root
  x= [0 for i in range(2)];
  a=d[2];
  b=d[1];
  c=d[0];
  delta=b*b-4*a*c;
  x[0]=(-b+sqrt(delta))/(2*a);
  x[1]=(-b-sqrt(delta))/(2*a);
  return x;
#root of linear equation
def linearroot(d):
  # linear rooot
  b=d[1];
  a=d[0];
  return -d[0]/d[1];

#calculates legendre gauss-coefficients as coefficients of the integral
#for n terms
def gauss_legendre_coefficients(x1,x2,n):
        eps=3e-15
        m=int((n+1.0)/2.0)
        xm=0.5*(x2+x1)
        xl=0.5*(x2-x1)
        nmax=100
        a=[[0.0 for i in range(n)] for j in range(2)]
        for i in range(1,m+1):
                z=cos(pi*((i-0.25)/(n+0.5)))
                for ii in range(nmax):
                        p1=1.0;
                        p2=0.0;
                        for j in range(1,n+1):
                          p3=p2
                          p2=p1
                          p1=(float)(((2.0*j-1.0)*z*p2-(j-1.0)*p3)/j)
                        pp=(float)(n*(z*p1-p2)/(z*z-1.0))
                        z1=z;
                        z=z1-p1/pp
                        if abs(z-z1) < eps: break
                a[0][i-1]=xm-xl*z
                a[0][n-i]=xm+xl*z
                a[1][i-1]=2.0*xl/((1.0-z*z)*pp*pp)
                a[1][n-i]=a[1][i-1]
        return a
#Gauss-Legendre integration
def integral(f,x1,x2,n):
  #integral f1(x)dx
  #integral of a function by using gauss-legendre quadrature
  #between x1 and x2
  a=Mathd.gauss_legendre_coefficients(x1,x2,n)
```

```
    z=0
    for i in range(n):
      z=z+a[1][i]*f(a[0][i])
    return z
# Solution of linear system of Equatio by using Gauss method
def gauss(a,b):
# Gauss elimination method with pivoting
  n=len(a)
  x=[0 for i in range(n)]
  for k in range(n):
    p=k
    buyuk=abs(a[k][k])
    for ii in range(k+1,n):
      dummy=abs(a[ii][k])
      if dummy>buyuk:
         buyuk=dummy
         p=ii
    if p!=k:
      for jj in range(k,n):
        dummy=a[p][jj]
        a[p][jj]=a[k][jj]
        a[k][jj]=dummy
      dummy=b[p]
      b[p]=b[k]
      b[k]=dummy
    for i in range(k+1,n):
       carpan=a[i][k]/a[k][k]
       a[i][k]=0;
       for j in range(k+1,n):
          a[i][j]=a[i][j]-carpan*a[k][j]
       b[i]   =b[i]-carpan*b[k]
  x[n-1]=b[n-1]/a[n-1][n-1]
  for i in range(n-1,-1,-1):
    toplam=0;
    for j in range(i+1,n):
      toplam=toplam+a[i][j]*x[j]
      x[i]=(b[i]-toplam)/a[i][i]
  return x;
# Solution of linear system of Equatio by using Gauss-Jordan method
def gaussjordan(a,b):
#Gauss - Jordan elimination solution of a system of equations
    n=len(a)
    np1=n+1
    x=[]
    C=[[0 for x in range(np1)] for y in range(n)]
    for i in range(n):
      x.append(0.0)
    for i in range(n):
      for j in range(n):
        C[i][j]=a[i][j]
      C[i][n]=b[i]
    for k in range(n):
      for j in range(k+1,np1):
         C[k][j]=C[k][j]/C[k][k]
    for i in range(n):
       if i!=k:
          carpan=C[i][k]
          C[i][k]=0.0
          for j in range(k+1,np1):
             C[i][j]=C[i][j]-carpan*C[k][j]
    for i in range(n):
       x[i]=C[i][n]
    return x;
# Solution of linear system of Equation (3 band matrix) by using Thomas algorithm
# matrix a is given
def thomas(a,r):
    n=len(a);
    f=[0.0 for i in range(n)];
    e=[0.0 for i in range(n)];
    g=[0.0 for i in range(n)];
    x=[0.0 for i in range(n)];
    for i in range(0,n): f[i]=a[i][i];
    for i in range(0,n-1): g[i]=a[i][i+1];
    for i in range(0,n-1): e[i+1]=a[i+1][i];
    for k in range(0,n): e[k]=e[k]/f[k-1];f[k]=f[k]-e[k]*g[k-1];
    for k in range(1,n): r[k]=r[k]-e[k]*r[k-1];
```

```
    x[n-1]=r[n-1]/f[n-1];
    for k in range(n-2,-1,-1):  x[k]=(r[k]-g[k]*x[k+1])/f[k];
    return x;
# Solution of linear system of Equation (3 band matrix) by using Thomas algorithm
#vectors f,e,g is given
def thomas_n(f,e,g,r):
    n=len(f);
    x=[0.0 for i in range(n)];
    for k in range(0,n): e[k]=e[k]/f[k-1];f[k]=f[k]-e[k]*g[k-1];
    for k in range(1,n): r[k]=r[k]-e[k]*r[k-1];
    x[n-1]=r[n-1]/f[n-1];
    for k in range(n-2,-1,-1):  x[k]=(r[k]-g[k]*x[k+1])/f[k];
    return x;
#Golden search (Fİbonnachi) optimisation
def golden(f1,a,b):
  # golden search
  r1=(sqrt(5)-1)/2;
  r2=r1*r1;
  h=b-a;
  ya=f1(a);
  yb=f1(b);
  c=a+r2*h;
  d=a+r1*h;
  yc=f1(c);
  yd=f1(d)
  k=1;
  epsilon=1.0e-10;
  while abs(yb-ya) > epsilon:
    k=k+1;
    if yc<yd:
      b = d;
      yb = yd;
      d = c;
      yd = yc;
      h = b - a;
      c = a + r2 * h;
      yc = f1(c);
    else:
      a = c;
      ya = yc;
      c = d;
      yc = yd;
      h = b - a;
      d = a + r1 * h;
      yd = f1(d);
  p=a;
  yp=ya;
  if yb<ya:
     p=b;
  return p;
#Brent root finding method
def brent(f,a,b):
   #brent root finding method
   maxit=500;
   iter=0;
   tol=1.0e-15;
   es=0.0000001;
   x1=a;f1=f(x1);
   x2=b;f2=f(x2);
   x3=(x1+x2)/2.0;f3=f(x3);
   if f1==0:    return x1;
   elif f2==0:  return x2;
   elif f3==0:  return x3;
   if f1*f2>0: print("No root is existed in the given region");
   p=-(f2*f3*x1*(f2-f3)+f3*f1*x2*(f3-f1)+f1*f2*x3*(f1-f2))/((f1-f2)*(f2-f3)*(f3-f1));
   fp=f(p);
   ea=abs(x3-p);
   while (ea>es) and (iter<maxit):
      if abs(f3)<tol: return x3;
      if (p<x1) and (p>x2):
         p=(x1+x2)/2.0;
         if p>x3:   x1=x3;f1=f3;x3=p;f3=fp;
         elif p<x3: x2=x3;f2=f3;x3=p;f3=fp;
      else:
         if p>x3: x1=x3;f1=f3;x3=p;f3=fp;
         elif p<x3: x2=x3;f2=f3;x3=p;f3=fp;
```

```
        p=-(f2*f3*x1*(f2-f3)+f3*f1*x2*(f3-f1)+f1*f2*x3*(f1-f2))/((f1-f2)*(f2-f3)*(f3-f1));
        fp=f(p);
        ea=abs(x3-p);
      iter=iter+1;
    if iter>=maxit: print("Warning : MAximum number of iteration is exceeded");
    return p;
 #Brent optimisation method
def brent_opt(f,a,b):
    #brent optimisation method
    maxit=500;
    iter=0;
    tol=1.0e-15;
    es=0.0000001;
    x1=a;f1= Mathd.df(f,x1);
    x2=b;f2= Mathd.df(f,x2);
    x3=(x1+x2)/2.0;f3= Mathd.df(f,x3);
    if f1==0:    return x1;
    elif f2==0:  return x2;
    elif f3==0:  return x3;
    if f1*f2>0: print("No root is existed in the given region");
    p=-(f2*f3*x1*(f2-f3)+f3*f1*x2*(f3-f1)+f1*f2*x3*(f1-f2))/((f1-f2)*(f2-f3)*(f3-f1));
    fp= Mathd.df(f,p);
    ea=abs(x3-p);
    while (ea>es) and (iter<maxit):
      if abs(f3)<tol: return x3;
      if (p<x1) and (p>x2):
        p=(x1+x2)/2.0;
        if p>x3:   x1=x3;f1=f3;x3=p;f3=fp;
        elif p<x3: x2=x3;f2=f3;x3=p;f3=fp;
      else:
        if p>x3: x1=x3;f1=f3;x3=p;f3=fp;
        elif p<x3: x2=x3;f2=f3;x3=p;f3=fp;
        p=-(f2*f3*x1*(f2-f3)+f3*f1*x2*(f3-f1)+f1*f2*x3*(f1-f2))/((f1-f2)*(f2-f3)*(f3-f1));
        fp= Mathd.df(f,p);
        ea=abs(x3-p);
      iter=iter+1;
    if iter>=maxit: print("Warning : Maximum number of iteration is exceeded");
    return p;
# Thiele root finding method
def TM1(f,x):
    #Thiele 1 method
    a=[0.0 for i in range(3)];
    nmax=500;
    tolerance=1.0e-10;
    fx=0;
    for i in range(0,nmax):
      fx=f(x);
      dfx=Mathd.df(f,x);
      d2fx=Mathd.d2f(f,x);
      d3fx=Mathd.dnf(f,x,3);
      P1=4.0*fx*dfx*(3.0*dfx*dfx*d2fx-3.0*fx*d2fx*d2fx+fx*dfx*d3fx);
      Q1=12.0*dfx*dfx*dfx*dfx*d2fx-18.0*fx*dfx*dfx*d2fx*d2fx+3.0*fx*fx*d2fx*d2fx*d2fx+4.0*fx*dfx*dfx*dfx*d3fx;
      x-=P1/Q1;
      if abs(fx)<tolerance: a[0]=x;a[1]=fx;a[2]=i;return a;
    print("Maximum number of iteration is exceeded in Thiele 1\n");
    a[0]=x;a[1]=fx;a[2]=i;
    return a;

def SIGN(a,b):
    return abs(a)*b/abs(b);
#Ridder root finding method
def ridder(f,x1,x2):
    xacc=1.0e-9;
    MAXITER=200;ABC=-1.0e30;
    f_low=f(x1);
    f_high=f(x2);
    b1=f_low > 0.0 and f_high < 0.0;
    b2=f_low < 0.0 and f_high > 0.0;
    b3=b1 or b2;
    if b3 :
      x_low=x1;
      x_high=x2;
      answer=ABC;
      for j in range(0,MAXITER):
        x_mid=0.5*(x_low+x_high);
        f_mid=f(x_mid);
```

```
            s=sqrt(f_mid*f_mid-f_low*f_high);
            if s == 0.0: return answer;
            if f_low>=f_high: xx=1;
            else: xx=-1;
            xnew=x_mid+(x_mid-x_low)*xx*f_mid/s;
            if abs(xnew-answer) <= xacc: return answer;
            answer=xnew;
            fnew=f(answer);
            if fnew == 0.0: return answer;
            if Mathd.SIGN(f_mid,fnew) != f_mid:x_low=x_mid;f_low=f_mid;x_high=answer;f_high=fnew;
            elif Mathd.SIGN(f_low,fnew) != f_low:x_high=answer;f_high=fnew;
            elif Mathd.SIGN(f_high,fnew) != f_high:x_low=answer;f_low=fnew;
            else: print("we should not reach to this point");
        if abs(x_high-x_low) <= xacc: return answer;
        print("Warning : Maximum number of iteration is exceeded \n");
    else:
        if f_low == 0.0: return x1;
        if f_high == 0.0: return x2;
        print("Warning : No roots existed between the given limits ");
    return 0.0;
#Horner synthetic division method for roots of polynomials
def horner(a):
# roots of a polynomial by using synthetic division process
# complex roots
    n=len(a);
    x1=complex(1.56435,0.1423523);
    n1=n;
    nmax=150;
    tolerance=1.0e-10;
    n1=n+1;
    ref=3.1;
    y=[complex(0.0,0.0) for i in range(n1-2)];
    n2=n1-2;
    for k in range(0,n2):
            n=len(a);
            x=x1;
            xold=3.4878;
            ref=abs(xold-x);
            b=[complex(0.0,0) for i in range(n)];
            c=[complex(0.0,0.0) for i in range(n-1)];
            b[n-1]=a[n-1];
            ii=0;
            while ref>tolerance and ii<nmax:
                for i in range(n-2,-1,-1):
                    b[i]=a[i]+b[i+1]*x;
                for i in range(n-2,-1,-1):  c[i]=b[i+1];
                Q=Mathd.f_poly(c,x);
                P=b[0];
                x=x-P/Q;
                ref=abs(xold-x);
                ii=ii+1;
                xold=x;
            a=c;
            n=len(a);
            y[k]=x;
    return y;

def eigen_LeGuerre(A):
    #eigen value result is complex use Horner method with synthetic division
    b=Mathd.polynomial_coefficients(A);
    return Mathd.LeGuerre_SD(b);
#Le Guerre synthetic division method for roots of polynomials
def LeGuerre_SD(a):
# roots of a polynomial by using synthetic division process
# complex roots
    n=len(a);
    x1=complex(1.56435,0.1);
    x=complex(1.56435,0.1);
    n1=n;
    nmax=500;
    tolerance=1.0e-20;
    n1=n+1;
    ref=3.1;
    y=[complex(0.0,0.0) for i in range(n1-2)];
    n2=n1-2;
    for k in range(0,n2):
```

```
        n=len(a);
        m=n-1;
        x=x1;
        xold=3.4878;
        ref=abs(xold-x);
        b=[complex(0.0,0) for i in range(n)];
        c=[complex(0.0,0.0) for i in range(n-1)];
        b[n-1]=a[n-1];
        ii=0;
        while ref>tolerance and ii<nmax:
          for i in range(n-2,-1,-1):
              b[i]=a[i]+b[i+1]*x;
          for i in range(n-2,-1,-1):  c[i]=b[i+1];
          dfx=Mathd.df_poly(a,x);
          fx=Mathd.f_poly(a,x);
          d2fx=Mathd.d2f_poly(a,x);
          G=dfx/fx;
          H=G*G-d2fx/fx;
          y1=sqrt((m*H-G*G)*(m-1.0));
          z1=G+y1;
          z2=G-y1;
          zbool=z1.real*z1.real+z1.imag*z1.imag> z2.real*z2.real+z2.imag*z2.imag;
          if zbool: z=z1;
          else: z=z2;
          aa=n/z;
          x=x-aa;
          ref=abs(xold-x);
          ii=ii+1;
          xold=x;
        if n==2: y[k]=-a[0]/a[1];
        else:    y[k]=x;
        a=c;
        n=len(a);
  return y;

def horner_f(a):
# roots of a polynomial by using synthetic division process
  n=len(a);
  x1=1.56435;
  n1=n;
  nmax=150;
  tolerance=1.0e-10;
  n1=n+1;
  ref=3.1;
  y=[0.0 for i in range(n1-2)];
  n2=n1-2;
  for k in range(0,n2):
        n=len(a);
        x=x1;
        xold=3.4878;
        ref=abs(xold-x);
        b=[0.0 for i in range(n)];
        c=[0.0 for i in range(n-1)];
        b[n-1]=a[n-1];
        ii=0;
        while ref>tolerance and ii<nmax:
          for i in range(n-2,-1,-1):
              b[i]=a[i]+b[i+1]*x;
          for i in range(n-2,-1,-1):  c[i]=b[i+1];
          Q=Mathd.f_poly(c,x);
          P=b[0];
          x=x-P/Q;
          ref=abs(xold-x);
          ii=ii+1;
          xold=x;
        a=c;
        n=len(a);
        y[k]=x;
  return y;

def trace(B):
    sum=0;
    n=len(B);
    for i in range(0,n):sum+=B[i][i];
    return sum;
```

```
def topla(A,a):
    n=len(A);
    m=len(A[0]);
    B = [[0 for x in range(m)] for y in range(n)];
    for i in range(0,n):
       for j in range(0,m):
         if i!=j: B[i][j]=A[i][j];
         else:     B[i][j]=A[i][j]+a;
    return B;

def AT(A):
    #transpose matrix
    n=len(A);
    m=len(A[0]);
    B = [[0 for x in range(m)] for y in range(n)];
    for i in range(0,n):
       for j in range(0,m):
         B[i][j]=A[j][i];
    return B;

def I(n):
#unit matrix of dimension n
    B = [[0 for x in range(n)] for y in range(n)];
    for i in range(0,n):
       for j in range(0,n):
         if i==j: B[i][j]=1.0;
    return B;

def invA(A):
#inverse matrix (n=m)
    n=len(A);
    I= Mathd.I(n);
    B = [[0 for x in range(n)] for y in range(n)];
    for i in range(n):
        B[i]=Mathd.gauss(A,I[i]);
    return B;

def invB(B):
    #inverse matrix (n!=m)
    x=Mathd.SVD(B);
    S=x[0];
    V=x[1];
    U=x[2];
    n=len(S);
    SI = [[0.0 for x in range(n)] for y in range(n)];
    for i in range(0,n):
      for j in range(0,n):
          if i==j: SI[i][j]=1.0/S[i][j];
    UT=Mathd.AT(U);
    return Mathd.carp_m_m(V,Mathd.carp_m_m(SI,UT));

def float_vect(A):
    #converts complex to float ignore complex part
    n=len(A);
    B=[0.0 for x in range(n)];
    for i in range(0,n):
        B[i]=A[i].real;
    return B;

def carp_c_m(left,right):
    #multiplying a matrix with a constant
    n=len(right);
    m=len(right[0]);
    b = [[0 for x in range(m)] for y in range(n)];
    for i in range(0,n):
       for j in range(0,m):
          b[i][j]=right[i][j]*left;
    return b;
    #end of multiplying a matrix with a constant double

def carp_m_m(left,right):
    #multiplying two matrisis
    n1=len(left);
    m1=len(left[0]);
    n2=len(right);
    m2=len(right[0]);
```

```
    b = [[0 for x in range(n1)] for y in range(m2)];
    if m1 != n2:
      print("warning : matrix sizes should be same");
      return b;
    for i in range(0,n1):
      for j in range(0,m2):
        for k in range(0,m1):
            b[i][j]+=left[i][k]*right[k][j];
    return b;

def add_m_m(left,right):
    #adding two matrisis
    n1=len(left);
    m1=len(left[0]);
    n2=len(right);
    m2=len(right[0]);
    b = [[0 for x in range(n1)] for y in range(m2)];
    if m1 != n2:
      print("warning : matrix sizes should be same");
      return b;
    for i in range(0,n1):
      for j in range(0,m2):
            b[i][j]=left[i][j]+right[i][j];
    return b;

def substract_m_m(left,right):
    #substracting two matrisis
    n1=len(left);
    m1=len(left[0]);
    n2=len(right);
    m2=len(right[0]);
    b = [[0 for x in range(n1)] for y in range(m2)];
    if m1 != n2:
      print("warning : matrix sizes should be same");
      return b;
    for i in range(0,n1):
      for j in range(0,m2):
            b[i][j]=left[i][j]-right[i][j];
    return b;

def mult_m_m(left,right):
    return Mathd.carp_m_m(left,right);

def polynomial_coefficients(A):
    n=len(A);
    m=len(A[0]);
    B = [[0 for x in range(m)] for y in range(n)];
    for i in range(0,n):
      for j in range(0,m):
        B[i][j]=A[i][j];
    a = [0.0 for y in range(n+1)];
    C = [[0.0 for x in range(m)] for y in range(n)];
    a[n]=1.0;
    a[n-1]=-Mathd.trace(B);
    for k in range(n-2,-1,-1):
       C=Mathd.topla(B,a[k+1]);
       B=Mathd.carp_m_m(A,C);
       a[k]=-Mathd.trace(B)/(n-k);
    return a;
#Eigen values by using Horners sysnthetic division method
def eigen_horner(A):
   #eigen value result is complex use Horner method with synthetic division
   b=Mathd.polynomial_coefficients(A);
   return Mathd.horner(b);

def eigen_horner_f(A):
   #eigen value result is float use Horner method with synthetic division
   b=Mathd.polynomial_coefficients(A);
   return Mathd.horner_f(b);
#Singular value decomposition
def SVD(A):
   #singular value decomposition
   n=len(A)
   B=Mathd.AT(A);
   C=Mathd.mult_m_m(B,A);
   D=Mathd.eigen_jacobi(C);
```

```
    V = [[0.0 for x in range(n)] for y in range(n)];
    E=D[0];
    for i in range(0,n):
        for j in range(0,n):
            V[i][j]=D[i+1][j];
    E=sorted(E, reverse=True); #eigen values
    S = [[0.0 for x in range(n)] for y in range(n)];
    #inverse of S
    SI = [[0.0 for x in range(n)] for y in range(n)];
    for i in range(0,n):
        for j in range(0,n):
            if i==j: S[i][j]=sqrt(E[i]);SI[i][j]=1.0/S[i][j];
    for i in range(0,n):
        sum=0.0;
        for j in range(0,n):
            sum=sum+V[i][j]*V[i][j];
        sum=sqrt(sum);
        for j in range(0,n):
           V[i][j]= V[i][j]/sum;
    VT=Mathd.AT(V);
    U=Mathd.mult_m_m(A,V);
    U=Mathd.mult_m_m(U,SI);
    UT=Mathd.AT(U);
    AA=Mathd.mult_m_m(U,Mathd.mult_m_m(S,VT));
    x=[S,V,U];
    return x;
  #Eigenvalue by using Jacobi method (Symmetric matrices only)
  def eigen_jacobi(A):
    # eigen value of a symmetric matrix
    # Jacobi method
    eps=1e-15;
    err=1;
    m=100;
    n=len(A);
    P= [[0.0 for x in range(n)] for y in range(n)];
    R= [[0.0 for x in range(n)] for y in range(n)];
    p=n;q=n;
    lambda1=[0.0 for x in range(n)];
    max=-9.99e100;
    P=Mathd.I(n);
    for i in range(0,n):
         for j in range(i+1,n):
             if abs(A[i][j]) > max: max=abs(A[i][j]);p=i;q=j;
    while abs(A[p][q]) > eps:
        fi=0.5*atan(2.0*A[p][q]/(A[p][p]-A[q][q]+1e-40));
        R=Mathd.I(n);
        R[p][p]=cos(fi);
        R[q][q]=cos(fi);
        R[p][q]=-sin(fi);
        R[q][p]=sin(fi);
        P=Mathd.carp_m_m(P,R);
        A=Mathd.carp_m_m(Mathd.AT(R),Mathd.carp_m_m(A,R));
        max=-9.99e100;
        for i in range(0,n):
            for j in range(i+1,n):
                if abs(A[i][j]) > max: max=abs(A[i][j]);p=i;q=j;
    for i in range(0,n):lambda1[i]=A[i][i];
    PP= [[0.0 for x in range(n)] for y in range(n+1)];
    PP[0]=lambda1;
    for i in range(1,n+1):
       for j in range(0,n):
          PP[i][j]=P[i-1][j];
    return PP;
 #Gauss random number generator
  def  gaussrandom(xort,r):
     iset=0;
     gset=0;
     aret=0;
     v1=2.0*random()-1.0;
     v2=2.0*random()-1.0;
     rsq=v1*v1+v2*v2;
     while rsq>=1.0 and rsq==0.0:
      if iset==0:
         v1=2.0*random()-1.0;
         v2=2.0*random()-1.0;
         rsq=v1*v1+v2*v2;
```

```
            fac=sqrt(-2.0*log(rsq)/rsq);
            gset=v1*fac;
            iset=1;
            aret=v2*fac;
        else:
            iset=0;
            aret=gset;
       return aret/8.0*r+xort;
   def gamma(z):return exp(lngamma(z)[0])

# natural logarithm of gamma function
   def lngamma(x):
    #    logarithm of gamma function
    #    xx[0] value of logarithm of gamma function
    #   xx[1] sign of gamma functon
        xx=[0 for i in range(2)]
        # result
        # a
        # b
        # c
        # p
        # q
        # u
        # w
        # z
        # int i
        # logpi
        #ls2pi;
        #tmp;
        sgngam = 1;
        logpi = 1.14472988584940017414;
        ls2pi = 0.91893853320467274178;
        if x<-34.0:
            q = -x;
            #double yy[]=lngamma(q);
            yy=lngamma(q);
            w = yy[0]
            tmp=yy[1]
            p = floor(q);
            i = int(round(p,0))
            if i%2==0:
                sgngam = -1
            else:
                sgngam = 1
            z = q-p;
            if  z>0.5 :
                p = p+1;
                z = p-q;
            z = q*sin(PI*z);
            result = logpi-log(z)-w;
            xx[0]=result
            xx[1]=sgngam
            return xx
        elif x<13 :
            z = 1;
            p = 0;
            u = x;
            while u>=3:
                p = p-1;
                u = x+p;
                z = z*u;
            while u<2:
                z = z/u;
                p = p+1;
                u = x+p;
            if z<0:
                sgngam = -1
                z = -z
            else:
                sgngam = 1
            if u==2:
                result = log(z);
                xx[0]=result;
                xx[1]=sgngam;
                return xx;
            p = p-2;
```

```
            x = x+p;
            b = -1378.25152569120859100;
            b = -38801.6315134637840924+x*b;
            b = -331612.992738871184744+x*b;
            b = -1162370.97492762307383+x*b;
            b = -1721737.00820839662146+x*b;
            b = -853555.664245765465627+x*b;
            c = 1;
            c = -351.81570143652347054 9+x*c;
            c = -17064.2106651881159223+x*c;
            c = -220528.590553854454839+x*c;
            c = -1139334.44367982507207+x*c;
            c = -2532523.07177582951285+x*c;
            c = -2018891.41433532773231+x*c;
            p = x*b/c;
            result = log(z)+p;
            xx[0]=result;
            xx[1]=sgngam;
            return xx;
        q = (x-0.5)*log(x)-x+ls2pi;
        if x>100000000:
            result = q
            xx[0]=result
            xx[1]=sgngam
            return xx
        p = 1/(x*x)
        if x>=1000.0 :
            q = q+((7.93650793650793650793 65*0.0001*p-2.7777777777777777777778*0.001)*p+0.08333333333333333333333)/x
        else:
            a = 8.11614167470508450300*0.0001
            a = -5.95061904284301438324*0.0001+p*a
            a = 7.93650340457716943945*0.0001+p*a
            a = -2.77777777730099687205*0.001+p*a
            a = 8.33333333333331927722*0.01+p*a
            q = q+a/x
        xx[0]= q
        xx[1]=sgngam
        return xx
#Beta function
  def beta(z,w):
        return exp(lngamma(z)[0]+lngamma(w)[0]-lngamma(z+w)[0])
#incomplete gamma function
  def gammap(a,x):
        if x<a+1.0:
            gamser=gser(a,x)
            return gamser
        elif x<0.0 or a<=0.0:print("Invalid arguments in routine GAMMP")
        else:
            gammcf=gcf(a,x)
            return 1.0-gammcf
  def gammaq(a,x):
        if x < 0.0 or a <= 0.0:
            print("Invalid arguments in routine GAMMQ")
        if x < (a+1.0):
            gamser=gser(a,x)
            return 1.0-gamser
        else:
            gammcf=gcf(a,x)
            return gammcf

  def gser(a,x):
        AX=200
        EPS=3.0e-7
        ap=0.0
        gln=lngamma(a)[0]
        ser=0
        if x<=0.0:
            if x<0.0:
                print("x less than 0 in routine GSER")
                gamser=0.0
                return gamser
        else:
            ap=a
            del1=1.0/a
            sum=1.0/a
            for n in range(1,ITMAX+1):
```

```
            ap =ap+1.0
            del1 =del1*x/ap
            sum =sum+del1
            xx1=abs(del1)
            xx2=abs(sum)*EPS
            if xx1<xx2:
                gamser=sum*exp(-x+a*log(x)-gln)
                return gamser
    return gamser
def gcf(a,x):
    ITMAX=200
    EPS=3.0e-7
    gammcf=0
    gold=0.0
    fac=1.0
    b1=1.0
    b0=0.0
    a0=1.0
    g=0
    gln=lngamma(a)[0]
    a1=x
    for n in range(1,ITMAX+1):
        an=float(n)
        ana=an-a
        a0=(a1+a0*ana)*fac
        b0=(b1+b0*ana)*fac
        anf=an*fac
        a1=x*a0+anf*a1
        b1=x*b0+anf*b1
        xx1=(g-gold)/g
        xx1=abs(xx1)
        if a1!=0:
            fac=1.0/a1
            g=b1*fac
            if xx1<EPS:
                gammcf=exp(-x+a*log(x)-gln)*g
                return gammcf
        gold=g
    return gammcf
 # Error function
def erf(x):
    if x < 0.0:
        y=-gammp(0.5,x*x)
    else:
        y=gammp(0.5,x*x)
    return y
 #complimentary error function
def erf(x):
     return 1.0-Mathd.erf(x)

 # Bessel polynomial J
def J(v,x):
    if x==0.0: x=1.0e-10
    JJ=0.0
    a=1.0
    fact=1
    b=0.0
    x1=0.0
    if x <=20:
        x5=1.0
        lna=0.0;
        x1=log(0.5*x)
        x1=x1*v
        lna1=log(0.25*x*x)
        for k in range(0,100):
            x2=lna
            x3=Mathd.lngamma(k+1)[0]
            x4=Mathd.lngamma(v+k+1)[0]
            b=x1+x2-x3-x4
            b=x5*exp(b)
            lna=lna+lna1
            x5=-1*x5
            JJ=JJ+b
    elif (v==0 and x==0):
        JJ=1.0
    else:
```

```
        JJ=sqrt(2.0/PI/x)*cos(x-0.5*v*PI-0.25*PI)
      return JJ
#Bessel I
  def I(v,x):
      if x==0.0: x=1.0e-10
      JJ=0;a=1.0;fact=1
      vtest=int((v*10)/10)
      x5=1
      lna=0.0
      x1=v*log(0.5*x)
      lna1=log(0.25*x*x)
      for k in range(100):
         x2=lna
         x3=Mathd.lngamma(k+1)[0]
         x4=Mathd.lngamma(v+k+1)[0];
         b=x1+x2-x3-x4;
         b=exp(b);
         lna+=lna1;
         JJ+=b
      if v==0 and x==0: JJ=1
      return JJ

#Bessel polynomial Y
  def Y(v,x):
     if x==0.0: x=1.0e-10
     YY=(J(v,x)*cos(v*PI)-J(-v,x))/sin(v*PI)
     return YY
  def K(v,x):
     if x==0.0:
        x=1.0e-10
     YY=PI/2.0*(I(-v,x)-I(v,x))/sin(v*Math.PI)
     return YY
   #natural logarithm of factorial
  def factln(n):
      #log(fact) of an integer number

a=[0.0,0.0,0.6931471805599453,1.0986122886681098,1.3862943611198906,1.6094379124341003,1.791759469228055,1.94591014905
53132,2.0794415416798357,2.1972245773362196,2.302585092994046,2.3978952727983707,2.4849066497880004,2.56494935746153
67,2.6390573296152584,2.70805020110221,2.772588722239781,2.833213344056216,2.8903717578961645,2.9444389791664403,2.99
5732273553991,3.044522437723423,3.091042453358316,3.1354942159291497,3.1780538303479458,3.2188758248682006,3.2580965
38021482,3.295836866004329,3.332204510175204,3.367295829986474,3.4011973816621555,3.4339872044851463,3.4657359027997
265,3.4965075614664802,3.5263605246161616,3.5553480614894135,3.58351893845611,3.6109179126442243,3.6375861597263857,3
.6635616461296463,3.6888794541139363,3.713572066704308,3.7376696182833684,3.7612001156935624,3.784189633918261,3.8066
624897703196,3.828641396489095,3.8501476017100584,3.871201010907891,3.8918202981106265,3.912023005428146,3.931825632
7243257,3.9512437185814275,3.970291913552122,3.9889840465642745,4.007333185232471,4.02535169073515,4.04305126783455,4
.060443010546419,4.07753744390572,4.0943445622221,4.110873864173311,4.127134385045092,4.143134726391533,4.15888308335
96715,4.174387269895637,4.189654742026425,4.204692619390966,4.219507705176107,4.23410650459726,4.248495242049359,4.26
26798770413155,4.276666119016055,4.290459441148391,4.30406509320417,4.31748811353631,4.330733340286331,4.34380542185
3684,4.356708826689592,4.3694478524670215,4.382026634673881,4.394449154672439,4.406719247264253,4.418840607796598,4.4
30816798843313,4.442651256490317,4.454347296253507,4.465908118654584,4.477336814478207,4.48863636973214,4.4998096703
30265,4.51085950651685,4.5217885770490405,4.532599493153256,4.543294782270004,4.553876891600541,4.564348191467836,4.5
74710978503383,4.584967478670572,4.59511985013459,4.605170185988092]
      if n < 0:
         print("Negative fact in routine FACTLN")
         return 0.0
      elif n > 100:
         b=lngamma(n+1.0)
         return b[0]
      else:
         return a[n]
   #factorial
  def fact(n):
      return exp(factln(n))
   #Legendre polynomial
  def P(n,x):
      if n==0: return 1.0
      elif n==1: return x
      else:return (P((n-1),x)*x*float(2*n-1)-P((n-2),x)*float(n-1))/float(n)
   #Derivative of Legendre Polynomial
  def dP(n,x):
      return float(n)/(x*x-1.0)*(x*P(n,x)-P((n-1),x))
   #Chebychev polynomials
  def T(n,x):
      if n==0: return 1.0
      elif n==1: return x
      else:return (2.0*T((n-1),x)*x-T((n-2),x))
```

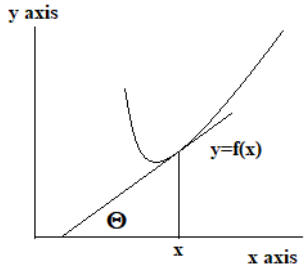
3. DERIVATIVES

3.1 DEFINITIONS

Definition of derivatives:

$$\frac{df(x)}{dx} = lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x}$$

Other definitions



Derivative is the slope of tangent to the function at a given x value

$$\frac{df(x)}{dx} = \tan(\theta)$$

Let us apply basic definition to some functions:

$$f(x) = x^2$$

$$\frac{df(x)}{dx} = lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x} = lim_{\Delta x \to 0} \frac{(x+\Delta x)^2 - x^2}{\Delta x} = lim_{\Delta x \to 0} \frac{x^2 + 2x\Delta x + \Delta x^2 - x^2}{\Delta x}$$

$$\frac{df(x)}{dx} = lim_{\Delta x \to 0} \frac{\Delta x(2x + \Delta x)}{\Delta x} = lim_{\Delta x \to 0}(2x + \Delta x) = \mathbf{2x}$$

$$f(x) = x^3$$

$$\frac{df(x)}{dx} = lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x} = lim_{\Delta x \to 0} \frac{(x+\Delta x)^3 - x^3}{\Delta x}$$

$$\frac{df(x)}{dx} = lim_{\Delta x \to 0} \frac{x^3 + 3x^2\Delta x + 3x\Delta x^2 + \Delta x^3 - x^3}{\Delta x} = lim_{\Delta x \to 0} \frac{3x^2\Delta x + 3x\Delta x^2 + \Delta x^3}{\Delta x}$$

$$\frac{df(x)}{dx} = lim_{\Delta x \to 0} \frac{\Delta x(3x^2 + 3x\Delta x + \Delta x^2)}{\Delta x} = lim_{\Delta x \to 0}(3x^2 + 3x\Delta x + \Delta x^2) = \mathbf{3x^2}$$

Some common analytic formulas for derivatives:

f(x)	$f'(x) = \frac{df(x)}{dx}$
x^n	nx^{n-1}
x^2	$2x$
x^3	$3x^2$
e^{ax}	ae^{ax}
$e^{u(x)}$	$u'(x)e^u$
e^{-x}	$-e^{-x}$
a^{bx}	$bln(a)a^{bx}$
a^x	$ln(a)a^x$

2^{3x}	$3ln(2)2^{3x}$
$ln(u(x))$	$\frac{u'(x)}{u(x)}$
cos(u(x))	-u'(x)sin(u(x))
sin(u(x))	u'(x)cos(u(x))
tan(u(x))	u'(x)$sec^2(u(x))$
$\arcsin(u(x))$	$\frac{u'(x)}{\sqrt{1-u(x)^2}}$
$\arccos(u(x))$	$-\frac{u'(x)}{\sqrt{1-u(x)^2}}$
$\arctan(u(x))$	$\frac{u'(x)}{1+u(x)^2}$
cosh(u(x))	u'(x)sinh(u(x))
sinh(u(x))	u'(x)cosh(u(x))
tanh(u(x))	u'(x)$sech^2(u(x))$
$\text{arcsinh}(u(x))$	$\frac{u'(x)}{\sqrt{1+u(x)^2}}$
$\text{arccosh}(u(x))$	$-\frac{u'(x)}{\sqrt{u(x)^2-1}}$
$\text{arctanh}(u(x))$	$\frac{u'(x)}{1-u(x)^2}$
$u(x) \pm v(x)$	$u'(x) \pm v'(x)$
$u(x)v(x)$	$u'(x)v(x) + v'(x)u(x)$
$\frac{1}{u(x)}$	$-\frac{u'(x)}{u^2(x)}$
$\frac{u(x)}{v(x)}$	$\frac{u'(x)v(x) - v'(x)u(x)}{v^2(x)}$

Examples of analytical derivations

EX1: take derivative of $y = f(x) = 2x^3 + 3x^2 + 5$

Answer: $f'(x) = 6x^2 + 6x$

EX2: take derivative of $y = f(x) = u(x)v(x) = (2x^3 + 3x^2 + 5)(x^2 - 2)$

$f'(x) = u'(x)v(x) + v'(x)u(x)$

$u'(x) = 6x^2 + 6x \qquad v'^{(x)} = 2x$

$f'(x) = (6x^2 + 6x)(x^2 - 2) + (2x^3 + 3x^2 + 5)(2x)$

EX3: take derivative of $y = f(x) = \frac{u(x)}{v(x)} = \frac{(2x^3+3x^2+5)}{(x^2-2)}$

$$f'(x) = \frac{u'(x)v(x) - v'(x)u(x)}{v^2(x)}$$

$$f'(x) = \frac{(6x^2 + 6x)(x^2 - 2) - (2x^3 + 3x^2 + 5)(2x)}{(x^2 - 2)^2}$$

EX4: Use the chain rule to calculate $\frac{dy}{dx}$

$y(x) = u(x)^2 - 2u(x) + 1$ where $u(x) = (x^2 + 4)^2$

$\frac{dy(x)}{dx} = \frac{dy(x)}{du(x)}\frac{du(x)}{dx} = [2u(x) - 2][2(x^2 + 4)2x]$=$[2(x^2 + 4)^2 - 2][2(x^2 + 4)2x]$

EX5: Use the chain rule to calculate $\frac{dy}{dx}$

$y(x) = 2u(x)^3 \quad u(v) = \sqrt{v^2 + 3} \qquad v(x) = \frac{1}{x}$

$\frac{dy(x)}{du(x)} = 6u(x)^2 \qquad \frac{du(v)}{dv} = (v^2 + 3)^{1/2} = \frac{1}{2}\frac{2v}{\sqrt{v^2+3}} \qquad \frac{dv(x)}{dx} = -\frac{1}{x^2}$

$$\frac{dy(x)}{dx} = \frac{-6u(x)^2\, v(x)}{x^2\sqrt{v^2 + 3}}$$

EX6: Calculate the derivative of $y(x) = x^{(x^2+2x+1)}$

$$\ln(y(x)) = (x^2 + 2x + 1)\ln(x)$$

$$\frac{\frac{dy(x)}{dx}}{y(x)} = (2x+2)\ln(x) + \frac{(x^2+2x+1)}{x}$$

$$\frac{dy(x)}{dx} = y(x)\left[(2x+2)\ln(x) + \frac{(x^2+2x+1)}{x}\right]$$

$$\frac{dy(x)}{dx} = x^{(x^2+2x+1)}\left[(2x+2)\ln(x) + \frac{(x^2+2x+1)}{x}\right]$$

NUMERICAL DERIVATIONS (DIFFERENCE FORMULAS)

In order to approximate derivation difference equations can be used.Difference equations can basically be derived from Taylor formulations

$$f(x_{i+1}) = \sum_{i=0}^{\infty} \frac{f^{(i)}(x_i)}{i!} h^i$$

$$f(x_{i+1}) = f(x_i) + f'(x_i)h + \frac{f''(x_i)}{2}h^2 + \frac{f^{(3)}(x_i)}{3!}h^3 + \frac{f^{(4)}(x_i)}{4!}h^4 + \cdots$$

If the first derivative is solved from this equation it is found as

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{h} - \frac{f''^{(x_i)}}{2}h - \frac{f^{(3)}(x_i)}{3!}h^2 - \frac{f^{(4)}(x_i)}{4!}h^3 + O(h^4)$$

Or

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{h} + O(h)$$

Term O(h) is the order of error. Error is proportional to the difference parameter h. When h is getting smaller, error will shrinked accordingly. If the second derivative is written the same way:

$$f''(x_i) = \frac{f(x_{i+2}) - 2f(x_{i+1}) + f(x_i)}{h^2} + O(h)$$

In order to decrease the error, if the second derivative difference equation is substituted into the first derivative Taylor equation a first derivative equation with a second order error can be obtained.

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{h} - \frac{f''^{(x_i)}}{2}h + O(h^2)$$

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{h} - \frac{1}{2}\frac{f(x_{i+2}) - 2f(x_{i+1}) + f(x_i)}{h^2}h + O(h^2)$$

$$f'(x_i) = \frac{f(x_{i+2}) - 4f(x_{i+1}) - 3f(x_i)}{2h} + O(h^2)$$

Difference equation can be written as forward or backward way. The backward difference equation:

$$f'(x_i) = \frac{f(x_i) - f(x_{i-1})}{h} + O(h)$$

. By taken average of forward and bakward formulas can create a central formula

$$f'(x_i) = \frac{f'_{forward} + f'_{backward}}{2} = \frac{\frac{f(x_i) - f(x_{i-1})}{h} + \frac{f(x_{i+1}) - f(x_i)}{h}}{2} + O(h^2)$$

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_{i-1})}{2h} + +O(h^2)$$

The coefficients of difference formulas for calculating derivatives are given in the following tables

Derivative (Difference) equations

Central finite difference

Derivativ	Accurac	−5	−4	−3	−2	−1	0	1	2	3	4	5

1	2					−1/2	0	1/2				
	4				1/12	−2/3	0	2/3	−1/12			
	6			−1/60	3/20	−3/4	0	3/4	−3/20	1/60		
	8		1/280	−4/10	1/5	−4/5	0	4/5	−1/5	4/105	−1/280	
2	2					1	−2	1				
	4				−1/12	4/3	−5/2	4/3	−1/12			
	6			1/90	−3/20	3/2	−49/18	3/2	−3/20	1/90		
	8		−1/560	8/315	−1/5	8/5	−205/72	8/5	−1/5	8/315	−1/560	
3	2				−1/2	1	0	−1	1/2			
	4			1/8	−1	13/8	0	−13/8	1	−1/8		
	6		−7/240	3/10	−169/12	61/30	0	−61/30	169/120	−3/10	7/240	
4	2				1	−4	6	−4	1			
	4			−1/6	2	−13/2	28/3	−13/2	2	−1/6		
	6		7/240	−2/5	169/60	−122/1	91/8	−122/1	169/60	−2/5	7/240	
5	2			−1/2	2	−5/2	0	5/2	−2	1/2		
	4		1/6	−3/2	13/3	−29/6	0	29/6	−13/3	3/2	−1/6	
	6	−13/28	19/36	−87/3	13/2	−323/4	0	323/48	−13/2	87/32	−19/36	13/28
6	2			1	−6	15	−20	15	−6	1		
	4		−1/4	3	−13	29	−75/2	29	−13	3	−1/4	
	6	13/240	−19/24	87/16	−39/2	323/8	−1023/2	323/8	−39/2	87/16	−19/24	13/24

Forward finite difference

Derivative	Accuracy	0	1	2	3	4	5	6	7	8
1	1	−1	1							
	2	−3/2	2	−1/2						
	3	−11/6	3	−3/2	1/3					
	4	−25/12	4	−3	4/3	−1/4				
	5	−137/60	5	−5	10/3	−5/4	1/5			
	6	−49/20	6	−15/2	20/3	−15/4	6/5	−1/6		

2	1	1	−2	1						
	2	2	−5	4	−1					
	3	35/12	−26/3	19/2	−14/3	11/12				
	4	15/4	−77/6	107/6	−13	61/12	−5/6			
	5	203/45	−87/5	117/4	−254/9	33/2	−27/5	137/180		
	6	469/90	−223/10	879/20	−949/18	41	−201/10	1019/180	−7/10	
3	1	−1	3	−3	1					
	2	−5/2	9	−12	7	−3/2				
	3	−17/4	71/4	−59/2	49/2	−41/4	7/4			
	4	−49/8	29	−461/8	62	−307/8	13	−15/8		
	5	−967/120	638/15	−3929/40	389/3	−2545/24	268/5	−1849/120	29/15	
	6	−801/80	349/6	−18353/120	2391/10	−1457/6	4891/30	−561/8	527/30	−469/240
4	1	1	−4	6	−4	1				
	2	3	−14	26	−24	11	−2			
	3	35/6	−31	137/2	−242/3	107/2	−19	17/6		
	4	28/3	−111/2	142	−1219/6	176	−185/2	82/3	−7/2	
	5	1069/80	−1316/15	15289/60	−2144/5	10993/24	−4772/15	2803/20	−536/15	967/240

Backward finite difference

Derivative	Accuracy	0	-1	-2	-3	-4	-5	-6	-7	-8
1	1	1	-1							
	2	3/2	-2	1/2						
	3	11/6	-3	3/2	-1/3					
	4	25/12	-4	3	-4/3	1/4				
	5	137/60	-5	5	-10/3	5/4	-1/5			
	6	49/20	-6	15/2	-20/3	15/4	-6/5	1/6		
2	1	1	−2	1						
	2	2	−5	4	−1					
	3	35/12	−26/3	19/2	−14/3	11/12				

	4	15/4	−77/6	107/6	−13	61/12	−5/6			
	5	203/45	−87/5	117/4	−254/9	33/2	−27/5	137/180		
	6	469/90	−223/10	879/20	−949/18	41	−201/10	1019/180	−7/10	
3	1	1	-3	3	1					
	2	5/2	-9	12	-7	3/2				
	3	17/4	-71/4	59/2	-49/2	41/4	-7/4			
	4	49/8	-29	461/8	-62	307/8	-13	15/8		
	5	967/120	-638/15	3929/40	-389/3	2545/24	-268/5	1849/120	-29/15	
	6	801/80	-349/6	18353/120	-2391/10	1457/6	-4891/30	561/8	-527/30	469/240
4	1	1	−4	6	−4	1				
	2	3	−14	26	−24	11	−2			
	3	35/6	−31	137/2	−242/3	107/2	−19	17/6		
	4	28/3	−111/2	142	−1219/6	176	−185/2	82/3	−7/2	
	5	1069/80	−1316/15	15289/60	−2144/5	10993/24	−4772/15	2803/20	−536/15	967/240

In order to Show utilisation of the table, the derivative of f(x)=sin(x)

y=f(x)=sin(x)

Derivative	Accuracy	-5	-4	-3	-2	-1	0	1	2	3	4	5
1	2	0.00000000	0.00000000	0.00000000	0.00000000	-0.50000000	0.00000000	0.50000000	0.00000000	0.00000000	0.00000000	0.00000000
	4	0.00000000	0.00000000	0.00000000	0.08333333	-0.66666667	0.00000000	0.66666667	-0.08333333	0.00000000	0.00000000	0.00000000
	6	0.00000000	0.00000000	-0.01666667	0.15000000	-0.75000000	0.00000000	0.75000000	-0.15000000	0.01666667	0.00000000	0.00000000
	8	0.00000000	0.00357143	-0.03809524	0.20000000	-0.80000000	0.00000000	0.80000000	-0.20000000	0.03809524	-0.00357143	0.00000000

Δx	0.01	-5	-4	-3	-2	-1	0	1	2	3	4	5		
x	0	-0.05	-0.04	-0.03	-0.02	-0.01	0	0.01	0.02	0.03	0.04	0.05		
y=f(x)	Accuracyy	-0.049979	-0.039989	-0.029996	-0.019998667	-0.01	0	0.009999833	0.0199987	0.0299955	0.0399893	0.049979	Σ	f'(x)
	2	0	0	0	0	0.0049999	0	0.004999917	0	0	0	0	0.01	0.999983
	4	0	0	0	-0.001666556	0.0066666	0	0.006666556	-0.001667	0	0	0	0.01	1
	6	0	0	0.0004999	-0.0029998	0.0074999	0	0.007499875	-0.003	0.000499925	0	0	0.01	1
	8	0	-0.000143	0.0011427	-0.003999733	0.0079999	0	0.007999867	-0.004	0.001142686	-0.000143	0	0.01	1

Derivative	Accuracy	-5	-4	-3	-2	-1	0	1	2	3	4	5
	2					1	-2	1				
	4				-0.083333333	1.3333333	-2.5	1.333333333	-0.083333			
2	6			0.0111111	-0.15	1.5	-2.722222	1.5	-0.15	0.011111111		
	8		-0.001786	0.0253968	-0.2	1.6	-2.847222	1.6	-0.2	0.025396825	-0.001786	

Δx	0.01	-5	-4	-3	-2	-1	0	1	2	3	4	5		
x	0	-0.05	-0.04	-0.03	-0.02	-0.01	0	0.01	0.02	0.03	0.04	0.05		
y=f(x)	Accuracyy	-0.049979	-0.039989	-0.029996	-0.019998667	-0.01	0	0.009999833	0.0199987	0.0299955	0.0399893	0.049979	Σ	f''(x)
	2	0	0	0	0	-0.01	0	0.009999833	0	0	0	0	0	0
	4	0	0	0	0.001666556	-0.013333	0	0.013333111	-0.001667	0	0	0	-2.2E-19	-2.2E-17
	6	0	0	-0.000333	0.0029998	-0.015	0	0.01499975	-0.003	0.000333283	0	0	2.71E-19	2.71E-17
	8	0	7.141E-05	-0.000762	0.003999733	-0.016	0	0.015999733	-0.004	0.00076179	-7.14E-05	0	2.17E-19	2.17E-17

Derivative	Accuracy	-5	-4	-3	-2	-1	0	1	2	3	4	5
	2	0	0	0	-0.5	1	0	-1	0.5	0	0	0
3	4	0	0	0.125	-1	1.625	0	-1.625	1	-0.125	0	0
	6	0	-0.029167	0.30000000	-1.40833333	2.03333333	0.00000000	-2.03333333	1.40833333	-0.30000000	0.02916667	0

Δx	0.0001	-5	-4	-3	-2	-1	0	1	2	3	4	5		Third
x	0	-0.05	-0.04	-0.03	-0.02	-0.01	0	0.01	0.02	0.03	0.04	0.05		Derivative
y=f(x)	Accuracyy	-0.049979	-0.039989	-0.0299955	-0.019998667	-0.0099998	0	0.009999833	0.01999867	0.0299955	0.03998933	0.049979	Σ	f'''(x)
	2	0	0	0	0.009999333	-0.0099998	0	-0.009999833	0.00999933	0	0	0	-1E-06	-0.999975
	4	0	0	-0.0037494	0.019998667	-0.0162497	0	-0.016249729	0.01999867	-0.003749438	0	0	-1E-06	-1
	6	0	0.0011664	-0.0089987	0.028164789	-0.020333	0	-0.020332994	0.02816479	-0.00899865	0.00116636	0	-1E-06	-1

First derivatives:

(central difference Formula)

$$f'(x_n) \cong \frac{f(x_n+\Delta x)-f(x_n-\Delta x)}{2\Delta x}$$

(Forward difference formula-same as in derivative definition except the limit)

$$f'(x_n) \cong \frac{f(x_n+\Delta x)-f(x_n)}{\Delta x}$$

(Backward difference formula)

$$f'(x_n) \cong \frac{f(x_n)-f(x_n-\Delta x)}{\Delta x}$$

In order to minimize error Δx should be relatively small, to have a further approximation a second or higher order derivative (difference) formulas can also be used

$$f'(x_n) \cong \frac{-f(x_n+2\Delta x)+8f(x_n+\Delta x)-8f(x_n-\Delta x)+f(x_n-2\Delta x)}{12\Delta x} \quad (2.5.5)$$

$$f'(x_n) \cong \frac{f(x_n+3\Delta x)-9f(x_n+2\Delta x)+45f(x_n+\Delta x)-45f(x_n-\Delta x)+9f(x_n-2\Delta x)-f(x_n-3\Delta x)}{60\Delta x} \quad (2.5.6)$$

$$f'(x_n) \cong \frac{-3f(x_n+4\Delta x)+32f(x_n+3\Delta x)-168f(x_n+2\Delta x)+672f(x_n+\Delta x)-672f(x_n-\Delta x)+168f(x_n-2\Delta x)-32f(x_n-3\Delta x)+3f(x_n-4\Delta x)}{840\Delta x}$$

Second derivatives:

$$f''(x_n) \cong \frac{f(x_n+\Delta x)-2f(x_n)+f(x_n-\Delta x)}{\Delta x^2}$$

$$f''(x_n) \cong \frac{-f(x_n+2\Delta x)+16f(x_n+\Delta x)-30f(x_n)+16f(x_n-\Delta x)-f(x_n-2\Delta x)}{12\Delta x^2}$$

```
def f(x):
    y=x*x-2.0*x+5.0
    return y
def df(f,x):
    dx=0.001
    y=(f(x+dx)-f(x-dx))/(2.0*dx)
    return y
def d2f(f,x):
    dx=0.001
    y=(f(x+dx)-2.0*f(x)+f(x-dx))/(dx*dx)
    return y
x=float(input("x="))
y=f(x);
dy=df(f,x)
d2y=d2f(f,x)
print("x=",x,"y=",y,"dy=",dy,"d2y=",d2y)
```

```
runfile('E:/okul/SCO1/derivative1.py', wdir='E:/okul/SCO1')

x=1
x= 1.0 y= 4.0 dy= 0.0 d2y= 2.000000000279556
```

```
def f(x):
    y=x*x-2.0*x+5.0
    return y
def df(f,x):
    dx=0.001
    y=(-f(x+2.0*dx)+8.0*f(x+dx)-8.0*f(x-dx)+f(x-2.0*dx))/(12.0*dx)
```

```
    return y
def d2f(f,x):
    dx=0.001
    y=(-f(x+2.0*dx)+16.0*f(x+dx)-30.0*f(x)+16.0*f(x-dx)-f(x-2.0*dx))/(12.0*dx*dx)
    return y
x=float(input("x="))
y=f(x);
dy=df(f,x)
d2y=d2f(f,x)
print("x=",x,"y=",y,"dy=",dy,"d2y=",d2y)
```

```
runfile('E:/okul/SCO1/derivative2.py', wdir='E:/okul/SCO1')

x=1
x= 1.0 y= 4.0 dy= -7.401486830834377e-14 d2y= 2.00000000064963
```

Using if_x abstract class for derivatives

```
from math import *
from if_x import *;

class f1(if_x):
  def func(self,x):
      y=x*x-2;
      return y;

x=[1.2,2.3,3.4,4.5,5.6,6.0,5.7,5.1,4.6]
n=len(x)
f = f1()
for i in range(n):
    print("i = ",i," x = ",x[i],"y=f(x) = ",f.func(x[i]),"dy/dx=df(x)/dx = ",f.dfunc(x[i],1))
```

```
n [27]: runfile('E:/okul/SCO1/if_x_test.py', wdir='E:/okul/SCO1')
Reloaded modules: if_x
i =  0  x =  1.2 y=f(x) =  -0.56 dy/dx=df(x)/dx =  2.399999999999504
i =  1  x =  2.3 y=f(x) =  3.289999999999999 dy/dx=df(x)/dx =  4.599999999999715
i =  2  x =  3.4 y=f(x) =  9.559999999999999 dy/dx=df(x)/dx =  6.7999999999986755
i =  3  x =  4.5 y=f(x) =  18.25 dy/dx=df(x)/dx =  9.000000000005155
i =  4  x =  5.6 y=f(x) =  29.359999999999996 dy/dx=df(x)/dx =  11.200000000007698
i =  5  x =  6.0 y=f(x) =  34.0 dy/dx=df(x)/dx =  12.000000000013854
i =  6  x =  5.7 y=f(x) =  30.490000000000002 dy/dx=df(x)/dx =  11.400000000007774
i =  7  x =  5.1 y=f(x) =  24.009999999999998 dy/dx=df(x)/dx =  10.200000000005225
i =  8  x =  4.6 y=f(x) =  19.159999999999997 dy/dx=df(x)/dx =  9.200000000007734
```

The derivative formula coefficients can be calculated numerically by using Taylor expansion. The general derivation formula is applied into abstract class f_x in order to achieve a general derivation format of any given function. Algorithm used in here is given by Bength Fornberg[47] article "Generation of Finite Difference Formulas on Arbitrary Spaced Grids" an has the following form:

$$\left.\frac{d^m f(x)}{dx^m}\right|_{x=x_0} \approx \sum_{\nu=0}^{n} \delta_{n,\nu}^{m} f(x+\alpha_\nu h) \quad m=0,1,\ldots,M;\, n=m,m+1,\ldots,N$$

Enter M,x_0,α_0 ,α_1 ,α_2 ,α_3 ,... ,α_N

$\delta_{0,0}^{0} := 1$

$c1 := 1$

for n:=1 to N do

  c2:=1

  for ν:=0 to n-1 do

    c3:=α_n - α_ν

    c2:=c2*c3

    if $n \le M$ *then* $\delta_{n-1,\nu}^{n} := 0$

    For m:=0 to min(n,M) do

      $\delta_{n,\nu}^{m} := ((\alpha_n - x_0)\delta_{n-1,\nu}^{n} - m\delta_{n-1,\nu}^{m-1}) / c3$

    next m

next ν

For m:=0 to min(n,M) do

$$\delta_{n,n}^{m} := \frac{c1}{c2}\left[m\delta_{n-1,n-1}^{m-1} - (\alpha_{n-1} - x_0)\delta_{n-1,n-1}^{m}\right]$$

next m

c1:=c2

next n

Defining abstract class f_x by using the definition above

```
from abc import ABC, abstractmethod
from math import *

class f_x(ABC):

    @abstractmethod
    def func(self,x):
        pass;

    def dfunc(self,x):
        h=1e-4
        n=1
        M=10
        return self.dnMfunc(x,n,M,h)

    def d2func(self,x):
        h=1e-4
        n=2
        M=10
        return self.dnMfunc(x,n,M,h)

    def d3func(self,x):
        h=1e-4
        n=3
        M=10
        return self.dnMfunc(x,n,M,h)

    def dnfunc(self,x,n):
        h=1e-4
        M=10
        return self.dnMfunc(x,n,M,h)

    def dnMfunc(self,x,N,Mi,hi):
        # order of the maximum derivative
        # N order of derivative
        # M degree of difference formula
        M=Mi;
        #double a[]=new double[0];
        h=hi;
        x0=0.0;
        alphai=[0,1,-1,2,-2,3,-3,4,-4,5,-5,6,-6,7,-7,8,-8,9,-9,10,-10,11,-11,12,-12,13,-13,14,-14,15,
        -15,16,-16,17,-17,18,-18,19,-19,20,-20,21,-21,22,-22,23,-23,24,-24,25,-25,26,-26,27,-27,28,-28,29,-29,30,-30,
        31,-31,32,-32,33,-33,34,-34,35,-35,36,-36,37,-37,38,-38,39,-39,40,-40,41,-41,42,-42,43,-43,44,-44,45,-45,46,
        -46,47,-47,48,-48,49,-49,50,-50,51,-51,52,-52,53,-53,54,-54,55,-55,56,-56,57,-57,58,-58,59,-59,60,-60,
        -61,61,-62,62,-63,63,-64,64,-65,65,-66,66,-67,67,-68,68,-70,70,-71,71,-72,72,-73,73,-74,74,-75,75,
        -76,76,-77,77,-78,78,-79,79,-80,80,-81,81,-82,82,-83,83,-84,84,-85,85,-86,86,-87,87,
        -88,88,-89,89,-90,90,-91,91,-92,92,-93,93,-94,94,-95,95,-96,96,-97,97,-98,98,-99,99,-100,100]
        alpha=alphai
        N1=len(alpha)-1;
        # M   degree of highest derivative
        #N+1 number of coefficients
        delta = [[[0 for x in range(M+1)] for y in range(N1+1)] for z in range(N1+1)];
        #double delta[][][]=new double[N1+1][N1+1][M+1];
        #double c1,c2,c3;
        delta[0][0][0]=1.0;
        c1=1.0
        for n in range(1,N1+1):
            c2=1
            for nu in range(0,n):
                c3=alpha[n]-alpha[nu]
                c2=c2*c3
                if n<=M: delta[n-1][nu][n]=0.0
                for m in range(0,(min(n,M)+1)):
```

```
                if m==0:
                    delta[n][nu][m]=((alpha[n]-x0)*delta[n-1][nu][m])/c3
                else:
                    delta[n][nu][m]=((alpha[n]-x0)*delta[n-1][nu][m]-m*delta[n-1][nu][m-1])/c3
            for m in range(0,(min(n,M)+1)):
                if m==0:delta[n][n][m]=c1/c2*(-(alpha[n-1]-x0)*delta[n-1][n-1][m])
                elif m!=0: delta[n][n][m]=c1/c2*(m*delta[n-1][n-1][m-1]-(alpha[n-1]-x0)*delta[n-1][n-1][m])
            c1=c2
        c=delta
        if Mi<N: M=N
        else: M=Mi
        h=hi;
        deriv=0
        h1=1.0/h
        h2=1.0;
        for j in range(N):
          h2=h2*h1
        for i in range(0,len(c[0])):
          deriv=deriv+c[M][i][N]*self.func(x+alpha[i]*h)
        return deriv*h2

    def gauss_legendre_coefficients(self,x1,x2,n):
        #calculates legendre gauss-coefficients as coefficients of the integral
        #for n terms
        eps=3e-15
        m=int((n+1.0)/2.0)
        xm=0.5*(x2+x1)
        xl=0.5*(x2-x1)
        nmax=100
        a=[[0.0 for i in range(n)] for j in range(2)]
        for i in range(1,m+1):
            z=cos(pi*((i-0.25)/(n+0.5)))
            for ii in range(nmax):
                #while abs(z-z1) > eps:
                p1=1.0;
                p2=0.0;
                for j in range(1,n+1):
                    p3=p2
                    p2=p1
                    p1=(float)(((2.0*j-1.0)*z*p2-(j-1.0)*p3)/j)
                pp=(float)(n*(z*p1-p2)/(z*z-1.0))
                z1=z;
                z=z1-p1/pp
                if abs(z-z1) < eps: break
            a[0][i-1]=xm-xl*z
            a[0][n-i]=xm+xl*z
            a[1][i-1]=2.0*xl/((1.0-z*z)*pp*pp)
            a[1][n-i]=a[1][i-1]
        return a

    def integral(self,x1,x2,n):
        #integral func(x)dx
        #integral of a function by using gauss-legendre quadrature
        #between x1 and x2
        a=self.gauss_legendre_coefficients(x1,x2,n)
        z=0
        for i in range(n):
            z=z+a[1][i]*self.func(a[0][i])
        return z
```

```
from math import *
from f_x import *;

class f1(f_x):
  def func(self,x):
      y=x*x-2;
      return y;

x=[1.2,2.3,3.4,4.5,5.6,6.0,5.7,5.1,4.6]
n=len(x)
f = f1()
for i in range(n):
    print("i = ",i," x = ",x[i],"y=f(x) = ",f.func(x[i]),"dy/dx=df(x)/dx = ",f.dfunc(x[i]))
```

```
runfile('E:/okul/SCO1/f_x.py', wdir='E:/okul/SCO1')

runfile('E:/okul/SCO1/if_x_test.py', wdir='E:/okul/SCO1')
```

```
i =  0  x =  1.2 y=f(x) =  -0.56 dy/dx=df(x)/dx =  2.3999999999997
i =  1  x =  2.3 y=f(x) =  3.289999999999999 dy/dx=df(x)/dx =  4.6000000000321135
i =  2  x =  3.4 y=f(x) =  9.559999999999999 dy/dx=df(x)/dx =  6.800000000043178
i =  3  x =  4.5 y=f(x) =  18.25 dy/dx=df(x)/dx =  8.999999999982425
i =  4  x =  5.6 y=f(x) =  29.359999999999996 dy/dx=df(x)/dx =  11.199999999972274
i =  5  x =  6.0 y=f(x) =  34.0 dy/dx=df(x)/dx =  11.999999999983274
i =  6  x =  5.7 y=f(x) =  30.490000000000002 dy/dx=df(x)/dx =  11.400000000032911
i =  7  x =  5.1 y=f(x) =  24.009999999999998 dy/dx=df(x)/dx =  10.199999999996114
i =  8  x =  4.6 y=f(x) =  19.159999999999997 dy/dx=df(x)/dx =  9.199999999963438
```

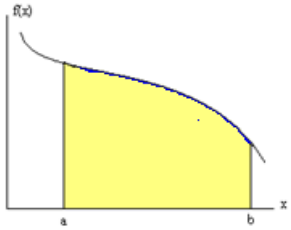
4. INTEGRAL

4.1 DEFINITIONS

Integral defines the area under the curve. It can also be defined as antiderivative(inverse derivative)
It can be defined as without limits, or with integration boundaries

$$I = \int f(x)dx$$

Or with limits



$$I = \int_a^b f(x)dx$$

Some analytical integration formulas are given below:

f(x)	$\int f(x)dx$
$\int x^n\,dx$	$\frac{x^{n+1}}{n+1}$
$\int x^3\,dx$	$\frac{x^4}{4}$
$\int e^x dx$	e^x
$\int e^{bx} dx$	$\frac{e^x}{b}$
$\int a^{bx}\,dx$	$\frac{a^{bx}}{\mathrm{bln}(a)}$
$\int \cos(x)\,dx$	$\sin(x)$
$\int \sin(x)\,dx$	$-\cos(x)$
$\int sec^2(x)\,dx$	$\tan(x)$
$\int \cosh(x)\,dx$	$\sinh(x)$
$\int \sinh(x)\,dx$	$\cos\mathrm{h}(x)$
$\int sech^2(x)\,dx$	$\tanh(x)$
$\int \frac{dx}{x}$	$\ln(\lvert x\rvert)$
$\int \frac{dx}{\sqrt{1-x^2}}$	$\arcsin(x)$
$\int \frac{dx}{\sqrt{a^2-x^2}}$	$\arcsin\left(\frac{x}{a}\right)$
$\int -\frac{dx}{\sqrt{1-x^2}}$	arccos(x)
$\int \frac{dx}{1+x^2}$	arctan(x)+C
$\int \frac{dx}{a^2+x^2}$	$\frac{1}{a}arctan\left(\frac{x}{a}\right)$
$\int \frac{dx}{x\sqrt{x^2-a^2}}$	$\frac{1}{a}arcsec\left(\left\lvert\frac{x}{a}\right\rvert\right)$
$\int \frac{dx}{\sqrt{a^2+x^2}}$	$ln\left(x+\sqrt{a^2+x^2}\right)$

$\int \frac{dx}{\sqrt{a^2-x^2}}$	$ln\left(\left\|x+\sqrt{a^2-x^2}\right\|\right)$
$\int \frac{dx}{a^2-x^2}$	$\frac{1}{2a}ln\left(\left\|\frac{x+a}{x-a}\right\|\right)$
$\int \frac{dx}{x\sqrt{a^2-x^2}}$	$-\frac{1}{a}ln\left(\left\|\frac{a+\sqrt{a^2-x^2}}{x}\right\|\right)$
$\int \frac{dx}{x\sqrt{a^2+x^2}}$	$-\frac{1}{a}ln\left(\left\|\frac{a+\sqrt{a^2+x^2}}{x}\right\|\right)$
$\int u(x)v'(x)\,dx$	$u(x)v(x)-\int u'(x)v(x)\,dx$
$\int ku(x)\,dx$	$k\int u(x)\,dx$
$\int ln(x)\,dx$	$xln(x)-x$

Examples of analytical integrations

EX1: take the integral of $y=f(x)=2x^3+3x^2+5$

Answer: $I=\frac{2}{4}x^4+\frac{3}{3}x^3+5x+C$

EX2: take integral of $y=f(x)=u'(x)v(x)=(2x^3+3x^2+5)(2x-2)$

$v(x)=\frac{2}{4}x^4+\frac{3}{3}x^3+5x \qquad u'(x)=2$

$\int u(x)v'(x)\,dx=u(x)v(x)-\int u'(x)v(x)\,dx$

$$\int u(x)v'(x)\,dx=\left(\frac{2}{4}x^4+\frac{3}{3}x^3+5x\right)(2x-2)-\int 2\left(\frac{2}{4}x^4+\frac{3}{3}x^3+5x\right)dx$$

$$\int u(x)v'(x)\,dx=\left(\frac{2}{4}x^4+\frac{3}{3}x^3+5x\right)(2x-2)-2\left(\frac{2}{20}x^5+\frac{3}{12}x^4+\frac{5}{2}x^2\right)+C$$

EX3: take integral of

$$I=\int x^2e^xdx$$

$$\frac{d}{dx}(uv)=udv+vdu$$

$$\int udv=uv-\int vdu$$

$u=x^2 \quad dv=e^xdx \quad du=2xdx \quad v=e^x$

$\int x^2e^xdx=x^2e^x-\int 2xe^xdx$

$u=x \quad dv=e^xdx \quad du=dx \quad v=e^x$

$$\int x^2e^xdx=x^2e^x-2\left[xe^x-\int e^xdx\right]=x^2e^x-2xe^x+2e^x+C$$

EX4

Calculate $\int \frac{3x^2+2x+3}{(x^2+1)^2}$

$$\frac{3x^2+2x+3}{(x^2+1)^2}=\frac{Ax+B}{x^2+1}+\frac{Bx+D}{(x^2+1)^2}=\frac{(Ax+B)(x^2+1)+Bx+D}{(x^2+1)^2}$$

$$\frac{3x^2+2x+3}{(x^2+1)^2}=\frac{Ax^3+Bx^2+(A+C)x+(B+D)}{(x^2+1)^2}$$

So $A = 0 \;\; B = 3 \;\; C = 2 \;\; D = 0$

$$\int \frac{3x^2 + 2x + 3}{(x^2+1)^2} = \int \frac{3dx}{(x^2+1)} + \int \frac{2xdx}{(x^2+1)^2}$$

$u = (x^2 + 1) \;\; du = 2xdx$

$$\int \frac{2xdx}{(x^2+1)^2} = \int \frac{du}{u^2} = \int u^{-2}du = -\frac{1}{u} = -\frac{1}{(x^2+1)}$$

$\int \frac{3x^2+2x+3}{(x^2+1)^2} = 3\text{atan}(x) - \frac{1}{(x^2+1)}$+c

EX5

Calculate $\int cos^2(x)dx$

$cos^2(x)$+= 1

$cos^2(x) - sin^2(x) = \cos(2x)$

$cos^2(x) = \frac{1+\cos(2x)}{2}$

$$\int cos^2(x)dx = \int \frac{1+\cos(2x)}{2}\, dx = \frac{x}{2} + \frac{\sin(2x)}{4}$$

EX6

Calculate $\int xsin^2(x)dx$

$$\int xsin^2(x)dx = \int \frac{x(1-\cos(2x)}{2}dx = \int \frac{x}{2}dx - \int \frac{\text{xcos}(2x)}{2}dx = \frac{x^2}{4} - \int \frac{\text{xcos}(2x)}{2}dx$$

$$\int \frac{\text{xcos}(2x)}{2}dx$$

$$\int udv = uv - \int vdu$$

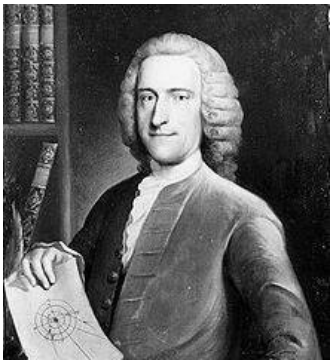
u=x dv=cos(2x)dx $du = dx \quad v = \int \cos(2x)\, dx = \frac{\sin(2x)}{2}$

$$\int \frac{\text{xcos}(2x)}{2}dx = x\frac{\sin(2x)}{2} - \int \frac{\sin(2x)}{2}dx = x\frac{\sin(2x)}{2} + \frac{\cos(2x)}{4}$$

$$\int xsin^2(x)dx = \frac{x^2}{4} - \left[x\frac{\sin(2x)}{2} + \frac{\cos(2x)}{4}\right] + C$$

4.2 NEWTON-COTES NUMERICAL INTEGRAL

If function f(x) is given and ,it is desired to calculate integral of the function between integration limits a and b. Definition of the integral is the area under the function between integration limits. The simplest approximation is to connect f(a) and f(b) points with a linear line and calculate the area of the trapezoidal under the line. This approach is called trapezoidal rule of Newton-Cotes formulations (Area finding formulations)



Roger Cotes

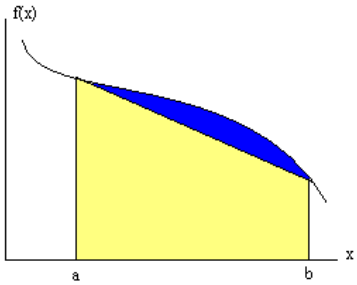


Figure 4.2 Finding integration with trapezoidal rule (yellow area) and error created by the method(blue area)

Integration formula :

$$I = \int_a^b f(x)dx = (b-a)\frac{f(a)+f(b)}{2}$$

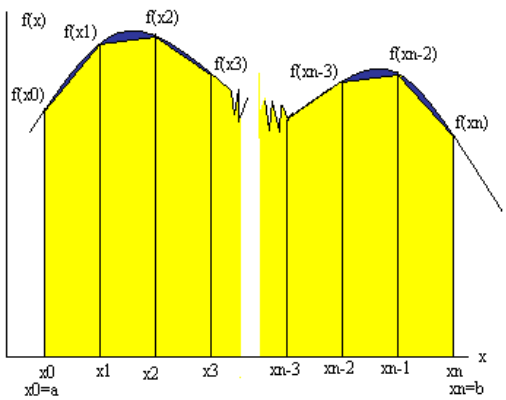


Figure 4.3 Finding integration with trapezoidal rule (yellow area) with multiple trapezoidal rule

If total area is divided into smaller areas amount of error will be reduced due to the fact that smaller piece of functions are closer to the line compare to a big piece of function. The same integration Formula for multipart equation will be:

$$I = \int_a^b f(x)dx = (x_n - x_0)\frac{f(x_0) + 2\sum_{i=1}^{n-1} f(x_i) + f(x_n)}{2n}$$

Now assume an example integration function of

$$I = \int_0^1 \frac{4}{\pi}\sqrt{1-x^2}dx = 1.0$$

Since the value of the function is known exactly, it will be easy to evaluate the error of numerical function evaluation process.

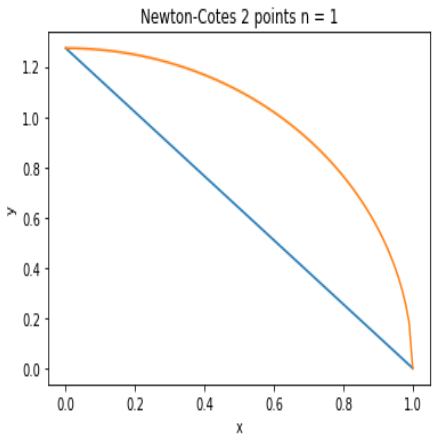
Trapezoidal intergral rule of Newton-Cotes approximation n=1

```
 from math import *
from if_x import *
from matplotlib import pyplot as plt
import numpy as np

class f1(if_x):
  def func(self,x):
     y=4.0/pi*sqrt(1.0-x*x)
     return y

def newton_cotes2(f,a,b,n):
   #Newton-cotes integral 2 points
   # trapezoidal rule
   # ff integrating function
   # a,b integral limits
   # n : number of sub division h=(b-a)/n;
   h=(b-a)/n;
   h1=h;
   sum=0.0
   nn=2*n
   x =[0 for w in range(2)]
   y =[0 for w in range(2)]
   xx =[0 for w in range(nn)]
   yy =[0 for w in range(nn)]
   j=0
   for p in range(n):
      for i in range(2):
         x[i]=a+i*h1+p*h
         xx[j]=x[i]
         y[i]=f.func(x[i])
         yy[j]=y[i]
         j=j+1
      sum=sum+h*(y[0]+y[1])/2.0
   n1=100
   xx1 =[0 for w in range(n1)]
   yy1 =[0 for w in range(n1)]
   dx=(b-a)/(n1-1)
   for i in range(n1):
     xx1[i]=dx*i
     yy1[i]=f.func(xx1[i])
   plt.plot(xx,yy,xx1,yy1)
   plt.xlabel("x")
   plt.ylabel("y")
   plt.title("Newton-Cotes 2 points n = "+str(n))
   plt.show()
   return sum

f=f1()
I= newton_cotes2(f,0.0,1.0,1)
print("I=",I)
```

```
 runfile('E:/okul/SCO1/newton_cotes2.py', wdir='E:/okul/SCO1')

Reloaded modules: if_x
```

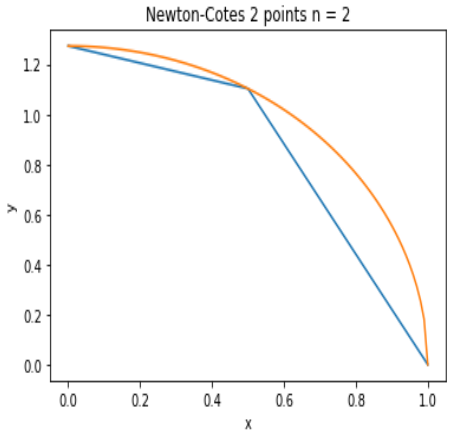
```
I= 0.6366197723675814
```

In order to see the error happened in the function visiually a graphical output form of the same program is created.

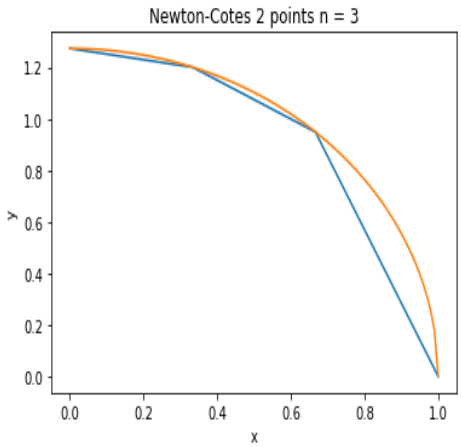


Now number of subsections will increase to see the change of the error

```
runfile('E:/okul/SCO1/newton_cotes2.py', wdir='E:/okul/SCO1')
Reloaded modules: if_x
I= 0.8696387816055828
```

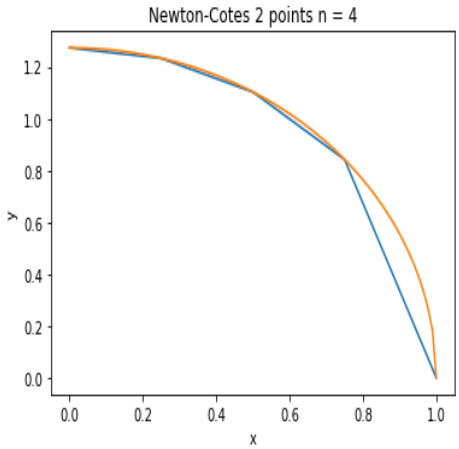



```
runfile('E:/okul/SCO1/newton_cotes2.py', wdir='E:/okul/SCO1')
Reloaded modules: if_x
I= 0.9286860839333196
```

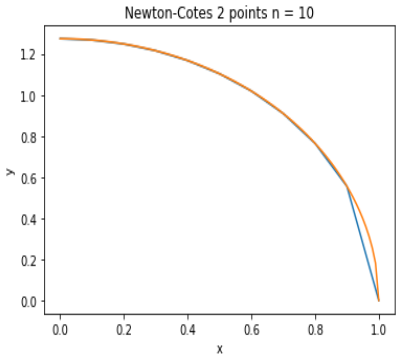



```
runfile('E:/okul/SCO1/newton_cotes2.py', wdir='E:/okul/SCO1')
```

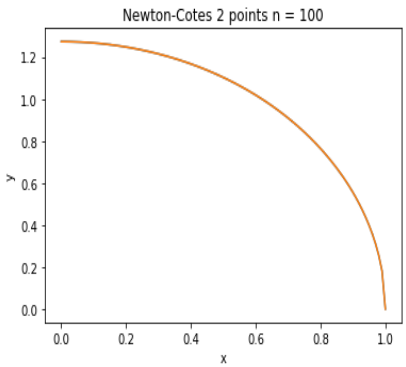
```
Reloaded modules: if_x
I= 0.9535638125074375
```




```
runfile('E:/okul/SCO1/newton_cotes2.py', wdir='E:/okul/SCO1')
Reloaded modules: if_x
I= 0.9881988750835945
```




```
runfile('E:/okul/SCO1/newton_cotes2.py', wdir='E:/okul/SCO1')
Reloaded modules: if_x
I= 0.9996257879552262
```



As it is seen from the example when trapezoidal steps are increased, approximation is getting better but even with 100 steps there are stil some error in the approximation. The next step in the approach is to use higher degrees of polinomials to connect points of integration. Quadratic (second degree) polinomial rule is called **Simpson 1/3 rule**. If step size of formulation is taken as h

$$h = \frac{b-a}{n}$$

Quadratic polynomial Newton-Cotes formula(Simpson 1/3 rule)

$$\boldsymbol{I} = \int_a^b f(x)dx = \frac{(b-a)}{6}[f(x_0) + 4f(x_1) + f(x_2)]$$

Or

$$h = \frac{b-a}{n} = \frac{b-a}{2}$$

$$\boldsymbol{I} = \int_a^b f(x)dx = \frac{h}{3}[f(x_0) + 4f(x_1) + f(x_2)]$$

(n=2 or three points are given in this Formula)

Quadratic polynomial Newton-Cotes formula(Simpson 1/3 rule) with multipoint approach

$$\boldsymbol{I} = \int_a^b f(x)dx = \frac{h}{3}\left[f(x_0) + 4\sum_{i=1,3,5,..}^{n-1} f(x_i) + \sum_{i=2,4,6,..}^{n-2} f(x_i) + f(x_n)\right]$$

Cubic polynomial Newton-Cotes formula(Simpson 3/8 rule)

$$\boldsymbol{I} = \int_a^b f(x)dx = \frac{(b-a)}{8}[f(x_0) + 3f(x_1) + 3f(x_2) + f(x_3)]$$

Or

$$h = \frac{b-a}{n} = \frac{b-a}{3}$$

$$\boldsymbol{I} = \int_a^b f(x)dx = \frac{3h}{8}[f(x_0) + 3f(x_1) + 3f(x_2) + f(x_3)]$$

(n=3 or four points are required for this formula)

Fourth order polynomial Newton-Cotes formula(Bode rule)

$$\boldsymbol{I} = \int_a^b f(x)dx = \frac{(b-a)}{90}[7f(x_0) + 32f(x_1) + 12f(x_2) + 32f(x_3) + 7f(x_4)]$$

$$h = \frac{b-a}{n} = \frac{b-a}{4}$$

$$\boldsymbol{I} = \int_a^b f(x)dx = \frac{2h}{45}[7f(x_0) + 32f(x_1) + 12f(x_2) + 32f(x_3) + 7f(x_4)]$$

(n=4 or five points are used in this Formula)

Fifth order polynomial Newton-Cotes formula(Boole rule)

$$\boldsymbol{I} = \int_a^b f(x)dx = \frac{(b-a)}{288}[19f(x_0) + 75f(x_1) + 50f(x_2) + 50f(x_3) + 75f(x_4) + 19f(x_5)]$$

Or

$$h = \frac{b-a}{n} = \frac{b-a}{5}$$

$$\boldsymbol{I} = \int_a^b f(x)dx = \frac{5h}{288}[19f(x_0) + 75f(x_1) + 50f(x_2) + 50f(x_3) + 75f(x_4) + 19f(x_5)]$$

(n=5 or six points are used in this Formula)

Eight order polynomial Newton-Cotes formula

$$\boldsymbol{I} = \int_a^b f(x)dx = \frac{(b-a)}{28350}[989f(x_0) + 5888f(x_1) - 928f(x_2) + 10496f(x_3) - 4540f(x_4) + 10496f(x_5) - 928f(x_6) + 58888f(x_7) + 989f(x_8)]$$

Or

$$h = \frac{b-a}{n} = \frac{b-a}{8}$$

$$\boldsymbol{I} = \int_a^b f(x)dx = \frac{8h}{28350}[989f(x_0) + 5888f(x_1) - 928f(x_2) + 10496f(x_3) - 4540f(x_4) + 10496f(x_5) - 928f(x_6) + 58888f(x_7) + 989f(x_8)]$$

(n=8 or nine points are used in this Formula)

Programs for this approaches are given below:

quadratic polynomial Newton - Cotes Formula (Simpson 1/3 rule)

```
from math import *
from if_x import *


class f1(if_x):
  def func(self,x):
     y=4.0/pi*sqrt(1.0-x*x)
     return y

def newton_cotes3(f,a,b,n):
   #Newton-cotes integral 3 points
   # Simpson 1/3 rule
   # ff integrating function
   # a,b integral limits
   # n : number of sub division h=(b-a)/n;
   h=(b-a)/n;
   h1=h/2.0;
   sum=0.0
   nn=2*n
   n1=3
```

```
    x =[0 for w in range(n1)]
    y =[0 for w in range(n1)]
    for p in range(n):
        for i in range(n1):
            x[i]=a+i*h1+p*h
            y[i]=f.func(x[i])
        sum=sum+h*(y[0]+4.0*y[1]+y[2])/6.0
    return sum

f=f1()
n=1
I= newton_cotes3(f,0.0,1.0,n)
print("n = ",n,"I=",I)
```

If the quadratic Newton-Cotes Formula (Simpson 1/3 rule) applied as one time integral zone (0 to 1)

```
runfile('E:/okul/SCO1/newton_cotes3.py', wdir='E:/okul/SCO1')
Reloaded modules: if_x
n =  1 I= 0.9473117846849166
```

If the quadratic Newton-Cotes Formula (Simpson 1/3 rule) applied as two times integral zone (0 to 1)

```
runfile('E:/okul/SCO1/newton_cotes3.py', wdir='E:/okul/SCO1')
Reloaded modules: if_x
n =  2 I= 0.9815388228080559
```

If the quadratic Newton-Cotes Formula (Simpson 1/3 rule) applied as 10 times integral zone (0 to 1)

```
runfile('E:/okul/SCO1/newton_cotes3.py', wdir='E:/okul/SCO1')
Reloaded modules: if_x
n =  10 I= 0.9983621080451945
```

cubic polynomial Newton - Cotes Formula (Simpson 3/8 rule)

```
from math import *
from if_x import *


class f1(if_x):
  def func(self,x):
      y=4.0/pi*sqrt(1.0-x*x)
      return y

def newton_cotes4(f,a,b,n):
    #Newton-cotes integral 4 points
    # Simpson 3/8 rule
    # ff integrating function
    # a,b integral limits
    # n : number of sub division h=(b-a)/n;
    h=(b-a)/n;
    h1=h/3.0;
    sum=0.0
    n1=4
    x =[0 for w in range(n1)]
    y =[0 for w in range(n1)]
    for p in range(n):
        for i in range(n1):
            x[i]=a+i*h1+p*h
            y[i]=f.func(x[i])
        sum=sum+h*(y[0]+3.0*y[1]+3.0*y[2]+y[3])/8.0
    return sum

f=f1()
n=1
```

```
I= newton_cotes4(f,0.0,1.0,n)
print("n = ",n,"I=",I)
```

If the cubic Newton-Cotes Formula (Simpson 3/8 rule) applied as one time integral zone (0 to 1)

```
runfile('E:/okul/SCO1/newton_cotes4.py', wdir='E:/okul/SCO1')
Reloaded modules: if_x
n =  1 I= 0.965194372879037
```

If the cubic Newton-Cotes Formula (Simpson 3/8 rule) applied as two times integral zone (0 to 1)

```
runfile('E:/okul/SCO1/newton_cotes4.py', wdir='E:/okul/SCO1')
Reloaded modules: if_x
n =  2 I= 0.9877870530380952
```

If the cubic Newton-Cotes Formula (Simpson 3/8 rule) applied as 10 times integral zone (0 to 1)

```
runfile('E:/okul/SCO1/newton_cotes4.py', wdir='E:/okul/SCO1')
Reloaded modules: if_x
n =  10 I= 0.998914915292993
```

fourth order polynomial Newton - Cotes Formula (Boole rule)

```
 from math import *
from if_x import *


class f1(if_x):
  def func(self,x):
      y=4.0/pi*sqrt(1.0-x*x)
      return y

def newton_cotes5(f,a,b,n):
    #Newton-cotes integral 5 points
    # Bole rule
    # ff integrating function
    # a,b integral limits
    # n : number of sub division h=(b-a)/n;
    h=(b-a)/n;
    h1=h/4.0;
    sum=0.0
    n1=5
    x =[0 for w in range(n1)]
    y =[0 for w in range(n1)]
    for p in range(n):
        for i in range(n1):
            x[i]=a+i*h1+p*h
            y[i]=f.func(x[i])
        sum=sum+h*(7.0*y[0]+32.0*y[1]+12.0*y[2]+32.0*y[3]+7.0*y[4])/90.0
    return sum

f=f1()
n=1
I= newton_cotes5(f,0.0,1.0,n)
print("n = ",n,"I=",I)
```

If the fourth order polynomial Newton-Cotes Formula (Boolee rule) applied as one time integral zone (0 to 1)

```
 runfile('E:/okul/SCO1/newton_cotes5.py', wdir='E:/okul/SCO1')
Reloaded modules: if_x
n =  1 I= 0.9838206253495985
```

If the fourth order polynomial Newton-Cotes Formula (Boole rule) applied as two times integral zone (0 to 1)

```
runfile('E:/okul/SCO1/newton_cotes5.py', wdir='E:/okul/SCO1')
Reloaded modules: if_x
n =  2 I= 0.9943031391613758
```

If the fourth order polynomial Newton-Cotes Formula (Boole rule) applied as 10 times integral zone (0 to 1)

```
runfile('E:/okul/SCO1/newton_cotes5.py', wdir='E:/okul/SCO1')
Reloaded modules: if_x
n =  10 I= 0.9994921635232705
```

8th order Newton-cotes formula:

```
from math import *

def newton_cotes9(f,a,b,n):
   #Newton-cotes integral 5 points
   # Bole rule
   # ff integrating function
   # a,b integral limits
   # n : number of sub division h=(b-a)/n;
   h=(b-a)/n;
   h1=h/8.0;
   sum=0.0
   n1=9
   x =[0 for w in range(n1)]
   y =[0 for w in range(n1)]
   for p in range(n):
      for i in range(n1):
         x[i]=a+i*h1+p*h
         y[i]=f(x[i])
      sum=sum+h/28350.0*(989.0*y[0]+5888.0*y[1]-928.0*y[2]+10496.0*y[3]-4540.0*y[4]+10496.0*y[5]-
928.0*y[6]+5888*y[7]+989.0*y[8])
   return sum

f = lambda x: 4.0/pi*sqrt(1.0-x*x)
n=1
I= newton_cotes9(f,0.0,1.0,n)
print("n = ",n,"I=",I)
```

If the eight order polynomial Newton-Cotes Formula applied as one time integral zone (0 to 1)

```
runfile('E:/okul/SCO1/newton_cotes9.py', wdir='E:/okul/SCO1')
n =  1 I= 0.9952701383291108
```

If the eight order polynomial Newton-Cotes Formula applied as two time integral zone (0 to 1)

```
runfile('E:/okul/SCO1/newton_cotes9.py', wdir='E:/okul/SCO1')
n =  2 I= 0.998330566008359
```

If the eight order polynomial Newton-Cotes Formula applied as 10 time integral zone (0 to 1)

```
runfile('E:/okul/SCO1/newton_cotes9.py', wdir='E:/okul/SCO1')
n =  10 I= 0.9998508846967017
```

The results will be summarised as one table

Table comparison of Newton-Cotes results for different degree of integrating polnomials

$$I = \int_0^1 \frac{4}{\pi}\sqrt{1-x^2}dx = 1.0$$

N	Integral approximation	Error ($I_{approximation}$-I_{exact})
Region is not divided up calculated as one piece		
1 (Trapezoidal)	0.6366197723675814	0.3633802276324186
2 (Simpson 1/3)	0.9473117846849166	0.052688215315083387
3 (Simpson 3/8)	0.965194372879037	0.03480562712096302
4 (Bode)	0.9838206253495985	0.016179374650401535
5	0.9872489399864514	0.012751060013548643
8	0.9952701383291108	0.004729861670889179

Region is divided into 2 equal pieces (0-0.5 and 0.5-1)		
1 (Trapezoidal)	0.8696387816055828	0.13036121839441717
2 (Simpson 1/3)	0.9815388228080559	0.018461177191944067
3 (Simpson 3/8)	0.9877870530380952	0.01221294696190478
4 (Bode)	0.9943031391613758	0.005696860838624196
5	0.9955081956385212	0.0044918043614787795
8	0.998330566008359	0.0044918043614787795
Region is divided into 10 equal pieces (0-0.1,0.1-0.2,…,0.9-1)		
1 (Trapezoidal)	0.9881988750835945	0.011801124916405503
2 (Simpson 1/3)	0.9983621080451945	0.0016378919548054593
3 (Simpson 3/8)	0.998914915292993	0.001085084707006967
4 (Bode)	0.9994921635232705	5.07836476729473E-4
5	0.9995994322513025	4.005677486974557E-4
8	0.9998508846967017	1.4911530329830835E-4

The same integration calculated by using excel

$$I = \int_0^1 \frac{4}{\pi}\sqrt{1-x^2}dx = 1.0$$

Direct integration Newton-Cotes formulas

x	f(x)	Trapezoidal	Simpson 1/3
0	1.27324	0.158531	0.314961
0.125	1.263253	0.156004	
0.25	1.232809	0.150821	0.294032
0.375	1.180325	0.142686	
0.5	1.102658	0.131036	0.246688
0.625	0.993922	0.114756	
0.75	0.842169	0.091161	0.137824
0.875	0.616404	0.038525	
1	0		
Integral		0.98352	0.993505

Direct integration Newton-Cotes formulas

x	f(x)	Trapezoidal	Simpson 1/3
0	1.27324	0.0795	0.158739
0.0625	1.27075	0.079188	
0.125	1.263253	0.07856	0.156223
0.1875	1.250658	0.077608	
0.25	1.232809	0.076321	0.151063
0.3125	1.209473	0.074681	
0.375	1.180325	0.072664	0.142972
0.4375	1.14492	0.070237	
0.5	1.102658	0.067355	0.131405
0.5625	1.052711	0.063957	
0.625	0.993922	0.059954	0.115302
0.6875	0.924607	0.055212	
0.75	0.842169	0.049513	0.092241
0.8125	0.742249	0.042458	
0.875	0.616404	0.033109	0.049764
0.9375	0.443069	0.013846	
1	0		
Integral		0.994162	0.99771

Direct integration Newton-Cotes formulas

x	f(x)	Trapezoidal	Simpson 1/3
0	1.27324	0.039779	0.079526

0.03125	1.272618	0.03974	
0.0625	1.27075	0.039662	0.079214
0.09375	1.267632	0.039545	
0.125	1.263253	0.039388	0.078587
0.15625	1.257601	0.039192	
0.1875	1.250658	0.038954	0.077636
0.21875	1.242403	0.038675	
0.25	1.232809	0.038354	0.076351
0.28125	1.221845	0.037989	
0.3125	1.209473	0.03758	0.074712
0.34375	1.19565	0.037125	
0.375	1.180325	0.036621	0.072698
0.40625	1.163438	0.036068	
0.4375	1.14492	0.035463	0.070274
0.46875	1.124692	0.034802	
0.5	1.102658	0.034084	0.067398
0.53125	1.078708	0.033303	
0.5625	1.052711	0.032457	0.064007
0.59375	1.024512	0.031538	
0.625	0.993922	0.030541	0.060014
0.65625	0.960715	0.029458	
0.6875	0.924607	0.028279	0.055289
0.71875	0.885243	0.026991	
0.75	0.842169	0.025577	0.04962
0.78125	0.794779	0.024016	
0.8125	0.742249	0.022276	0.042627
0.84375	0.683394	0.020309	
0.875	0.616404	0.018042	0.033463
0.90625	0.538252	0.015333	
0.9375	0.443069	0.011858	0.017774
0.96875	0.315813	0.004935	
1	0		0
Integral		0.997934	0.999191

EXAMPLE :

$$I = \int_0^1 sin(1 - x + x^2)dx$$

solve with bole rule by hand. Use n=2

Step size h=(b-a)/n

x0	0	0.841471	7	5.890297		
x1	0.125	0.777465	32	24.87888		
x2	0.25	0.726009	12	8.712104		
x3	0.375	0.692988	32	22.17561		
x4	0.5	0.681639	7	4.771471		
				66.42836	0.005556	0.369046
x4	0.5	0.681639	7	4.771471		
x5	0.625	0.692988	32	22.17561		
x6	0.75	0.726009	12	8.712104		
x7	0.875	0.777465	32	24.87888		
x8	1	0.841471	7	5.890297		
				66.42836	0.005556	0.369046
					I=	**0.738093**

```
from math import *

def newton_cotes5(f,a,b,n):
    #Newton-cotes integral 5 points
```

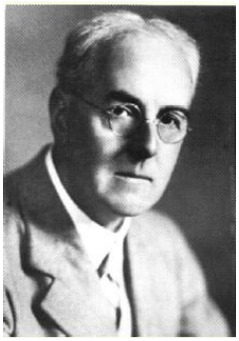
```
    # Bole rule
    # ff integrating function
    # a,b integral limits
    # n : number of sub division h=(b-a)/n;
    h=(b-a)/n;
    h1=h/4.0;
    sum=0.0
    n1=5
    x =[0 for w in range(n1)]
    y =[0 for w in range(n1)]
    for p in range(n):
        for i in range(n1):
            x[i]=a+i*h1+p*h
            y[i]=f(x[i])
        sum=sum+h*(7.0*y[0]+32.0*y[1]+12.0*y[2]+32.0*y[3]+7.0*y[4])/90.0
    return sum

f = lambda x: sin(1.0-x+x*x)
n=1
I= newton_cotes5(f,0.0,1.0,n)
print("n = ",n,"I=",I)
```

```
runfile('E:/okul/SCO1/newton_cotes5_test.py', wdir='E:/okul/SCO1')
Reloaded modules: if_x, newton_cotes5
n =  1 I= 0.7380534760475129
```

7.2 4.3 RICHARDSON AND AITKEN EXTRAPOLATION FORMULAS

7.2.1 RICHARDSON EXTRAPOLATION(ROMBERG INTEGRATION)

Richardson extrapolation formula can be applied to decrease integration error. Richardson extrapolation Formula has the folloving form :



Lewis Fry Richardson

$$I \cong I(h_2) + \frac{1}{\left(\frac{h_1}{h_2}\right)^2 - 1}[I(h_2) - I(h_1)]$$

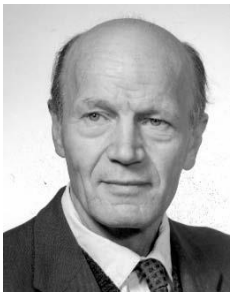
In this Formula h_2 should be smaller than h_1 . If $h_2 = \frac{h_1}{2}$ equation becomes:

$$I \cong \frac{4}{3}I(h_2) - \frac{1}{3}I(h_1)$$

As a starting interpolation any of the Newton-Cotes formulations can be taken. A General form of the above equation can be written as

$$I_{j,k} \cong \frac{4^{k-1}I_{j+1,k-1} - I_{j,k-1}}{4^{k-1} - 1}$$

If trapezoidal rule is used as starting point of this iterative formulation, it is called **Romberg Integration[95]**



Werner Romberg

Consider the previous integral

$$I = \int_0^1 \frac{4}{\pi}\sqrt{1-x^2}dx = 1.0$$

that solved by excel, if the results are integrated into the romberg integration formula:

Romberg: Trapezoidal

2*h	I0	I1	I2
0.125	0.98352	0.99771	0.999685
0.0625	0.994162	0.999191	
0.03125	0.997934		

Romberg: Simpson 1/3

2*h	I0	I1	I2
0.125	0.993505	0.999111	0.999877
0.0625	0.99771	0.999685	
0.03125	0.999191		

Program 4-1 Romberg Integration formula

```
 from math import *
def integral_romberg(f,a,b):
   #integral f(x)dx
   #romberg integration
   #increase n to increase accuracy
   n=8
   R=[[0 for z in range(n+1)] for w in range(n+1)]
   m=1
   h=b-a
   close=1
   tol=1e-40;
   j=0;k=0
   ret=0.0;
   R[0][0]=h/2.0*(f(a)+f(b))
   #for j in range(1,n):
   while(close>tol and j<n):
      j=j+1
      h=h/2.0
      trap(f,a,h,j,m,R)
      m=m*2
      for k in range(1,j+1):
         R[j][k]=R[j][k-1]+(R[j][k-1]-R[j-1][k-1])/(pow(4,k)-1.0)
         close=abs(R[j-1][j-1]-R[j][j])
         ret=R[j][k]
   return ret

def trap(f,a,h,j,m,R):
   sum=0
   for p in range(1,m+1):
      sum=sum+f(a+h*(2*p-1))
   R[j][0]=R[j-1][0]/2.0+h*sum

f = lambda x:2.0/(1.0+2.0*x*x)
```

```
I=integral_romberg(f,-3.0,3.0)
print("I=",I)
```

Romberg integration f(x)= 2.0/(1.0+2.0*x*x)

```
 runfile('E:/okul/SCO1/integral_romberg.py', wdir='E:/okul/SCO1')
I= 3.7881660832213675
```

If the iterative results of Romberg integration is given as a table

Table 4-1 Romberg integration of

$$I = \int_{-3}^{3} \frac{2}{1+2x^2}\,dx$$

6.3157895	8.2105263							
4.2488038	3.5598086	3.2497608						
3.8058295	3.6581714	3.664729	3.6713158					
3.7866756	3.7802909	3.7884322	3.7903958	3.7908628				
3.7877768	3.7881439	3.7886674	3.7886711	3.7886644	3.7886622			
3.7880687	3.788166	3.7881675	3.7881596	3.7881576	3.7881571	3.7881569		
3.7881417	3.7881661	3.7881661	3.7881661	3.7881661	3.7881661	3.7881661	3.7881661	
3.78816	3.7881661	3.7881661	3.7881661	3.7881661	3.7881661	3.7881661	3.7881661	3.7881661

Solving of another integral

$$I = \int_{0}^{\pi} \sin(x)\,dx = 2$$

by romberg Integration Formula

```
f = lambda x:sin(x)
I=integral_romberg(f,0.0,pi)
print("I = ",I)
```

```
runfile('E:/okul/SCO1/integral_romberg.py', wdir='E:/okul/SCO1')
I =  1.9999999999999993
```

Table 4-2 Romberg integration $I = \int_0^{\pi} \sin(x)dx = 2$

1.5707963267948900	2.0943951023931900						
1.8961188979370300	2.0045597549844200	1.9985707318238300					
1.9742316019455500	2.0002691699483800	1.9999831309459800	2.0000055499796700				
1.9935703437723300	2.0000165910479300	1.9999997524545700	2.0000000162880400	1.9999999945872900			
1.9983933609701400	**2.0000010333694100**	**1.9999999961908400**	**2.0000000000596700**	**1.9999999999960300**	**2.0000000000013200**		
1.9995983886400300	2.0000000645300000	1.9999999999407000	2.0000000000002200	1.9999999999999900	1.9999999999999900	1.9999999999999900	
1.9998996001842000	2.0000000040322500	1.9999999999990700	2.0000000000000000	2.0000000000000000	2.0000000000000000	2.0000000000000000	2.0000000000000000
1.9999749002350500	2.0000000002520000	1.9999999999999800	1.9999999999999900	1.9999999999999900	1.9999999999999900	1.9999999999999900	1.9999999999999900

Built-in Romberg integration in python

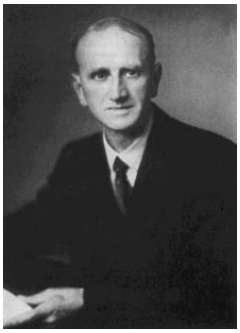
```
import scipy.integrate as integrate
import scipy.special as special
from math import *
f = lambda x: sin(x)
I = integrate.romberg(f, 0, pi)
print("Integral = ",I)
```

```
runfile('C:/Users/meral/.spyder-py3/autosave/integral_scripy.py', wdir='C:/Users/meral/.spyder-py3/autosave')
Integral =  2.000000000001321
```

7.2.1 AITKEN EXTRAPOLATION FORMULA

Another extrapolation Formula is te Aitken extrapolation. This equation is basically derived from Euler-McLaurin Formula. Aitken interpolation has the following form:

$$I = I_{4n} - \frac{(I_{4n} - I_{2n})^2}{(I_{4n} - I_{2n}) - (I_{2n} - I_n)}$$



Alexander Craig "Alec" Aitken

By combining this equation with Simpson 1/3 formulation given above, more accurate results can be obtain

```
from math import *
from if_x import *

class f1(if_x):
  def func(self,x):
      y=4.0/pi*sqrt(1.0-x*x)
      return y

def newton_cotes3(f,a,b,n):
    #Newton-cotes integral 4 points
    # Simpson 3/8 rule
    # ff integrating function
    # a,b integral limits
    # n : number of sub division h=(b-a)/n;
    h=(b-a)/n;
    h1=h/3.0;
    sum=0.0
    nn=2*n
    n1=4
    x =[0 for w in range(n1)]
    y =[0 for w in range(n1)]
    for p in range(n):
        for i in range(n1):
            x[i]=a+i*h1+p*h
            y[i]=f.func(x[i])
        sum=sum+h*(y[0]+3.0*y[1]+3.0*y[2]+y[3])/8.0
    return sum

def Aitken_simpson1_3(ff,a,b,n):
    #Aitken interpolation formula
    I4n=newton_cotes3(ff,a,b,4*n)
    I2n=newton_cotes3(ff,a,b,2*n)
    In=newton_cotes3(ff,a,b,n)
    I=I4n-(I4n-I2n)*(I4n-I2n)/((I4n-I2n)-(I2n-In))
    return I

f=f1()
n=2
while n<257:
    I= Aitken_simpson1_3(f,0.0,1.0,n)
    print("n = ",n,"I=",I)
    n=n*2
```

```
runfile('E:/okul/SCO1/newton_cotes3.py', wdir='E:/okul/SCO1')
Reloaded modules: if_x
n =  2 I= 0.9999928680748176
n =  4 I= 0.9999986596595479
```

```
n =  8 I= 0.99999975624287
n =  16 I= 0.9999999563192528
n =  32 I= 0.9999999922266964
n =  64 I= 0.9999999986213309
n =  128 I= 0.9999999997558837
n =  256 I= 0.9999999999568113
```

4.4 GAUSS-LEGENDRE NUMERICAL INTEGRAL

A problem known from antique times, is to create a rectangle with the same area with a polynomial. IT is called a quadrature problem.

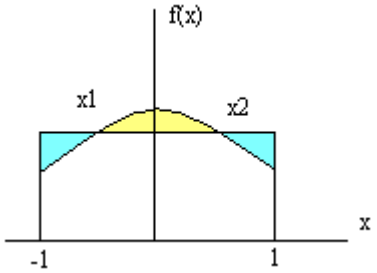


Figure 4.4.1 a rectangle with the same area with a polynomial

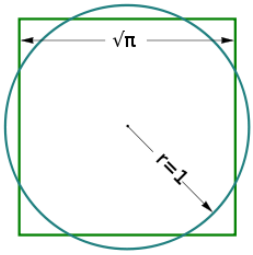


Figure 4.4.2 A rectangle with the same area with a circle

If point x1 and x2 points intersecting the polynomial and rectangle is known, area calculation of the polynomial can be interchange with the area calculation of the rectangle. Or in more general means instead of integration process of summation can be substituted. As a general definition:

$$I_w(-1,1) \cong \int_{-1}^{1} w(x)f(x)dx \cong \sum_{k=1}^{n} c_k f(x_k)$$

Can be written. In this equation w(x) is the weight factor. In the first of a such formulation, Gauss-Legendre integral formulation, weight factor can be taken as 1.

$$I_w(-1,1) \cong \int_{-1}^{1} f(x)dx \cong \sum_{k=1}^{n} c_k f(x_k)$$

If this integration is solved for a general polynamial equation. Following relation is found:

$$\int_{-1}^{1} x^k dx = \frac{1-(-1)^{k+1}}{k+1}$$

This equation can be written in open form as:

$$\int_{-1}^{1} f(x)dx = \frac{1-(-1)^{k+1}}{k+1} \cong \sum_{i=1}^{n} c_i x_i^k = c_1 x_1^k + c_2 x_2^k + c_3 x_3^k + .. + c_n x_n^k$$

For a special case of n=2 equation becomes :

$$f(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3$$

$$\int_{-1}^{1} f(x)dx \cong c_1(a_0 + a_1x_1 + a_2x_1^2 + a_3x_1^3) + c_2(a_0 + a_1x_2 + a_2x_2^2 + a_3x_2^3)$$

$$\cong a_0 \int_{-1}^{1} dx + a_1 \int_{-1}^{1} xdx + a_2 \int_{-1}^{1} x^2dx + a_3 \int_{-1}^{1} x^3dx$$

$$\int_{-1}^{1} f(x)dx \cong a_0\big(1-(-1)\big) + \frac{a_1}{2}(1^2-(-1)^2) + \frac{a_2}{3}(1^3-(-1)^3) + \frac{a_4}{4}(1^4-(-1)^4)$$

$$c_1(a_0 + a_1x_1 + a_2x_1^2 + a_3x_1^3) + c_2(a_0 + a_1x_2 + a_2x_2^2 + a_3x_2^3) = 2a_0 + \frac{2}{3}a_2$$

Coefficient of same will be equal in both sşde of the equation therefore:

From a_0: $c_1 + c_2 = 2$

From a_1: $c_1x_1 + c_2x_2 = 0$

From a_2: $c_1x_1^2 + c_2x_2^2 = \frac{2}{3}$

From a_3: $c_1x_1^3 + c_2x_2^3 = 0$

Solution of this system of equation yields:

Roots: $x_{1,2} = \pm 1/\sqrt{3}$ and Coefficients : $c_{1,2}=1$

The general solution of this problem can be defined with the Legendre polynomials. Carl Friedrich Gauss was the first to derive the Gauss–Legendre quadrature rule. Carl Gustav Jacob Jacobi discovered the connection between the quadrature rule and the orthogonal family of Legendre polynomials.

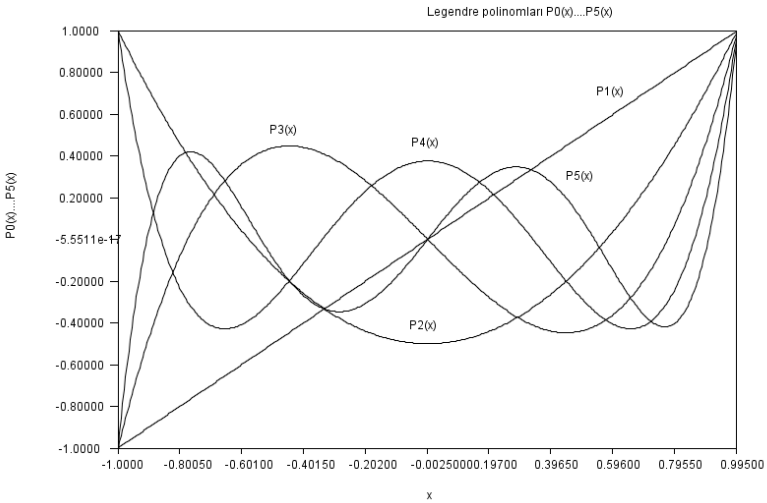
Johann Carl Fredrich Gauss	Carl Gustav Jacob Jacobi	Adrien-Marie Legendre

Legendre polynomials has a general definition as:

$$(k+1)P_{k+1}(x) = (2k+1)P_k(x) - kP_{k-1}(x), \quad k \geq 1$$

Where $P_0(x) = 1$ and $P_1(x) = x$

The first 6 Legendre polynomials are giving in graphic form.



The roots of Legendre Polynomials are the roots of the Gauss-Legendre integration Formula. For example if $P_2(x)$ value is calculated from the above general form:

$$P_2(x) = \frac{3x^2 - 1}{2}$$

The root of this is equal to $x_{1,2} = \mp\frac{1}{\sqrt{3}}$. In general roots can be calculated from Legendre polynomials as follows:

$$\Theta_{n,k} = \frac{n - k + 3/4}{n + 1/2}\pi$$

$$x_{n,k} = \left[1 - \frac{1}{8n^2} + \frac{1}{8n^3} - \frac{1}{384n^4}\left(39 - \frac{28}{sin^2\Theta_{n,k}}\right)\right] cos\Theta_{n,k} + E(n^{-5})$$

This equation contains an error level of n^{-5}. In order to decrease the error Newton-Raphson root finding method can be usefull. For detais of Newton-Raphson method please look at the related chapter.

$$x_{\nu+1} = x_\nu - \frac{1 - x_\nu^2}{P_{n-1}(x_\nu) - P_n(x_\nu)}$$

After finding the roots coeficients can be calculated as

$$c_1 x_1^{2j} + c_2 x_2^{2j} + \cdots + c_n x_n^{2j} = \frac{2}{2j + 1} \qquad 0 \le j \le n$$

Or

$$c_k = \frac{21 - x_k^2}{[nP_{n-1}(x_k)]^2}$$

As a last concept, consider that Gauss-Legendre integration limits are -1 and 1
Region x=[-1,1] can be converted to z=[a,b] by changing the variables
Curve fitting:

Variable	Value1	Value2
x	-1	1
z	a	b

$$z = f(x) = d_0 + d_1 x$$

$$a = d_0 + d_1(-1)$$

$$b = d_0 + d_1(1)$$

$$d_0 = \frac{a + b}{2}$$

$$a = d_0 + d_1(-1)$$

$$(-1)b = (-1)[d_0 + d_1(1)]$$

$$d_1 = \frac{b-a}{2}$$

$$z = f(x) = d_0 + d_1 x$$

$$dz = d_1 dx$$

$$I(-1,1) = \int_{-1}^{1} f(x)dx \cong \sum_{k=1}^{n} c_k f(x_k)$$

$$I(a,b) = \int_{a}^{b} f(z)dz = \int_{-1}^{1} d_1 f(d_0 + d_1 x)dx \quad \cong d_1 \sum_{k=1}^{n} c_k f(d_0 + d_1 x_k)$$

$$I(a,b) \cong \frac{b-a}{2} \sum_{k=1}^{n} c_k f\left(\left(\frac{a+b}{2}\right) + \left(\frac{b-a}{2}\right) x_k\right)$$

First 10 Gauss-Legendre integration formulas are given in the table

Table Gauss – Legendre integration roots and coefficients

N	x_k	c_k
2	-0.577350269189625	1.000000000000000
	0.577350269189625	1.000000000000000
3	-0.774596669241483	0.555555555555552
	0.000000000000000	0.888888888888888
	0.774596669241483	0.555555555555552
4	-0.861136311594052	0.347854845137447
	-0.339981043584856	0.652145154862546
	0.339981043584856	0.652145154862546
	0.861136311594052	0.347854845137447
5	-0.906179845938664	0.236926885056181
	-0.538469310105683	0.478628670499366
	0.000000000000000	0.568888888888888
	0.538469310105683	0.478628670499366
	0.906179845938664	0.236926885056181
6	-0.932469514203152	0.171324492379162
	-0.661209386466264	0.360761573048138
	-0.238619186083196	0.467913934572691
	0.238619186083196	0.467913934572691
	0.661209386466264	0.360761573048138
	0.932469514203152	0.171324492379162
7	-0.949107912342758	0.129484966168862
	-0.741531185599394	0.279705391489276
	-0.405845151377397	0.381830050505119
	0.000000000000000	0.417959183673469
	0.405845151377397	0.381830050505119
	0.741531185599394	0.279705391489276
	0.949107912342758	0.129484966168862
8	-0.960289856497536	0.101228536290369
	-0.796666477413626	0.222381034453374
	-0.525532409916329	0.313706645877887
	-0.183434642495649	0.362683783378362
	0.183434642495649	0.362683783378362
	0.525532409916329	0.313706645877887
	0.796666477413626	0.222381034453374
	0.960289856497536	0.101228536290369
9	-0.968160239507626	0.081274388361569
	-0.836031107326635	0.180648160694857
	-0.613371432700590	0.260610696402935
	-0.324253423403808	0.312347077040002

	0.000000000000000	**0.330239355001259**
	0.324253423403808	**0.312347077040002**
	0.613371432700590	**0.260610696402935**
	0.836031107326635	**0.180648160694857**
	0.968160239507626	**0.081274388361569**
10	**-0.973906528517171**	**0.066671344308684**
	-0.865063366688984	**0.149451349150580**
	-0.679409568299024	**0.219086362515982**
	-0.433395394129247	**0.269266719309996**
	-0.148874338981631	**0.295524224714752**
	0.148874338981631	**0.295524224714752**
	0.433395394129247	**0.269266719309996**
	0.679409568299024	**0.219086362515982**
	0.865063366688984	**0.149451349150580**
	0.973906528517171	**0.066671344308684**

EXAMPLE : Calculate integral of $f(x) = \frac{\pi}{4}\sqrt{1-x^2}$ function between limits 0 and 1 with three points Gauss-Legendre integration formula by hand and by computer program

For n=3 α=(1-0)/2=0.5 β=(1+0)/2=0.5

I=0.5*(0.555555555555553 *f(0.5*-0.774596669241483+0.5)+ 0.888888888888889*f(0.5*0+0.5)+ 0.555555555555553 *f(0.5*0.774596669241483+0.5))
I=1.004609
excel versions of the same integral

For n=4
Gauss Integral Formula

a		0.0000000000
b		1.0000000000
a	(b-a)/2	0.5000000000
b	(b+a)/2	0.5000000000

rk	**Ck**	**xk=a+b*rk**	**yk=f(xk)**	**z=a*Ck*yk**
-0.861136312	**0.347854845**	0.0694318442	1.2701668325	0.2209168434
-0.339981044	**0.652145155**	0.3300094782	1.2019096333	0.3919097720
0.339981044	**0.652145155**	0.6699905218	0.9452143982	0.3082084950
0.861136312	**0.347854845**	0.9305681558	0.4661568155	0.0810774534
			Integral	1.002112564

For n=6
Gauss Integral Formula

a		0.0000000000
b		1.0000000000
a	(b-a)/2	0.5000000000
b	(b+a)/2	0.5000000000

rk	**Ck**	**xk=a+b*rk**	**yk=f(xk)**	**z=a*Ck*yk**
-0.932469514	**0.171324492**	0.0337652429	1.2725135329	0.1090063675
-0.661209386	**0.360761573**	0.1693953068	1.2548389257	0.2263488324
-0.238619186	**0.467913935**	0.3806904070	1.1773678593	0.2754534138
0.238619186	**0.467913935**	0.6193095930	0.9996800599	0.2338821151
0.661209386	**0.360761573**	0.8306046932	0.7090193556	0.1278934690
0.932469514	**0.171324492**	0.9662347571	0.3280671473	0.0281029687
			Integral	1.000687166

For n=10
Gauss Integral Formula

a		0.0000000000
b		1.0000000000
a	(b-a)/2	0.5000000000
b	(b+a)/2	0.5000000000

rk	**Ck**	**xk=a+b*rk**	**yk=f(xk)**	**z=a*Ck*yk**

-0.973906529	**0.066671344**	0.0130467357	1.2731311764	0.0424406835
-0.865063367	**0.149451349**	0.0674683167	1.2703383629	0.0949268911
-0.679409568	**0.219086363**	0.1602952159	1.2567754345	0.1376711792
-0.433395394	**0.269266719**	0.2833023029	1.2210757598	0.1643975319
-0.148874339	**0.295524225**	0.4255628305	1.1521912348	0.1702502107
0.148874339	**0.295524225**	0.5744371695	1.0422085462	0.1539989363
0.433395394	**0.269266719**	0.7166976971	0.8879368571	0.1195459222
0.679409568	**0.219086363**	0.8397047841	0.6914240280	0.0757407876
0.865063367	**0.149451349**	0.9325316833	0.4597517249	0.0343552578
0.973906529	**0.066671344**	0.9869532643	0.2050004799	0.0068338288
			Integral	1.000161229

For n=60

$$f(x) = \frac{\pi}{4}\sqrt{1 - x^2}$$

Gauss Integral Formula

a		0.0000000000
b		1.0000000000
a	(b-a)/2	0.5000000000
b	(b+a)/2	0.5000000000

rk	**Ck**	**xk=a+b*rk**	**yk=f(xk)**	**z=a*Ck*yk**
-0.999210123227436	0.002026811968873	0.0003949384	1.2732394454	0.0012903085
-0.995840525118838	0.004712729926953	0.0020797374	1.2732367912	0.0030002106
-0.989787895222221	0.007389931163346	0.0051060524	1.2732229468	0.0047045150
-0.981067201752598	0.010047557182288	0.0094663991	1.2731824942	0.0063961870
-0.969701788765052	0.012678166476816	0.0151491056	1.2730934350	0.0080702453
-0.955722255839996	0.015274618596784	0.0221388721	1.2729274803	0.0097217409
-0.939166276116423	0.017829901014207	0.0304168619	1.2726504171	0.0113456155
-0.920078476177627	0.020337120729457	0.0399607619	1.2722225443	0.0129366717
-0.898510310810046	0.022789516943998	0.0507448446	1.2715991671	0.0144895654
-0.874519922646898	0.025180477621521	0.0627400387	1.2707311395	0.0159988085
-0.848171984785929	0.027503556749925	0.0759140076	1.2695654443	0.0174587826
-0.819537526162145	0.029752491500789	0.0902312369	1.2680457997	0.0188637609
-0.788693739932264	0.031921219019296	0.1056531300	1.2661132804	0.0202079397
-0.755723775306585	0.034003892724946	0.1221381123	1.2637069454	0.0214854777
-0.720716513355730	0.035994898051084	0.1396417433	1.2607644611	0.0226905441
-0.683766327381355	0.037888867569243	0.1581168363	1.2572227133	0.0238173724
-0.644972828489477	0.039680695452381	0.1775135858	1.2530184000	0.0248603208
-0.604440597048510	0.041365551235585	0.1977797015	1.2480886002	0.0258139365
-0.562278900753944	0.042938892835936	0.2188605496	1.2423713133	0.0266730243
-0.518601400058569	0.044396478795787	0.2406993000	1.2358059656	0.0274327167
-0.473525841761707	0.045734379716114	0.2632370791	1.2283338822	0.0280885441
-0.427173741583078	0.046948988848912	0.2864131292	1.2198987211	0.0286365057
-0.379670056576798	0.048037031819971	0.3101649717	1.2104468712	0.0290731374
-0.331142848268448	0.048995575455757	0.3344285759	1.1999278109	0.0293955768
-0.281722937423261	0.049822035690550	0.3591385313	1.1882944300	0.0296016238
-0.231543551376029	0.050514184532509	0.3842282243	1.1755033149	0.0296897957
-0.180739964873425	0.051070156069856	0.4096300176	1.1615149982	0.0296593761
-0.129449135396945	0.051488451500981	0.4352754323	1.1462941736	0.0295104560
-0.077809333949537	0.051767943174910	0.4610953330	1.1298098787	0.0292439668
-0.025959772301248	0.051907877631221	0.4870201138	1.1120356457	0.0288617051
0.025959772301248	0.051907877631221	0.5129798862	1.0929496235	0.0283663477
0.077809333949537	0.051767943174910	0.5389046670	1.0725346712	0.0277614570
0.129449135396945	0.051488451500981	0.5647245677	1.0507784251	0.0270514770
0.180739964873425	0.051070156069856	0.5903699824	1.0276733412	0.0262417190
0.231543551376029	0.050514184532509	0.6157717757	1.0032167145	0.0253383371
0.281722937423261	0.049822035690550	0.6408614687	0.9774106755	0.0243482948
0.331142848268448	0.048995575455757	0.6655714241	0.9502621669	0.0232793209
0.379670056576798	0.048037031819971	0.6898350283	0.9217829005	0.0221398573
0.427173741583078	0.046948988848912	0.7135868708	0.8919892958	0.0209389978
0.473525841761707	0.045734379716114	0.7367629209	0.8609024020	0.0196864187

0.518601400058569	0.044396478795787	0.7593007000	0.8285478029	0.0183923025
0.562278900753944	0.042938892835936	0.7811394504	0.7949555081	0.0170672547
0.604440597048510	0.041365551235585	0.8022202985	0.7601598294	0.0157222152
0.644972828489477	0.039680695452381	0.8224864142	0.7241992435	0.0143683648
0.683766327381355	0.037888867569243	0.8418831637	0.6871162440	0.0130170282
0.720716513355730	0.035994898051084	0.8603582567	0.6489571814	0.0116795738
0.755723775306585	0.034003892724946	0.8778618877	0.6097720929	0.0103673124
0.788693739932264	0.031921219019296	0.8943468700	0.5696145247	0.0090913950
0.819537526162145	0.029752491500789	0.9097687631	0.5285413466	0.0078627110
0.848171984785929	0.027503556749925	0.9240859924	0.4866125631	0.0066917881
0.874519922646898	0.025180477621521	0.9372599613	0.4438911240	0.0055886953
0.898510310810046	0.022789516943998	0.9492551554	0.4004427432	0.0045629483
0.920078476177627	0.020337120729457	0.9600392381	0.3563357382	0.0036234215
0.939166276116423	0.017829901014207	0.9695831381	0.3116409221	0.0027782634
0.955722255839996	0.015274618596784	0.9778611279	0.2664316177	0.0020348207
0.969701788765052	0.012678166476816	0.9848508944	0.2207839805	0.0013995680
0.981067201752598	0.010047557182288	0.9905336009	0.1747782074	0.0008780470
0.989787895222221	0.007389931163346	0.9948939476	0.1285028199	0.0004748135
0.995840525118838	0.004712729926953	0.9979202626	0.0820735936	0.0001933953
0.999210123227436	0.002026811968873	0.9996050616	0.0357805413	0.0000362602
			Integral	1.000000837

Python code:

```
from math import *

def I(x):
   y=x*x*exp(x)-2.0*x*exp(x)+2.0*exp(x)
   return y
def integral(f,0a,b):
    c=[0.236926885056181,0.478628670499366,0.568888888888888,0.478628670499366,0.236926885056181]
    x=[-0.906179845938664,-0.538469310105683,0.000000000000000,0.538469310105683,0.906179845938664]
    d0=(a+b)/2.0
    d1=(b-a)/2.0
    n=len(c)
    I=0
    for k in range(n):
       I+=d1*c[k]*f(d0+d1*x[k])
    return I

f = lambda x:x*x*exp(x)
a=float(input("a : "))
b=float(input("b : "))
I1=integral(f,a,b)
I2=I(b)-I(a)
print("I1  : ",I1,"\nI2 : ",I2);
```

```
runfile('E:/okul/SCO1/integral_GL1.py', wdir='E:/okul/SCO1')
a : 0
b : 1
I1  :  0.718281828393345
I2 :  0.7182818284590451
```

Excel version:

Gauss-Legendre integral

a	0		d0	0.5
b	1		d1	0.5

c	x	d0+d1*x	f(d0+d1*x)	d1*c*f(d0+d1*x)
0.23692689	**-0.90617985**	0.04691008	0.002306243	0.000273205
0.47862867	**-0.53846931**	0.23076534	0.067075095	0.016052032
0.56888889	**0**	0.5	0.412180318	0.117242401
0.47862867	**0.53846931**	0.76923466	1.277003387	0.305605217

	0.23692689	**0.906179846**	0.95308992	2.356076838	0.279108973
Integral					**0.718281828**

$$I = \int cos^2(x)dx$$

$$cos^2(x) + sin^2(x) = 1$$

$$cos^2(x) - sin^2(x) = \cos(2x)$$

$$2cos^2(x) = 1 + \cos(2x)$$

$$cos^2(x) = \frac{1 + \cos(2x)}{2}$$

$$I = \int cos^2(x)dx = \int \frac{1 + \cos(2x)}{2} dx = \frac{x}{2} + \frac{\sin(2x)}{4}$$

```
 from math import *

def I(x):
   y=x/2.0+sin(2*x)/4.0
   return y

def integral(f,a,b):
    c=[0.236926885056181,0.478628670499366,0.568888888888888,0.478628670499366,0.236926885056181]
    x=[-0.906179845938664,-0.538469310105683,0.000000000000000,0.538469310105683,0.906179845938664]
    d0=(a+b)/2.0
    d1=(b-a)/2.0
    n=len(c)
    I=0
    for k in range(n):
       I+=d1*c[k]*f(d0+d1*x[k])
    return I

f = lambda x:cos(x)*cos(x)
a=0
b=pi
I1=integral(f,a,b)
I2=I(b)-I(a)
print("I1  : ",I1,"\nI2 : ",I2);
```

```
runfile('E:/okul/SCO1/integral_GL2.py', wdir='E:/okul/SCO1')
I1  :  1.5707479410585858
I2 :  1.5707963267948966
```

Excel:

Gauss-Legendre integral

a	0		d0	1.570796327
b	3.141592654		d1	1.570796327

c	x	d0+d1*x	f(d0+d1*x)	d1*c*f(d0+d1*x)
0.23692689	**-0.90617985**	0.14737235	0.978438168	0.364139346
0.47862867	**-0.53846931**	0.72497071	0.560280459	0.421234625
0.56888889	**0**	1.57079633	3.75247E-33	3.35324E-33
0.47862867	**0.53846931**	2.41662194	0.560280459	0.421234625
0.23692689	**0.906179846**	2.9942203	0.978438168	0.364139346
Integral				**1.570747941**

In the next code function is defined as a lambda variable and Gauss-Legendre coefficients are calculated from Legendre polynomial formulations.

```
 from math import *
```

```
from if_x import *
def gauss_legendre_coefficients(x1,x2,n):
    #calculates legendre gauss-coefficients as coefficients of the integral
    #for n terms
        eps=3e-15
        m=int((n+1.0)/2.0)
        xm=0.5*(x2+x1)
        xl=0.5*(x2-x1)
        nmax=100
        a=[[0.0 for i in range(n)] for j in range(2)]
        for i in range(1,m+1):
            z=cos(pi*((i-0.25)/(n+0.5)))
            for ii in range(nmax):
                #while abs(z-z1) > eps:
                p1=1.0;
                p2=0.0;
                for j in range(1,n+1):
                    p3=p2
                    p2=p1
                    p1=(float)(((2.0*j-1.0)*z*p2-(j-1.0)*p3)/j)
                pp=(float)(n*(z*p1-p2)/(z*z-1.0))
                z1=z;
                z=z1-p1/pp
                if abs(z-z1) < eps: break
            a[0][i-1]=xm-xl*z
            a[0][n-i]=xm+xl*z
            a[1][i-1]=2.0*xl/((1.0-z*z)*pp*pp)
            a[1][n-i]=a[1][i-1]
        return a

def integral(f,x1,x2,n):
        #integral func(x)dx
        #integral of a function by using gauss-legendre quadrature
        #between x1 and x2
        a=gauss_legendre_coefficients(x1,x2,n)
        z=0
        for i in range(n):
            z=z+a[1][i]*f.func(a[0][i])
        return z

class f1(if_x):func=lambda self,x: cos(x)*cos(x)
f=f1()
#f = lambda x:cos(x)*cos(x)
a=0
b=pi
I1=integral(f,a,b,10)
print("I1  : ",I1);
```

```
runfile('E:/okul/SCO1/integral_GL.py', wdir='E:/okul/SCO1')
Reloaded modules: if_x
I1  :  1.57079632679489
```

Built-in Gauss-Legendre integration in python

```
import scipy.integrate as integrate
import scipy.special as special
from math import *
f = lambda x: 4.0/pi*sqrt(1.0-x*x)
I = integrate.quad(f, 0, 1.0)
print("Integral = ",I)
```

```
runfile('C:/Users/meral/.spyder-py3/autosave/integral_scripy.py', wdir='C:/Users/meral/.spyder-py3/autosave')
Integral =  (1.0000000000000004, 1.1247469622333028e-10)
```

Gauss-Legendre integration using built-in roots_legendre function

```
# -*- coding: utf-8 -*-
"""
Created on Sun Jan  9 08:00:19 2022

@author: Mustafa Turhan Coban
Roots of Legendre Polynomial
```

```
Using scripy special function roots_legendre
"""
from scipy.special import *
from math import *

def I(x):
   y=x/2.0+sin(2*x)/4.0
   return y

def integral(f,a,b,n):
    [x,c]=roots_legendre(n)
    print("x = \n",x)
    print("y = \n",x)
    d0=(a+b)/2.0
    d1=(b-a)/2.0
    I=0
    for k in range(n):
       I+=d1*c[k]*f(d0+d1*x[k])
    return I

n=5
f = lambda x:cos(x)*cos(x)
a=0
b=pi
I1=integral(f,a,b,n)
I2=I(b)-I(a)
print("I1  : ",I1,"\nI2 : ",I2);
```

```
runfile('D:/okul/SCO1/integral_GL_scirpy_special.py', wdir='D:/okul/SCO1')
x =
 [-0.90617985 -0.53846931  0.         0.53846931  0.90617985]
y =
 [-0.90617985 -0.53846931  0.         0.53846931  0.90617985]
I1  :  1.5707479410586112
I2 :  1.5707963267948966
```

```
n=10
f = lambda x:cos(x)*cos(x)
a=0
b=pi
I1=integral(f,a,b,n)
I2=I(b)-I(a)
print("I1  : ",I1,"\nI2 : ",I2);
```

```
runfile('D:/okul/SCO1/integral_GL_scirpy_special.py', wdir='D:/okul/SCO1')
x =
 [-0.97390653 -0.86506337 -0.67940957 -0.43339539 -0.14887434  0.14887434
  0.43339539  0.67940957  0.86506337  0.97390653]
y =
 [-0.97390653 -0.86506337 -0.67940957 -0.43339539 -0.14887434  0.14887434
  0.43339539  0.67940957  0.86506337  0.97390653]
I1  :  1.5707963267949017
I2 :  1.5707963267948966
```

In the Gauss-Legendre integral formulation weight function was taken as 1.

$$I(-1,1) = \int_{-1}^{1} f(x)w(x)dx \cong \sum_{k=1}^{n} c_k f(x_k)$$

If the weight function is taken as: $w(x) = \frac{1}{\sqrt{1-x^2}}$ **Gauss-Chebychev** integration formula is created. This equation has the form of

$$I(-1,1) = \int_{-1}^{1} f(x)w(x)dx = \int_{-1}^{1} f(x)w(x)dx \cong \sum_{k=1}^{n} \frac{1}{\sqrt{1-x^2}} c_k f(x_k)$$

x_k in the equation is the roots of chebcychev polynomials. General definition of the Chbychev polynomials are given as:

$P_0(x)=1$

$P_1(x)=x$

$P_2(x)=2x^2-1$

$P_n(x)=2xP_{n-1}(x) - P_{n-2}(x)$, n=3,4,5..

It should be note that Chebychev polinomials are orthogonal polnomials. The roots of the polynomials are defined as:

$$x_k = cos\left(\frac{(k+1/2)\pi}{n}\right) \qquad k = 0,1,2,,\dots,n-1$$

Coeeficients of the Formula can be calculated as:

$$c_k = \frac{\pi}{n} \quad k = 0,1,2,,\dots,n-1$$

x=[-1,1] integration range in Gauss-Chebychev Formula can be change to more general z=[a,b] range by using similar formulations used in Gauss-Legendre formulations.

$$k_1 = \frac{x_2 - x_1}{2}$$

$$k_2 = \frac{x_2 + x_1}{2}$$

$$x = k_1 * x_k + k_2$$

$$I(a,b) = \int_a^b f(x)w(x)dx = \int_{-1}^{1} f(x)w(x)dx \cong \sum_{k=1}^{n} \frac{k_1}{\sqrt{1-x^2}} c_k f(k_1 * x_k + k_2)$$

As an example program if following integral is to be solved:

```
from math import *
from if_x import *


class f1(if_x):
  def func(self,x):
      y1=self.w(x)*exp(x)
      return y1;
  def w(self,x):
      w1=sqrt(1.0-x*x)
      return w1


def gauss_chebychev_coefficients(n):
    #Gauss-Chebychev quadratürü katsayilarini hesaplar
    sgngam=1
    a=[[0.0 for i in range(n)] for j in range(2)]
    for j in range(n+1):
        k=j-1
        a[0][k]=cos((2.0*j-1.0)*pi/(2.0*n))
        a[1][k]=pi/float(n)
    return a

def gauss_chebychev_integral(f_xnt,x1,x2,n):
    # n : number of integral coefficients
    # this routine first generates gauss chebychev coefficients
    # for [xa,xb] band
    # then calculates gauss chebychev integral
    k1=(x2-x1)/2.0
    k2=(x2+x1)/2.0
    a=[[0.0 for i in range(n)] for j in range(2)]
    a=gauss_chebychev_coefficients(n)
    toplam=0
```

```
    z1=0
    for k in range(n):
        x=k1*a[0][k]+k2
        toplam=toplam+k1*a[1][k]*f_xnt.func(x)
    return toplam

f=f1()
I=gauss_chebychev_integral(f,0.0,1.0,50)
print(I)
```

```
runfile('E:/okul/SCO1/gauss_chebychev_integral.py', wdir='E:/okul/SCO1')
Reloaded modules: if_x
1.7379087256915644
```

Gauss-Hermit formula : this formulation uses the weight function and calculates integrals from minus infinity to plus infinity.

$$w(x) = e^{-x^2}$$

$$I_w(-\infty,\infty) \cong \int_{-\infty}^{\infty} w(x)f(x)dx \cong \int_{-\infty}^{\infty} e^{-x^2} f(x)dx \cong \sum_{k=1}^{n} c_k f(x_k)$$

In this equation x_k is the roots of the Hermite polynomials. Hermit Polinomials defined as:

$P_{-1}(x)=0$

$P_0(x)=1$

$P_1(x)=2x$

$P_2(x)=4x^2-2$

$P_3(x)=8x^3-12x$

$P_n(x)=2xP_{n-1}(x) - 2(n-1)P_{n-2}(x)$, n=1,2,3,4,5..

Coefficients are given as:

$$c_k = \frac{2^{n-1} n! \sqrt{\pi}}{n^2 P_{n-1}(x_k)^2}$$

Coefficients for the first 7 sets are given in the following table

Table 7.3-6 Gauss –Hermit integration roots and coefficients

N	x_k	c_k
2	**+/-0.707167811**	**0.886226926**
3	**0**	**1.181635901**
	+/- 1.224744871391589	**0.295408975**
4	**+/- 0.524647623275290**	**0.80491409**
	+/- 1.650680123885784	**0.081312835**
5	**0**	**0.945308721**
	+/-0.9585724646	**0.393619323**
	+/-2.0201828705	**0.019953242**
6	**+/- 2.350604973674492**	**0.00453001**
	+/- 1.335849074013697	**0.15706732**
	+/- 0.436077411927616	**0.724629595**
7	**+/- 2.651961356835233**	**0.000971781**
	+/- 1.673551628767471	**0.054515583**
	+/- 0.816287882858965	**0.425607253**
	0	**0.810264618**

Gauss_Hermite integral equation example code :

Integration function

$$I = \int_{-\infty}^{\infty} \frac{dx}{1+x^2}$$

```
from math import *
```

```
from if_x import *


class f1(if_x):
  def func(self,x):
      y1=self.w(x)*1.0/(1+x*x)
      return y1;
  def w(self,x):
      w1=exp(x*x)
      return w1
def gauss_hermite_coefficients(n):
    a=[[0.0 for i in range(n)] for j in range(2)]
    r=0
    eps=1e-10;
    pipm4 = pow(pi, -0.25)
    nn=int((n+2)/2)
    for i in range(nn):
        if i==0:
            r =sqrt(float(2.0*n+1.0))-1.85575*pow(float(2.0*n+1.0), -1.0/6.0)
        elif i==1:
            r = r-1.14*pow(float(n), 0.426)/r
        elif i==2:
            r = 1.86*r-0.86*a[0][0]
        elif i==3:
            r = 1.91*r-0.91*a[0][1]
        else:
            r = 2*r-a[0][i-2]
        p2 = 0.0
        p3 = pipm4
        j1=0
        for j in range(n):
            p1 = p2
            p2 = p3
            p3 = p2*r*sqrt(2.0/float(j+1))-p1*sqrt(float(j)/float(j+1));
            j1=j
        j=j1+1
        dp3 = sqrt(float(2.0*j))*p2
        r1=r
        r = r-p3/dp3
        while abs(r-r1)>=eps*(1+abs(r))*100.0:
             p2 = 0.0
             p3 = pipm4
             for j in range(n):
                 p1 = p2
                 p2 = p3
                 p3 = p2*r*sqrt(2.0/float(j+1))-p1*sqrt(float(j)/float(j+1));
                 j1=j
             j=j1+1
             dp3 = sqrt(float(2.0*j))*p2
             r1=r
             r = r-p3/dp3
        a[0][i] = r
        a[1][i] = 2.0/(dp3*dp3);
        a[0][n-1-i] = -a[0][i];
        a[1][n-1-i] = a[1][i];
    return a

def gauss_hermite_integral(f_xnt,n):
    # n : number of integral coefficients
    # this routine first generates gauss hermite coefficients
    # for [x1,x2] band
    # then calculates gauss hermite integral
    a=[[0.0 for i in range(n)] for j in range(2)]
    a=gauss_hermite_coefficients(n)
    print("a=\n",a)
    z=0
    x1=0
    for i in range(n):
        x1=f_xnt.func(a[0][i])
        z=z+a[1][i]*x1
    return z


f=f1()
print("integral  : ",gauss_hermite_integral(f,7));
```

```
runfile('E:/okul/SCO1/gauss_hermite_integral.py', wdir='E:/okul/SCO1')
Reloaded modules: if_x
a=
 [[2.6519613568352334, 1.6735516287674717, 0.8162878828589646, -0.0, -0.8162878828589646, -1.6735516287674717, -
2.6519613568352334], [0.0009717812123401424, 0.05451557982996558, 0.42560725261012705, 0.8102646175568071,
0.42560725261012705, 0.05451557982996558, 0.0009717812123401424]]
integral  :  2.5512038801916415

In [81]:
```

```
f=f1()
print("integral  : ",gauss_hermite_integral(f,20));
```

```
runfile('E:/okul/SCO1/gauss_hermite_integral.py', wdir='E:/okul/SCO1')
Reloaded modules: if_x
a=
 [[5.387480890011233, 4.603682449550744, 3.944764040115625, 3.3478545673832163, 2.788806058428131, 2.2549740020892757,
1.7385377121165861, 1.234076215395323, 0.7374737285453943, 0.24534070830090127, -0.24534070830090127, -
0.7374737285453943, -1.234076215395323, -1.7385377121165861, -2.2549740020892757, -2.788806058428131, -
3.3478545673832163, -3.944764040115625, -4.603682449550744, -5.387480890011233], [2.2293936242484147e-13,
4.3993406240405176e-10, 1.0860693697609841e-07, 7.802556261260467e-06, 0.0002283385990645384 2, 0.0032437733393699776,
0.02481052088734086, 0.10901720602002156, 0.28667550368103384, 0.46224366960058466, 0.46224366960058466,
0.28667550368103384, 0.10901720602002156, 0.02481052088734086, 0.0032437733393699776, 0.0002283385990645384 2,
7.802556261260467e-06, 1.0860693697609841e-07, 4.3993406240405176e-10, 2.2293936242484147e-13]]
integral  :  2.80468330172924
```

```
f=f1()
print("integral  : ",gauss_hermite_integral(f,20));
```

```
runfile('E:/okul/SCO1/gauss_hermite_integral.py', wdir='E:/okul/SCO1')
Reloaded modules: if_x
a=
 [[13.40648733814491, 12.823799749487808, 12.342964222859674, 11.915061943114166, 11.521415400787031,
11.152404385585125, 10.802260753684715, 10.467185421342883, 10.144509941292847, 9.832269807777969, 9.528965823390115,
9.233420890219161, 8.944689217325475, 8.661996168134518, 8.384696940416266, 8.112247311162792, 7.844182384460821,
7.580100807857489, 7.319652822304577, 7.062531060248881, 6.808463352858802, 6.557207031921542, 6.308544361112136,
6.062278832614303, 5.818232135203517, 5.5762416493299245, 5.33615836013836, 5.097845105089136, 4.86117509179121,
4.6260306357871555, 4.392302078682684, 4.15988685513103, 3.928688683427671, 3.698616859318492, 3.4695856364185893,
3.241513679631013, 3.0143235803311557, 2.7879414239819895, 2.562296402372608, 2.3373204639068783, 2.112947996371188,
1.8891155374270083, 1.6657615087415094, 1.4428259702159327, 1.220250391218953, 0.9979774360981053, 0.7759507615401458,
0.554114823591617, 0.3324146923422318, 0.1107958724224395, -0.1107958724224395, -0.3324146923422318, -0.554114823591617,
-0.7759507615401458, -0.9979774360981053, -1.220250391218953, -1.4428259702159327, -1.6657615087415094, -
1.8891155374270083, -2.112947996371188, -2.3373204639068783, -2.562296402372608, -2.7879414239819895, -
3.0143235803311557, -3.241513679631013, -3.4695856364185893, -3.698616859318492, -3.928688683427671, -4.15988685513103, -
4.392302078682684, -4.6260306357871555, -4.86117509179121, -5.097845105089136, -5.33615836013836, -5.5762416493299245, -
5.818232135203517, -6.062278832614303, -6.308544361112136, -6.557207031921542, -6.808463352858802, -7.062531060248881, -
7.319652822304577, -7.580100807857489, -7.844182384460821, -8.112247311162792, -8.384696940416266, -8.661996168134518, -
8.944689217325475, -9.233420890219161, -9.528965823390115, -9.832269807777969, -10.144509941292847, -10.467185421342883,
-10.802260753684715, -11.152404385585125, -11.521415400787031, -11.915061943114166, -12.342964222859674, -
12.823799749487808, -13.40648733814491], [5.908067863103223e-79, 1.9728604274119031e-72, 3.083028963828961e-67,
9.019220432276765e-63, 8.518883076551092e-59, 3.459477733807211e-55, 7.191529461537315e-52, 8.597534439303944e-49,
6.42072208009955e-46, 3.185217604477932e-43, 1.100470663095527e-40, 2.748784872798951e-38, 5.1162325987743434e-36,
7.274572594631579e-34, 8.06743427795364e-32, 7.101812226172967e-30, 5.037791166163028e-28, 2.917350072619246e-26,
1.3948384373378411e-24, 5.561019523433332e-23, 1.8649961539456107e-21, 5.302313526665779e-20, 1.286832523068123e-18,
2.682491650786072e-17, 4.829834747689982e-16, 7.548896321650472e-15, 1.0288748904462765e-13, 1.2278784799174774e-12,
1.2879038035689307e-11, 1.191300622469033e-10, 9.74792119251736e-10, 7.075857257415599e-09, 4.568127498474821e-08,
2.629097480438831e-07, 1.3517971581501169e-06, 6.221524815349301e-06, 2.5676159379201832e-05, 9.517162777530007e-05,
0.00031729197102648603, 0.0009526921885249078, 0.0025792732600307335, 0.006303000285579659, 0.013915665220208456,
0.027779127385918166, 0.05017581267742089, 0.08205182739122158, 0.12153798684410347, 0.16313003050278288,
0.19846284859264707, 0.21889262957968678, 0.21889262957968678, 0.19846284859264707, 0.16313003050278288,
0.12153798684410347, 0.08205182739122158, 0.05017581267742089, 0.027779127385918166, 0.013915665220208456,
0.006303000285579659, 0.0025792732600307335, 0.0009526921885249078, 0.00031729197102648603, 9.517162777530007e-05,
2.5676159379201832e-05, 6.221524815349301e-06, 1.3517971581501169e-06, 2.629097480438831e-07, 4.568127498474821e-08,
7.075857257415599e-09, 9.74792119251736e-10, 1.191300622469033e-10, 1.2879038035689307e-11, 1.2278784799174774e-12,
1.0288748904462765e-13, 7.548896321650472e-15, 4.829834747689982e-16, 2.682491650786072e-17, 1.286832523068123e-18,
5.302313526665779e-20, 1.8649961539456107e-21, 5.561019523433332e-23, 1.3948384373378411e-24, 2.917350072619246e-26,
5.037791166163028e-28, 7.101812226172967e-30, 8.06743427795364e-32, 7.274572594631579e-34, 5.1162325987743434e-36,
2.748784872798951e-38, 1.100470663095527e-40, 3.185217604477932e-43, 6.42072208009955e-46, 8.597534439303944e-49,
7.191529461537315e-52, 3.459477733807211e-55, 8.518883076551092e-59, 9.019220432276765e-63, 3.083028963828961e-67,
1.9728604274119031e-72, 5.908067863103223e-79]]
integral  :  2.9965945892152823
```

Gauss-Laguerre formulation has the weight factor in the form of

$$w(x) = x^{\alpha} e^{-x}$$

The integral limits of this formulation is in between o and infinity. The equation is in the form of

$$I_w(0,\infty) = \int_0^{\infty} f(x)w(x)dx = \int_0^{\infty} x^{\alpha} e^{-x} f(x)dx \cong \sum_{k=0}^{n-1} c_k f(x_k)$$

In this equation x_k is the root of Laguerre polinomials. Leguerre polynomial has the form of:

$P_{-1}(x)=0$

$P_0(x)=1$

$(n+1)P^{\alpha}_{n+1}(x)=(-x+2*n+\alpha+1)P^{\alpha}_{n}(x) - (n+\alpha)P^{\alpha}_{n-1}(x)$, n=1,2,3,4,5...

Coefficients,

$$c_k = \frac{1}{x_k \left[\frac{dP_n(x_k)}{dx_k}\right]^2}$$

For the first 5 values, roots and coefficient of the equation is given in **Table 8.4-3** (α=0), **Table 8.4-4** (α=1) and **Table 8.4-5** (α=2) .

Table 7.3-9 Gauss – Laquerre integral roots and coefficients (α=0)

N	x_k	c_k
2	0.585786437626905	0.853553390592534
	3.414213562373090	0.146446609406723
3	0.415774556783479	0.711093009915089
	2.294280360279040	0.278517733568676
	6.289945082937470	0.010389256497354
4	0.322547689619392	0.603154104288259
	1.745761101158340	0.357418692437802
	4.536620296921120	0.038887908507759
	9.395070912301130	0.000539294705556
5	0.263560319718141	0.521755610467045
	1.413403059106510	0.398666821407258
	3.596425771040720	0.075942449681708
	7.085810005858830	0.003611758663878
	12.640800844275700	0.000023369972386

Table 7.3-10 Gauss – Laquerre integral roots and coefficients (α=1)

N	x_k	c_k
2	1.267949192431120	0.788675133123411
	4.732050807568870	0.211324865405185
3	0.935822227524088	0.588681472855445
	3.305407289332270	0.391216059222280
	7.758770483143630	0.020102459732679
4	0.743291927981432	0.446870579513482
	2.571635007646270	0.477635774492073
	5.731178751689090	0.074177784726293
	10.953894312683100	0.001315849683447
5	0.617030853278271	0.348014523429866
	2.112965958578520	0.502280674138866
	4.610833151017530	0.140915919102187
	8.399066971204840	0.008719893025972

Table 7.3-11 Gauss – Laquerre integral roots and coefficients (α=2)

N	x_k	c_k
2	2.000000000000000	1.500000000000000
	6.000000000000000	0.500000000000000
3	1.517387080677410	1.037494961490390
	4.311583133719520	0.905750004703039
	9.171029785603060	0.056755033772202
4	1.226763263500300	0.725524997698604
	3.412507358696940	1.063424292391060
	6.902692605851610	0.206696130999709
	12.458036771951100	0.004354579188558
5	1.031109144093380	0.520917396835042
	2.837212823953820	1.066705933159050
	5.620294272598700	0.383549720007093
	9.682909837664020	0.028564233510280

	15.828473921690000	**0.000262712802303**

If we investigate Gauss-Leguerre Formula with an example. We will calculate the integral

$$I = \int_0^{\infty} \frac{x}{e^x + 1} dx$$ Exact integral value I= 0.822467.

```
 from math import *
from if_x import *
class f1(if_x):
    def __init__(self, alpha,xa):
      self.alpha=alpha
      self.xa=xa

    def func(self,x):
      y=x+self.xa
      y1=self.w(x)*x/(exp(x)+1)
      return y1;

    def w(self,x):
      if self.alpha==0:
          w1=exp(x)
      elif self.alpha==1:
          w1=exp(x)/x
      elif self.alpha==2:
          w1=exp(x)/(x*x)
      else:
          w1=exp(x)/pow(x,self.alpha)
      return w1

def gauss_laguerre_coefficients(n,alpha):
    #Gauss-Laguerre quadratürü katsayilarini hesaplar
    sgngam=1;
    a=[[0.0 for i in range(n)] for j in range(2)]
    r=0;
    eps=1.0e-10
    for i in range(n):
        if i==0:
            r = (1.0+alpha)*(3.0+0.92*alpha)/(1+2.4*n+1.8*alpha)
        elif i==1:
            r = r+(15.0+6.25*alpha)/(1.0+0.9*alpha+2.5*n)
        else:
            r = r+((1.0+2.55*(i-1))/(1.9*(i-1))+1.26*(i-1)*alpha/(1+3.5*(i-1)))/(1.0+0.3*alpha)*(r-a[0][i-2])
        #  do blok
        p2 = 0;
        p3 = 1;
        for j in range(n):
            p1 = p2
            p2 = p3
            p3 = ((-r+2*j+alpha+1)*p2-(j+alpha)*p1)/(j+1)
        dp3 = (n*p3-(n+alpha)*p2)/r
        r1 = r;
        r = r-p3/dp3
        while abs(r-r1)>=eps*(1+abs(r))*100:
            p2 = 0;
            p3 = 1;
            for j in range(n):
                p1 = p2
                p2 = p3
                p3 = ((-r+2*j+alpha+1)*p2-(j+alpha)*p1)/(j+1)
            dp3 = (n*p3-(n+alpha)*p2)/r
            r1 = r;
            r = r-p3/dp3
        a[0][i] = r
        lng=lgamma(float(n))
```

```
        lng1=lgamma(float(alpha+n))
        a[1][i] = -exp(lng1-lng)/(dp3*n*p2)
    return a

def  gauss_laquerre_integral(f_xnt,alpha,n):
    #a=[[0.0 for i in range(n)] for j in range(2)]
    a=gauss_laguerre_coefficients(n,alpha)
    print(a)
    z=0.0
    for i in range(n):
        z=z+a[1][i]*f_xnt.func(a[0][i])
    return z



alpha=0.0
xa=0.0;
exact=0.822467
f=f1(alpha,xa)
print("integral  : ",gauss_laquerre_integral(f,alpha,30),"exact integral = ",exact)
```

```
runfile('E:/okul/SCO1/gauss_leguerre_integral.py', wdir='E:/okul/SCO1')
Reloaded modules: if_x
[[0.047407180540804436, 0.2499239167531601, 0.6148334543927677, 1.1431958256661, 1.8364545546225732, 2.696521874557215,
3.7258145077795097, 4.927293765849882, 6.3045155909650745, 7.8616932933702595, 9.603775985479261, 11.536546597956141,
13.666744693064238, 16.002221188981068, 18.55213484014315, 21.327204321783128, 24.340035764532693, 27.60555479678096,
31.141586701111237, 34.96965200824907, 39.11608494906789, 43.61365290848486, 48.5039861638042, 53.841385406507506,
59.6991218592355, 66.18061779443849, 73.44123859555988, 81.73681050672769, 91.55646652253684, 104.1575244310589],
[0.11604408459540358, 0.22085107571058027, 0.24139979501275974, 0.19463676844642563, 0.1237284159740859,
0.06367878042225962, 0.026860475337671705, 0.009338070904071624, 0.00268069689343822, 0.0006351291219520559,
0.00012390745991046017, 1.9828788490041096e-05, 2.5893509659220805e-06, 2.740942840536074e-07, 2.3328311650249643e-08,
1.5807455747673273e-09, 8.4274791224058e-11, 3.485161232810341e-12, 1.0990180557315944e-13, 2.588312617200288e-15,
4.4378377037318635e-17, 5.365916649660166e-19, 4.3939468922898974e-21, 2.3114097943795043e-23, 7.27458849806314e-26,
1.239149701197934e-28, 9.832375073911014e-32, 2.842323546988629e-35, 1.8786080302787062e-39, 8.745980212838154e-45]]
integral  :  0.822467013985489 exact integral= 0.822467
```

```
alpha=0.0
xa=0.0;
exact=0.822467
f=f1(alpha,xa)
print("integral  : ",gauss_laquerre_integral(f,alpha,5),"exact integral = ",exact)
```

```
runfile('E:/okul/SCO1/gauss_leguerre_integral.py', wdir='E:/okul/SCO1')
Reloaded modules: if_x
[[0.2635603197181409, 1.413403059106517, 3.596425771040722, 7.085810005858837, 12.640800844275782], [0.5217556104670454,
0.398666821407258, 0.07594244968170755, 0.0036117586638776344, 2.3369972385775723e-05]]
integral  :  0.8226695527673321 exact integral= 0.822467
```

```
alpha=1.0
xa=0.0;
exact=0.822467
f=f1(alpha,xa)
print("integral  : ",gauss_laquerre_integral(f,alpha,5),"exact integral = ",exact)
```

```
runfile('E:/okul/SCO1/gauss_leguerre_integral.py', wdir='E:/okul/SCO1')
Reloaded modules: if_x
[[0.6170308532782706, 2.112965958578524, 4.610833151017532, 8.399066971204842, 14.26010306592083], [0.3480145234298671,
0.5022806741388668, 0.14091591910218698, 0.008719893025972364, 6.897332356299197e-05]]
integral  :  0.8224790352572539 exact integral= 0.822467
```

```
alpha=2.0
xa=0.0;
exact=0.822467
f=f1(alpha,xa)
print("integral  : ",gauss_laquerre_integral(f,alpha,5),"exact integral = ",exact)
```

```
runfile('E:/okul/SCO1/gauss_leguerre_integral.py', wdir='E:/okul/SCO1')
```

```
Reloaded modules: if_x
[[1.0311091440933815, 2.837212823953822, 5.620294272598708, 9.682909837664027, 15.828473921690062], [0.5209173968350423,
1.0667059331590585, 0.3835497200070941, 0.02856423351028029, 0.0002627128023032893]]
integral  :  0.7985241741742577 exact integral= 0.822467
```

Now let us look at integration:

$$I_w(x_a,\infty)=\int_{x_a}^{\infty} f(x)w(x)dx=\int_{x_a}^{\infty} x^{\alpha}e^{-x}f(x)dx\cong\sum_{k=0}^{n-1} c_k f(x_k)$$

The following coordinate system shift can be applied:

$y=x-x_a$ or $x=y+x_a$ Integration will be converted to

$$I_w(x_a,\infty)=\int_{0}^{\infty} f(y+x_a)w(y+x_a)dx=\int_{x_a}^{\infty} x^{\alpha}e^{-x}f(x+x_a)dx\cong\sum_{k=0}^{n-1} c_k f(x_k)$$

Consider function:

$$I(1,\infty)=\int_{1}^{\infty}\frac{sin\left(\frac{1}{x}\right)}{x^{3/2}}dx$$

$$x_a=1$$

```
alpha=0.0
xa=1.0;
f=f1(alpha,xa)
print("integral  : ",gauss_laquerre_integral(f,alpha,100))
```

```
runfile('E:/okul/SCO1/gauss_leguerre_integral.py', wdir='E:/okul/SCO1')
Reloaded modules: if_x
[[0.014386146995420105, 0.07580361202335732, 0.18631410205718538, 0.34596918099143037, 0.5548109375809116,
0.8128912841156704, 1.1202738350075412, 1.4770343299238287, 1.8832608263423956, 2.339053849646035, 2.8445265427553568,
3.399804827445709, 4.005027581758654, 4.66034683556891, 5.365927985585114, 6.121950030804022, 6.9286058293761705,
7.78610237786252, 8.694661113922166, 9.654518243555078, 10.665925094121675, 11.729148494472224, 12.844471183641028,
14.012192249694277, 15.232627600466701, 16.506110468081992, 17.832991949326388, 19.213641584136063, 20.64844797466835,
22.137819447656707, 23.682184763002375, 25.28199387183404, 26.937718727574264, 28.64985415389129, 30.418918773790974,
32.245456004520676, 34.13003512342152, 36.07325241037997, 38.075732373107094, 40.138129062115546, 42.2611274829848,
44.44544511431187, 46.69183354065154, 49.001080210772436, 51.37401033270399, 53.811488918355664, 56.31442299196172,
58.88376397828218, 61.520510288396146, 64.22571012310156, 67.00046451641931, 69.84593064455838, 72.76332542897458,
75.75392946593993, 78.81909131941147, 81.96023221906012, 85.1788512112189, 88.47653081739462, 91.85494326304931,
95.31585734883173, 98.86114604761353, 102.49279492391656, 106.21291148804681, 110.02373561603092, 113.92765118897162,
117.92719913257324, 122.02509207044163, 126.22423084475042, 130.52772320679952, 134.93890504022767, 139.46136455424073,
144.09896997721273, 148.85590139775826, 153.73668754797302, 158.7462485117131, 163.88994558258725, 169.17363981000304,
174.60376118237662, 180.1873909402457, 185.93236023966696, 191.84736937224832, 197.94213310214326, 204.2275595670305,
210.71597286157694, 217.42139327200147, 224.3598947888746, 231.5500680251725, 239.01362975131497, 246.77624096724867,
254.86862925704816, 263.328168469161, 272.20117002409256, 281.54632828389737, 291.4401336163771, 301.9858552516392,
313.3295340040755, 325.6912634370265, 339.4351019234496, 355.2613118885341, 374.9841128343427], [0.036392603866754164,
0.07967660316000183, 0.11211473945390009, 0.13035661296159656, 0.13404333969551574, 0.12540709075415185,
0.10831411251219196, 0.08709664472146864, 0.06555103174372773, 0.046340178472734334, 0.030846308627686937,
0.019367828113984967, 0.011485442360184952, 0.006438951001616868, 0.003414979989699636, 0.001714319740187997,
0.0008148715915919501, 0.0003668548366019691, 0.00015645207417942547, 6.32108705294959e-05, 2.419575226543846e-05,
8.774309763846465e-06, 3.0142674860306974e-06, 9.80833589943247e-07, 3.022638743555172e-07, 8.8200583953501e-08,
2.4364258562121572e-08, 6.369711373923579e-09, 1.5756003204633361e-09, 3.686329201351558e-10, 8.154798924231819e-11,
1.7050625568269007e-11, 3.3682141708675598e-12, 6.283524955367358e-13, 1.1064983897712306e-13, 1.8383505027476552e-14,
2.8801152959819496e-15, 4.2526130352121874e-16, 5.914432524715152e-17, 7.743089116683089e-18, 9.536249450237321e-19,
1.1040946284115464e-19, 1.2008469704557837e-20, 1.2260070077194504e-21, 1.174021713903432e-22, 1.0535940558744862e-23,
8.853231431142677e-25, 6.959197933876984e-26, 5.112369815465154e-27, 3.5062721488151923e-28, 2.2426462726895116e-29,
1.3362094234375714e-30, 7.407428957448482e-32, 3.8158601369670034e-33, 1.8242001974877066e-34, 8.081632334099159e-36,
3.313063800120945e-37, 1.2548329963224134e-38, 4.383790970655894e-40, 1.4101392233228778e-41, 4.168863967052771e-43,
1.1304818883525694e-44, 2.806037425516308e-46, 6.361270289295286e-48, 1.3139810964552342e-49, 2.466810378067207e-51,
4.197746138285721e-53, 6.456266885216199e-55, 8.947332294739816e-57, 1.113565993502358e-58, 1.2402333218532835e-60,
1.231376572606176e-62, 1.0853674489005896e-64, 8.454956490873427e-67, 5.792630756482023e-69, 3.4718547761782743e-71,
1.809859533362588e-73, 8.153752606431713e-76, 3.1524725471255963e-78, 1.0379005887285149e-80, 2.88487551622973e-83,
6.704801190897425e-86, 1.288963732139567e-88, 2.0248482977502184e-91, 2.5633920937868946e-94, 2.573961241939659e-97,
2.0126543532779845e-100, 1.1994838794060327e-103, 5.3121277504058205e-107, 1.6959659533568917e-110, 3.762082093544577e-
```

```
114, 5.53964175433192e-118, 5.110640476639482e-122, 2.7399654684224493e-126, 7.713611482180124e-131, 9.88249455581916e-136, 4.6468629521768924e-141, 5.626037204505862e-147, 8.90503138348086e-154, 3.2465651275863856e-162]]
integral  :  0.6204492430715238
```

4.5 CLENSHAW-CURTIS INTEGRATION

Clenshaw-Curtis formulation is based on $x = \cos(\theta)$ transformation. When this transformation applies, in the range of -1 to 1 integration equation becomes:

$$\int_{-1}^{1} f(x)dx = \int_{0}^{\pi} f(\cos(\theta))d(\cos(\theta)) = \int_{0}^{\pi} f(\cos(\theta))\sin(\theta)\,d\theta$$

Form. This integration can be opened as a series as:

$$f(\cos(\theta)) = \frac{a_0}{2} + \sum_{k=1}^{\infty} a_k \cos(k\theta)$$

By applying this transformation integral can be carried out as:

$$\int_{0}^{\pi} f(\cos(\theta))\sin(\theta)\,d\theta = a_0 + \sum_{k=1}^{\infty} \frac{2a_{2k}}{1-(2k)^2}$$

Where

$$a_k = \frac{2}{k}\int_{0}^{\pi} f(\cos(\theta))\cos(k\theta)\,d\theta$$

This integration cam also open to series as:

$$a_k = \frac{2}{N}\left[\frac{f(1)}{2} + \frac{f(-1)}{2}(-1)^k + \sum_{n=1}^{N-1} f(cos\left(\frac{n\pi}{N}\right) cos\left(\frac{nk\pi}{N}\right)\right]$$

When the actual integral is in between limits of x_1 and x_2 values, coordinate conversion should be apply

$$k_2 = \frac{x_2 - x_1}{2}$$

$$k_1 = \frac{x_2 + x_1}{2}$$

$$\int_{a}^{b} f(x)dx = k_1\left[a_0 + \sum_{k=1}^{\infty} \frac{2a_{2k}}{1-(2k)^2}\right]$$

$$x = k_2 + k_1 cos\left(\frac{n\pi}{N}\right)$$

$$a_k = \frac{2}{N}\left[\frac{f(x_1)}{2} + \frac{f(x_2)}{2}(-1)^k + \sum_{n=1}^{N-1} f(x) cos\left(\frac{nk\pi}{N}\right)\right]$$

```
from math import *
from if_x import *


class f1(if_x):
  def func(self,x):
      y=x*x
      return y

def clenshaw_curtis_coefficients(f,x1,x2,N):
    a=[0.0 for z in range(2*N+1)]
    cc=-1
    k1=(x2-x1)/2.0
    k2=(x2+x1)/2.0
    for k in range(2*N):
        a[k]=0.0
        for n in range(1,N):
            x=k2+k1*cos(pi*n/N)
            a[k]=a[k]+f.func(x)*cos(n*k*pi/N)
        cc=-cc
```

```
        a[k]=a[k]+f.func(x2)/2.0+f.func(x1)/2.0*cc
        a[k]=a[k]*2.0/n
    return a

def clenshaw_curtis_integral(f,x1,x2,N):
    a=clenshaw_curtis_coefficients(f,x1,x2,N)
    z=a[0]
    k1=(x2-x1)/2.0
    k2=(x2+x1)/2.0
    for k in range(1,2,N):
        z=z+2.0*a[2*k]/(1.0-4.0*float(k*k))
    return k1*z

def Aitken_clenshaw_curtis(ff,a,b,N):
    #Aitken interpolation formula
    I4n=clenshaw_curtis_integral(ff,a,b,4*N)
    I2n=clenshaw_curtis_integral(ff,a,b,2*N)
    In=clenshaw_curtis_integral(ff,a,b,N)
    I=I4n-(I4n-I2n)*(I4n-I2n)/((I4n-I2n)-(I2n-In))
    return I

f=f1()
I=clenshaw_curtis_integral(f,0.0,1.0,200)
I2=Aitken_clenshaw_curtis(f,0.0,1.0,200);
print("I=",I)
print("I2=",I2)
```

```
runfile('E:/okul/SCO1/clenshaw_curtis_integral.py', wdir='E:/okul/SCO1')
Reloaded modules: if_x
I= 0.3350083752093801
I2= 0.3333354166796865
```

4.6 INTEGRATION FORMULATION WITH ADJUSTABLE ERROR

If integration in region **[a,b]** to be defined for a continious integrable function f(x), Integration can be written as:

$I = \int_a^b f(x)dx$

If a sub-region is defined for the integration. For example, $m = \frac{a-b}{2}$ definition is given integral can be redefined as

$$I = \int_a^m f(x)dx + \int_m^b f(x)dx$$

For numerical integration it is generally assumed that the error of the second definition will be less than the first definition. It is possible to continue this process by dividing each integral to subregions again. The important concept in here is to stop dividing integral into smaller parts when the desired accuracy is achieved. In this section several integration methods with error control will be investigated.

4.5.1 Adjustable Simpson 1/3 integral

If simpson 1/3 integration equation is written in [a,b] region

$$I(a,b) = I(x_0, x_4) = \frac{h}{3}[f(x_0) + 4f(x_2) + f(x_4)]$$

$$\int_a^b f(x)dx = \int_{x_0}^{x_4} f(x)dx = I(x_0, x_4) - h^5 \frac{f^{(4)}(d_1)}{90}$$

Where

$$h = \frac{b-a}{2}$$

$$x_0 = a$$

$$x_2 = \frac{a+b}{2}$$
$$x_4 = b$$

and d_1 is a number between a and b. and $h^5 \frac{f^{(4)}(d_1)}{90}$ equation gives amount of error in the integral equation. $f^{(4)}$term in the error equation is the fourth derivative of the function. If the integration region is divided into two subregion, equation becomes

$$I(x_0, x_2) = \frac{h}{6}[f(x_0) + 4f(x_1) + f(x_2)]$$
$$I(x_2, x_4) = \frac{h}{6}[f(x_2) + 4f(x_3) + f(x_4)]$$
$$\int_{x_0}^{x_2} f(x)dx + \int_{x_2}^{x_4} f(x)dx = I(x_0, x_2) + I(x_2, x_4) - \frac{h^5}{16}\frac{f^{(4)}(d_2)}{90}$$

In this equation

$$x_2 = \frac{a+b}{4}$$
$$x_3 = \frac{3(a+b)}{4}$$

.d_2 is stil a value in between a and b.

If it is assumed that $f^{(4)}(d_1) \cong f^{(4)}(d_2)$Equation becomes

$$h^5 \frac{f^{(4)}(d_1)}{90} \cong \frac{16}{15}[I(x_0, x_2) + I(x_2, x_4) - I(x_0, x_4)]$$
$$\frac{h^5}{16}\frac{f^{(4)}(d_1)}{90} \cong \frac{1}{15}[I(x_0, x_2) + I(x_2, x_4) - I(x_0, x_4)]$$

If we substitute this into integration equation

$$\left|\frac{1}{15}[I(x_0, x_2) + I(x_2, x_4) - I(x_0, x_4)]\right| \cong \frac{1}{15}|I(x_0, x_2) + I(x_2, x_4) - I(x_0, x_4)|$$

In this case for error ε>0 , it can be written the equation

$$\frac{1}{15}|I(x_0, x_2) + I(x_2, x_4) - I(x_0, x_4)| < \varepsilon$$

If this condition is met, then approximately

$$\int_{x_0}^{x_4} f(x)dx \cong I(x_0, x_2) + I(x_2, x_4)$$

can be written. If the given condition can not be met, division process will be continue till the condition met. Described algorithm is used in the next program.

Program Simson 1/3 integration with adjustable error

```
 from math import *
from if_x import *
def simpson1_3(f,x,y,sum,eps,k,MI,Q):
   #Newton-cotes integral 3 points
   #adaptive Simpson 1/3 rule
   h=(x[4]-x[0]);
   h1=h/2.0;
   h2=h/4.0;
   x1=[0 for z in range(5)]
   y1=[0 for z in range(5)]
   x2=[0 for z in range(5)]
   y2=[0 for z in range(5)]
   k=k+1
   for i in range(5):
      x[i]=x[0]+i*h2
      y[i]=f.func(x[i])
```

```
    sum1=h1*(y[0]+4.0*y[1]+y[2])/6.0;
    sum2=h1*(y[2]+4.0*y[3]+y[4])/6.0;
    x1[0]=x[0];y1[0]=y[0];
    x1[2]=x[1];y1[2]=y[1];
    x1[4]=x[2];y1[4]=y[2];
    x2[0]=x[2];y2[0]=y[2];
    x2[2]=x[3];y2[2]=y[3];
    x2[4]=x[4];y2[4]=y[4];
    if 1.0/15.0*abs(sum1+sum2-sum)<eps or k>MI:
        Q=Q+sum1+sum2
    else:
        Q=Q+simpson1_3(f,x1,y1,sum1,eps,k,MI,Q)+simpson1_3(f,x2,y2,sum2,eps,k,MI,Q);
    return Q

def adaptive_simpson_integral(f,a,b,eps,MI):
    h=(b-a)
    h2=h/4.0
    x=[0 for z in range(5)]
    y=[0 for z in range(5)]
    for i in range(5):
        x[i]=a+i*h2
        y[i]=f.func(x[i])
    sum=h*(y[0]+4.0*y[2]+y[4])/6.0
    Q=0.0
    Q=simpson1_3(f,x,y,sum,eps,0,MI,Q)
    return Q

class f1(if_x):func=lambda self,x: x*sqrt(x);
f=f1()
eps=1.0e-10
MI=100
I=adaptive_simpson_integral(f,0.0,1.0,eps,MI)
print("I = ",I)
```

Simson 1/3 integration with the adaptive simpson integration

```
runfile('E:/okul/SCO1/ASimpsonIntegral.py', wdir='E:/okul/SCO1')
Reloaded modules: if_x
I =  0.4000000007335708
```

4.5.2 Adjestable Gauss-Legendre integration

The same concept can be repeated by using Gauss-Legendre integration

```
from math import *
from if_x import *
def gauss_legendre_coefficients(x1,x2,n):
    #calculates legendre gauss-coefficients as coefficients of the integral
    #for n terms
        eps=3e-15
        m=int((n+1.0)/2.0)
        xm=0.5*(x2+x1)
        xl=0.5*(x2-x1)
        nmax=100
        a=[[0.0 for i in range(n)] for j in range(2)]
        for i in range(1,m+1):
            z=cos(pi*((i-0.25)/(n+0.5)))
            for ii in range(nmax):
                #while abs(z-z1) > eps:
                p1=1.0;
                p2=0.0;
                for j in range(1,n+1):
                    p3=p2
                    p2=p1
                    p1=(float)(((2.0*j-1.0)*z*p2-(j-1.0)*p3)/j)
                pp=(float)(n*(z*p1-p2)/(z*z-1.0))
                z1=z;
                z=z1-p1/pp
                if abs(z-z1) < eps: break
            a[0][i-1]=xm-xl*z
            a[0][n-i]=xm+xl*z
            a[1][i-1]=2.0*xl/((1.0-z*z)*pp*pp)
```

```
            a[1][n-i]=a[1][i-1]
        return a

def integral(f,x1,x2,n):
        #integral func(x)dx
        #integral of a function by using gauss-legendre quadrature
        #between x1 and x2
        a=gauss_legendre_coefficients(x1,x2,n)
        z=0
        for i in range(n):
            z=z+a[1][i]*f.func(a[0][i])
        return z

def adaptive_integral(f,x1,x2):
    eps=1.0e-12
    MAXITER=100
    return adaptive_integral1(f,x1,x2,eps,MAXITER)

def adaptive_integral1(f,x1,x2,eps,MAXITER):
    n=1;
    ans1=integral(f,x1,x2,n)
    ans2=2.0*ans1
    while(abs(ans2-ans1)>eps and n<MAXITER):
        ans2=ans1
        n=n+1
        ans1=integral(f,x1,x2,n)
    if n==MAXITER: print("requird error level can not be achievd n="+n );
    return ans2


class f1(if_x):func=lambda self,x: x*sqrt(x)
f=f1()
a=0.0
b=1.0
I1=adaptive_integral(f,a,b)
print("I1  : ",I1);
```

```
runfile('E:/okul/SCO1/integral_GL3.py', wdir='E:/okul/SCO1')
Reloaded modules: if_x
I1  :  0.3999999999829868
```

4.5.3 Adjustable Gauss-Kronrod integration

Gauss-Kronrod integration formula

$$I_w(-1,1) \cong \int_{-1}^{1} f(x)dx \cong \sum_{k=1}^{n} c_k f\left(x_k^G\right) + \sum_{k=1}^{n} c_{n+k} f\left(x_k^K\right)$$

Gauss Kronrod integration formula uses two terms. First term is standard Gauss-Legendre integration term. The second term coefficients are taken as same as the Gauss-Legendre term, but roots are different. It is called Kronrod integration. In our example code constant integration of 15 terms will be used for each equation (total of 30 terms). Program utilises Gauss-Legendre terms as first, and then uses Gauss-Kronrod terms. To improve error reduction equation can be continued with half dividing technique. Gauss-Kronrod Formula is prefferred method for many package programs such as **Mathlab**.

```
from math import *
from if_x import *
def gauss_kronrod_integral(ff,a, b, eps,n):
    # eps hata miktarı
    Aused=0
    integral=0
    toplamhata = 0
    nn = 0
    ng = 0
    i = 0
    j = 0
    h = 0
    v = 0
    k1 = 0
```

```
k2 = 0
intg = 0
intk = 0
ta = 0
tb = 0
Aw = 4
A = [[0.0 for i in range(Aw-1+1)] for j in range(n-1+1)]    #[n-1+1][Aw-1+1];
nn = 61;
ng = 15;
x =[0.0 for i in range(nn)]
ck =[0.0 for i in range(nn)]
cg =[0.0 for i in range(nn)]
#Gauss coefficients
cg[0] = 0.007968192496166605615465883474674
cg[1] = 0.018466468311090959142302131912047
cg[2] = 0.028784707883323369349719179611292
cg[3] = 0.038799192569627049596801936446348
cg[4] = 0.048402672830594052902938140422808
cg[5] = 0.057493156217619066481721689402056
cg[6] = 0.065974229882180495128128515115962
cg[7] = 0.073755974737705206268243850022191
cg[8] = 0.080755895229420215354694938460530
cg[9] = 0.086899787201082979802387530715126
cg[10] = 0.092122522237786128717632707087619
cg[11] = 0.096368737174644259639468626351810
cg[12] = 0.099593420586795267062780282103569
cg[13] = 0.101762389748405504596428952168554
cg[14] = 0.102852652893558840341285636705415
x[0] = 0.999484410050490637571325895705811
x[1] = 0.996893484074649540271630050918695
x[2] = 0.991630996870404594858628366109486
x[3] = 0.983668123279747209970032581605663
x[4] = 0.973116322501126268374693868423707
x[5] = 0.960021864968307512216871025581798
x[6] = 0.944374444748559979415831324037439
x[7] = 0.926200047429274325879324277080474
x[8] = 0.905573307699907798546522558925958
x[9] = 0.882560535792052681543116462530226
x[10] = 0.857205233546061098958658510658944
x[11] = 0.829565762382768397442898119732502
x[12] = 0.799727835821839083013668942322683
x[13] = 0.767777432104826194917977340974503
x[14] = 0.733790062453226804726171131369528
x[15] = 0.697850494793315796932292388026640
x[16] = 0.660061064126626961370053668149271
x[17] = 0.620526182989242861140477556431189
x[18] = 0.579345235826361691756024932172540
x[19] = 0.536624148142019899264169793311073
x[20] = 0.492480467861778574993693061207709
x[21] = 0.447033769538089176780609900322854
x[22] = 0.400401254830394392535476211542661
x[23] = 0.352704725530878113471037207089374
x[24] = 0.304073202273625077372677107199257
x[25] = 0.254636926167889846439805129817805
x[26] = 0.204525116682309891438957671002025
x[27] = 0.153869913608583546963794672743256
x[28] = 0.102806937966737030147096751318001
x[29] = 0.051471842555317695833025213166723
x[30] = 0.000000000000000000000000000000000
#Kronrod coefficients
ck[0] = 0.001389013698677007624551591226760
ck[1] = 0.003890461127099884051267201844516
ck[2] = 0.006630703915931292173319826369750
ck[3] = 0.009273279659517763428441146892024
ck[4] = 0.011823015253496341742232898853251
ck[5] = 0.014369729507045804812451432443580
ck[6] = 0.016920889189053272627572289420322
ck[7] = 0.019414141193942381173408951050128
ck[8] = 0.021828035821609192297167485738339
ck[9] = 0.024191162078080601365686370725232
ck[10] = 0.026509954882333101610601709335075
ck[11] = 0.028754048765041292843978785354334
ck[12] = 0.030907257562387762472884252943092
ck[13] = 0.032981447057483726031814191016854
ck[14] = 0.034979338028060024137499670731468
ck[15] = 0.036882364651821229223911065617136
```

```
ck[16] = 0.03867894562472759295034865153228 1
ck[17] = 0.04037453895153595911199527975246 8
ck[18] = 0.04196981021516424614714754128597 0
ck[19] = 0.04345253970135606931683172811707 3
ck[20] = 0.04481480013331626631923555516167 23
ck[21] = 0.04605923827100698811627173555937 4
ck[22] = 0.04718554656929915394526147818109 9
ck[23] = 0.04818586175708712914077949229830 5
ck[24] = 0.04905543455502977888752816536723 8
ck[25] = 0.04979568342707420635781156937994 2
ck[26] = 0.05040592140278234684089308565358 5
ck[27] = 0.05088179589874960649229747304980 5
ck[28] = 0.05122154784925877217065628260494 4
ck[29] = 0.05142612853745902593386287921578 1
ck[30] = 0.05149472942945156755834043364709 9
for i in range(nn-1,int(nn/2-1),-1):
    x[i] = -x[nn-1-i]
for i in range(nn-1,int(nn/2-1),-1):
    ck[i] = ck[nn-1-i]
for i in range(ng-1,-1,-1):
    cg[nn-2-2*i] = cg[i]
    cg[1+2*i] = cg[i]
for i in range(0,int(nn/2+1)):
    cg[2*i] = 0
k1 = 0.5*(b-a);
k2 = 0.5*(b+a);
intg = 0;
intk = 0;
for i in range(0,nn):
    v = ff.func(k1*x[i]+k2)
    intk = intk+v*ck[i];
    if  i%2==1:
        intg = intg+v*cg[i]
intk = intk*(b-a)*0.5
intg = intg*(b-a)*0.5
A[0][0] = abs(intg-intk);
A[0][1] = intk;
A[0][2] = a;
A[0][3] = b;
toplamhata = A[0][0]
if toplamhata<eps:
    sonuc = True
    integral = intk
    Aused = 1
    return integral
Aused = 1
for h in range(1,n):
    Aused = h+1
    gir(A, h, Aw)
    toplamhata = toplamhata-A[h-1][0]
    ta = A[h-1][2]
    tb = A[h-1][3]
    A[h-1][2] = ta
    A[h-1][3] = 0.5*(ta+tb)
    A[h][2] = 0.5*(ta+tb)
    A[h][3] = tb
    for j in range(h-1,h+1):
        k1 = 0.5*(A[j][3]-A[j][2])
        k2 = 0.5*(A[j][3]+A[j][2])
        intg = 0.0
        intk = 0.0
        for i in range(0,nn):
            v = ff.func(k1*x[i]+k2);
            intk = intk+v*ck[i];
            if  i%2==1:
                intg = intg+v*cg[i]
        intk = intk*(A[j][3]-A[j][2])*0.5
        intg = intg*(A[j][3]-A[j][2])*0.5
        A[j][0] = abs(intg-intk)
        A[j][1] = intk
        toplamhata = toplamhata+A[j][0]

    cik(A, h-1, Aw);
    cik(A, h, Aw);
    if toplamhata<eps:
        break
```

```
    sonuc = toplamhata<eps
    integral = 0
    for j in range(0,Aused):
        integral = integral+A[j][1]
    return integral

def gir(A,n,Awidth):
    i = 0
    p = 0
    t = 0
    maxcp = 0
    if n==1:
        return
    for i in range(0,Awidth):
        t = A[n-1][i]
        A[n-1][i] = A[0][i]
        A[0][i] = t
    p = 0
    while 2*p+1<n-1:
        maxcp = 2*p+1;
        if 2*p+2<n-1:
            if A[2*p+2][0]>A[2*p+1][0]:
                maxcp = 2*p+2
        if  A[p][0]<A[maxcp][0]:
            for i in range(0,Awidth):
                t = A[p][i]
                A[p][i] = A[maxcp][i]
                A[maxcp][i] = t
            p = maxcp
        else:
            break



def cik(A,n,Awidth):
    i = 0
    p = 0
    t = 0
    kk = 0
    if n==0:
        return
    p = n
    while p!=0:
        kk =int( (p-1)/2)
        if A[p][0]>A[kk][0]:
            for i in range(0,Awidth):
                t = A[p][i];
                A[p][i] = A[kk][i]
                A[kk][i] = t
            p = kk
        else:
            break


class f1(if_x):func=lambda self,x: x*sqrt(x);
f=f1()
print("integral of class        f1 : ",gauss_kronrod_integral(f,0.0,1.0,1.0e-20,50))
```

```
runfile('E:/okul/SCO1/adaptive_Gauss_Kronrod_integral.py', wdir='E:/okul/SCO1')
Reloaded modules: if_x
integral of class        f1 :  0.4
```

4.6 MONTE-CARLO INTEGRAL

Monte Carlo integral is based on a very simple principle

$$\int_a^b f(x)dx = (b-a)f_{average}$$

$$x = a + (b-a)random_number(0,1)$$

$$f_{average} = \frac{\sum_{i=0}^{n} f(x)}{n} \qquad n \gg 1$$

In the interval [a,b] a random x number is selected and function is evalueated in this point. When this sequaence is repeated for a very large number of times avarage value of the function can be calculated and directly used in the calculatetions of the integral. Integral is the calculateion of the avarage function value with the interval width. Due to fact that this method requires a huge amount of function evaluation, therefore it is not very practical for one variable integration, but in multivariable form it becomes a practical, sometimes only available approach.

Program One variable Monte Carlo integration

```
 from math import *
import random
def monte_carlo_integral1(f,a,b,n):
    h=b-a
    sum=0.0
    x=0.0
    y=0.0
    for i in range(n):
        r=random.random()
        x=a+(b-a)*r
        sum=sum+f(x)
    y=sum/n*(b-a)
    return y



f = lambda x:x*sqrt(x)
a=0.0
b=1.0
n=100000
I1=monte_carlo_integral1(f,a,b,n)
print("I1  : ",I1);
```

One variable Monte-carlo integration

```
 runfile('E:/okul/SCO1/monte_carlo_integral1.py', wdir='E:/okul/SCO1')
I1  :  0.40079847874898333
```

If Monte Carlo equation is written for two independent variables

$$x = a_x + (b_x - a_x) random_number(0,1)$$
$$y = a_y + (b_y - a_y) random_number(0,1)$$
$$f_{average} = \frac{\sum_{i=0}^{n} f(x,y)}{n} \qquad n \gg 1$$

Integration subfield

$$\mu(D) = \int_{a_y}^{b_y} \int_{a_x=\alpha(x)}^{b_x=\beta(x)} dydx$$

Integration

$$I = \mu(D) f_{average} = \mu(D) \frac{\sum_{i=0}^{n} f(x,y)}{n}$$

As it is seen from the equation process fro one variable integration and two variable integraton is almost the same. This is the main factor why it is frequently use as a high dimensional integration tool.

```
from math import *
from f_xj import *
from random import *
# ff is function in f_xj class
def  monte_carlo(ff,a,b,n):
    x=[0.0 for i in range(2)]
    h=1.0
    sum=0
```

```
    for j in range(2):
        h=h*(b[j]-a[j])
    for i in range(n):
        for j in range(2):
            x[j]=a[j]+(b[j]-a[j])*random()
        sum=sum+ff.func(x)
    return h*sum/float(n)

class f1(f_xj):func=lambda self,x: -(2.0*x[0]*x[1]+2.0*x[0]-x[0]*x[0]-2.0*x[1]*x[1])
f=f1()
a=[-1.0,-1.0]
b=[1.0,1.0]
n=1000000
I=monte_carlo(f,a,b,n)
print("integral of class        f1 : ",I)
```

```
runfile('E:/okul/SCO1/monte_carlo_integral2.py', wdir='E:/okul/SCO1')
integral of class        f1 :  3.997230300289306
```

```
runfile('E:/okul/SCO1/monte_carlo_integral2.py', wdir='E:/okul/SCO1')
Reloaded modules: f_xj
integral of class        f1 :  3.999268303856159
```

Due to statistical nature of the method, result wil change in each runs

4.7 MULTIDIMENSIONAL INTEGRATION

A two dimensional integration

$$I(D) = \iint_D f(x,y)dydx$$

Where D is a region in xy plane. If this region is defined as $a \le x \le b$ and $c(x) \le y \le d(x)$, one way two calculate two dimensional integration is to repeat one dimensional integration twice. In the first step intehration function g(x) is calculated as an integration in y dimension and then function g(x) is solved to get integration result.

$$I = \int_a^b \int_{c(x)}^{d(x)} f(x,y)dydx$$

$$g(x) = \int_{c(x)}^{d(x)} f(x,y)dydx \qquad a \le x \le b$$

$$I(D) = \int_a^b g(x)dx$$

Any integration formulation can be used for this purpose. All it has to be done is to evaluate integration Formula twice onece for each independent variable. As our first example Gauss-Legendre formulation will be used to evaluate double integration. Two code is defined. In the first one Gauss-Legendre coefficients are defined as constants, in the second one they are calculated through Legendre polynomials. In the two dimensional integration x dimesion limits are still given as constant values, but y dimension limits c(x) and d(x) should be given as functions for proper evaluation of the integral.

$$I = \int_{x=a}^{b} \int_{y=c(x)}^{d(x)} f(x,y)dydx = \int_{x=a}^{b} \frac{d(x)-c(x)}{2} \sum_{j=1}^{n} c_{n,j} f\left(x, \frac{(d(x)-c(x))r_{n,j} + d(x) + c(x)}{2}\right) dx$$

$$g(x) = \frac{d(x)-c(x)}{2} \sum_{j=1}^{n} c_{n,j} f\left(x, \frac{(d(x)-c(x))r_{n,j} + d(x) + c(x)}{2}\right)$$

$$I = \int_{x=a}^{b} \int_{y=c(x)}^{d(x)} f(x,y)dydx = \int_{x=a}^{b} g(x)dx = \frac{b-a}{2} \sum_{i=1}^{m} c_{m,i} f\left(\frac{b-a}{2} r_{m,i} + \frac{b+a}{2}\right)$$

In this equation $r_{n,j}$ root values and $c_{n,j}$ are equation coefficients for integration of y, and $r_{m,i}$ root values and $c_{m,i}$ are equation coefficients for integration of x. If n=m r and c coefficients will be same. These values can be calculated from Legendre polynomials or they can be directly given as an array in computer code.

Example function:

$$\int_{x=1.4}^{2} \int_{y=1}^{1.5} \ln(x+2y)\, dydx$$

```
 # -*- coding: utf-8 -*-
"""
Created on Mon Dec 20 14:24:53 2021

@author:Mustafa Turhan Coban
"""
#integral_3D_GL2.py
from math import *


#2 dimensional function

def I(f,a,b,fc,fd):
    #integral f(x)dx
    #integral of a function by using gauss-legendre quadrature
    #coefficients are pre-calculated for 60 terms for [-1,1]
    #band then utilises variable transform
    n=60;
    r=[0 for z in range(n)]
    c=[0 for z in range(n)]
    r[ 0] = .15532579626752470000E-02;
    r[ 1] = .81659383601264120000E-02;
    r[ 2] = .19989067515846230000E-01;
    r[ 3] = .36899976285362850000E-01;
    r[ 4] = .58719732103973630000E-01;
    r[ 5] = .85217118808615820000E-01;
    r[ 6] = .11611128394758690000E+00;
    r[ 7] = .15107475260334210000E+00;
    r[ 8] = .18973690850537860000E+00;
    r[ 9] = .23168792592899010000E+00;
    r[10] = .27648311523095540000E+00;
    r[11] = .32364763723456090000E+00;
    r[12] = .37268153691605510000E+00;
    r[13] = .42306504319570830000E+00;
    r[14] = .47426407872234120000E+00;
    r[15] = .52573592127765890000E+00;
    r[16] = .57693495680429170000E+00;
    r[17] = .62731846308394490000E+00;
    r[18] = .67635236276543910000E+00;
    r[19] = .72351688476904450000E+00;
    r[20] = .76831207407100990000E+00;
    r[21] = .81026309149462140000E+00;
    r[22] = .84892524739665800000E+00;
    r[23] = .88388871605241310000E+00;
    r[24] = .91478288119138420000E+00;
    r[25] = .94128026789602640000E+00;
    r[26] = .96310002371463720000E+00;
    r[27] = .98001093248415370000E+00;
    r[28] = .99183406163987350000E+00;
    r[29] = .99844674203732480000E+00;
    c[ 0] = .39840962480827790000E-02;
    c[ 1] = .92332341555455000000E-02;
    c[ 2] = .14392353941661670000E-01;
    c[ 3] = .19399596284813530000E-01;
    c[ 4] = .24201336415292590000E-01;
    c[ 5] = .28746578108808720000E-01;
    c[ 6] = .32987114941090080000E-01;
    c[ 7] = .36877987368852570000E-01;
    c[ 8] = .40377947614710090000E-01;
    c[ 9] = .43449893600541500000E-01;
    c[10] = .46061261118893050000E-01;
```

```
    c[11] = .48184368587322120000E-01;
    c[12] = .49796710293397640000E-01;
    c[13] = .50881194874202750000E-01;
    c[14] = .51426326446779420000E-01;
    c[15] = .51426326446779420000E-01;
    c[16] = .50881194874202750000E-01;
    c[17] = .49796710293397640000E-01;
    c[18] = .48184368587322120000E-01;
    c[19] = .46061261118893050000E-01;
    c[20] = .43449893600541500000E-01;
    c[21] = .40377947614710090000E-01;
    c[22] = .36877987368852570000E-01;
    c[23] = .32987114941090080000E-01;
    c[24] = .28746578108808720000E-01;
    c[25] = .24201336415292590000E-01;
    c[26] = .19399596284813530000E-01;
    c[27] = .14392353941661670000E-01;
    c[28] = .92332341555455000000E-02;
    c[29] = .39840962480827790000E-02;
    for i in range(30):
        r[i+30]=-r[i]
        c[i+30]=c[i]
    m=n
    x=[0 for z in range(2)]
    #1=========================
    a1=a
    b1=b;
    h1=(b1-a1)/2.0;
    h2=(b1+a1)/2.0;
    J=0;
    #2=========================
    for i in range(m):
        JX=0
        x[0]=h1*r[i]+h2
        d1=fd(x[0])
        c1=fc(x[0])
        k1=(d1-c1)/2.0;
        k2=(d1+c1)/2.0;
        #3==========================
        for j in range(n):
            x[1]=k1*r[j]+k2
            Q=f(x)
            JX+=c[j]*Q
        J=J+c[i]*k1*JX
    J=h1*J;
    return J

a=1.4;
b=2.0;
ff=lambda x:log(x[0]+2.0*x[1])
fc=lambda x:1
fd=lambda x:1.5
I=I(ff,a,b,fc,fd)
print("integral = : ",I)
```

```
runfile('D:/okul/SCO1/integral_3D_GL2.py', wdir='D:/okul/SCO1')
integral = :  0.429554527548266
```

In the next example Guass-Legendre polynomial coefficients will be generated by Legendre polynomial formulations. C and r values are directly calculated from Legendre polynomials.

```
 #integral_3D_GL3.py
from math import *
#2 dimensional function


def gauss_legendre_coefficients(x1,x2,n):
    #calculates legendre gauss-coefficients as coefficients of the integral
    #for n terms
        eps=3e-15
        m=int((n+1.0)/2.0)
        xm=0.5*(x2+x1)
        xl=0.5*(x2-x1)
        nmax=100
```

```
        a=[[0.0 for i in range(n)] for j in range(2)]
        for i in range(1,m+1):
            z=cos(pi*((i-0.25)/(n+0.5)))
            for ii in range(nmax):
                #while abs(z-z1) > eps:
                p1=1.0;
                p2=0.0;
                for j in range(1,n+1):
                    p3=p2
                    p2=p1
                    p1=(float)(((2.0*j-1.0)*z*p2-(j-1.0)*p3)/j)
                pp=(float)(n*(z*p1-p2)/(z*z-1.0))
                z1=z;
                z=z1-p1/pp
                if abs(z-z1) < eps: break
            a[0][i-1]=xm-xl*z
            a[0][n-i]=xm+xl*z
            a[1][i-1]=2.0*xl/((1.0-z*z)*pp*pp)
            a[1][n-i]=a[1][i-1]
        return a

def I(f,a,b,fc,fd,n,m):
    #integral f(x)dx
    #integral of a function by using gauss-legendre quadrature
    #coefficients are pre-calculated for 60 terms for [-1,1]
    #band then utilises variable transform
    rn=[0 for z in range(n)]
    cn=[0 for z in range(n)]
    rm=[0 for z in range(n)]
    cm=[0 for z in range(n)]
    an=gauss_legendre_coefficients(-1.0,1.0,n)
    rn=an[0]
    cn=an[1]
    am=gauss_legendre_coefficients(-1.0,1.0,m)
    rm=am[0]
    cm=am[1]
    x=[0 for z in range(2)]
    #1=========================
    a1=a
    b1=b
    h1=(b1-a1)/2.0;
    h2=(b1+a1)/2.0;
    J=0;
    #2=========================
    for i in range(m):
        JX=0;
        x[0]=h1*rm[i]+h2;
        d1=fd(x[0]);
        c1=fc(x[0]);
        k1=(d1-c1)/2.0;
        k2=(d1+c1)/2.0;
        #3==========================
        for j in range(n):
            x[1]=k1*rn[j]+k2
            Q=f(x)
            JX+=cn[j]*Q
        J=J+cm[i]*k1*JX
    J=h1*J;
    return J

a=1.4;
b=2.0;
ff=lambda x:log(x[0]+2.0*x[1])
fc=lambda x:1
fd=lambda x:1.5
n=60
m=60
I=I(ff,a,b,fc,fd,n,m)
print("integral = : ",I)
```

```
runfile('E:/okul/SCO1/integral_3D_GL1.py', wdir='E:/okul/SCO1')
integral = :  0.4295545275482747
```

For the two variable integration **Newton-Cotes formulas** can also be used. As a example a Simpson 1/3 two dimensional integration code is prepared. For the quadratic polynomial Newton-Cotes equation (simpson 1/3 method) equations can be written as follows;
One dimensional form was

$$h = \frac{b-a}{n} = \frac{b-a}{2}$$

$$I = \int_a^b f(x)dx = \frac{b-a}{6}[f(x_0) + 4f(x_1) + f(x_2)]$$

If this Formula can be written in composite form.

$$I = \int_a^b f(x)dx = \frac{h}{3}\left[f(a) + 2\sum_{j=1}^{\left(\frac{n}{2}\right)-1} f\left(x_{2j}\right) + 4\sum_{j=1}^{\left(\frac{n}{2}\right)} f\left(x_{2j-1}\right) + f(b)\right]$$

This Formula can be open to the second dimension similar to Gauss-Legendre integration
By defining $k = \frac{d(x)-c(x)}{m}$ integration equation becomes

$$I = \int_{c(x)}^{d(x)} f(x,y)dy = \frac{k}{3}\left[f(x,y_0) + 2\sum_{j=1}^{\left(\frac{m}{2}\right)-1} f\left(x,y_{2j}\right) + 4\sum_{j=1}^{\left(\frac{m}{2}\right)} f\left(x,y_{2j-1}\right) + f(x,y_m)\right]$$

From here x integration can be carried out as:

$$= \int_a^b \int_{c(x)}^{d(x)} f(x,y)dy = \int_a^b \frac{k}{3}\left[f(x,y_0) + 2\sum_{j=1}^{\left(\frac{m}{2}\right)-1} f\left(x,y_{2j}\right) + 4\sum_{j=1}^{\left(\frac{m}{2}\right)} f\left(x,y_{2j-1}\right) + f(x,y_m)\right]dx$$

The same integration equation is then applied to the outer integration. Complete code is given in the program below. In the example the same function as in previous Gauss-Legendre integration is taken to compare the results

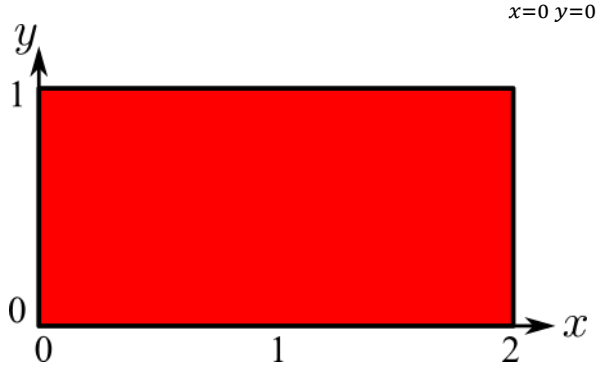
```
 # integral_3D_NC1
from math import *

def I(f,a,b,fc,fd,m,n):
   # integral f(x)dx
   # integral of a function by using simpson metod
   h=(b-a)/n
   J1=0.0
   J2=0.0
   J3=0.0
   x=[0.0 for i in range(2)]
   HX=0.0
   # double K1,K2,K3
   # double d,c
   # double Q,L,J
   for i in range(n+1):
      x[0]=a+h*i
      d=fd(x[0])
      c=fc(x[0])
      HX=(d-c)/float(m)
      x[1]=c
      K1=f(x)
      x[1]=d
      xk=f(x)
      K1=K1+xk
      K2=0.0
      K3=0.0
      for j in range(1,m):
         x[1]=fc(x[0])+float(j)*HX
         Q=f(x)
         #if i==0:
         if(int(j/2)*2==j):
            K2=K2+Q
         else:
```

```
            K3=K3+Q
        L=(K1+2.0*K2+4.0*K3)*HX/3.0
        if (i==0) or (i==n):
            J1=J1+L
        elif int(i/2)*2==i:
            J2=J2+L
        else:
            J3=J3+L
    J=h*(J1+2.0*J2+4.0*J3)/3.0
    return J

a=1.4;
b=2.0;
ff=lambda x:log(x[0]+2.0*x[1])
fc=lambda x:1
fd=lambda x:1.5
n=60
m=60
I=I(ff,a,b,fc,fd,n,m)
print("integral = : ",I)
```

```
 runfile('D:/okul/SCO1/integral_3D_NC1.py', wdir='D:/okul/SCO1')
integral = :  0.4295545275452859
```

Some more examples

$$\int_{x=0}^{2}\int_{y=0}^{1} xy^2 dx\, dy$$



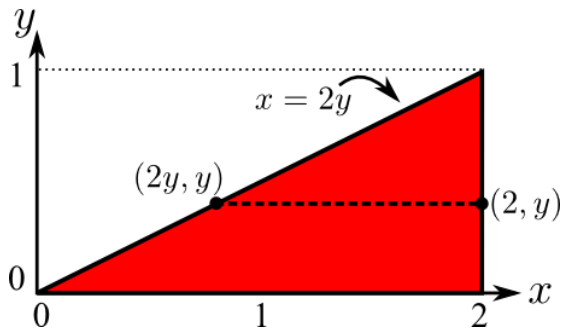
This integral analytically solvable let us do it first

$$\int_{y=0}^{1}\left(\int_{x=0}^{2} xy^2 dx\right) dy = \int_{y=0}^{1}\left(\frac{x^2}{2}y^2\Big|_{x=0}^{2}\right) = \int_{y=0}^{1} 2y^2 dy = \frac{2}{3}y^3|_{y=0}^{1} = \frac{2}{3}$$

```
 a=0.0;
b=2.0;
ff=lambda x:x[0]*x[1]*x[1]
fc=lambda x:0.0
fd=lambda x:1.0
I=I(ff,a,b,fc,fd)
print("integral = : ",I)
```

```
runfile('D:/okul/SCO1/integral_3D_GL2.py', wdir='D:/okul/SCO1')
integral = :  0.6666666666666499
```

$$\int_{y=0}^{1}\left(\int_{x=2y}^{2} xy^2 dx\right) dy$$

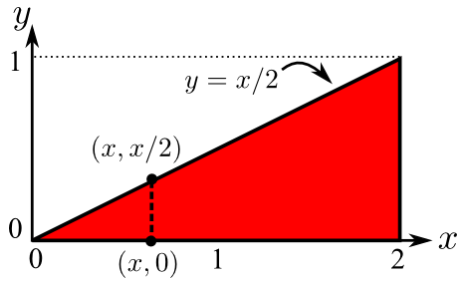


$$\int_{y=0}^{1}\left(\int_{x=2y}^{2} xy^2 dx\right)dy = \int_{y=0}^{1}\left(\frac{x^2}{2}y^2\Big|_{x=2y}^{2}\right)dy = \int_{y=0}^{1}(2y^2-2y^4)\,dy = \frac{2}{3}y^3-\frac{2}{5}y^5\Big|_{y=0}^{1} = \frac{4}{15}$$

```
a=0.0;
b=1.0;
ff=lambda x:x[1]*x[0]*x[0]
fc=lambda y:2.0*y
fd=lambda y:2.0
I=I(ff,a,b,fc,fd)
print("integral = : ",I)
```

```
runfile('D:/okul/SCO1/integral_3D_GL2.py', wdir='D:/okul/SCO1')
integral = :  0.2666666666666609
```

$$\int_{x=0}^{2}\left(\int_{y=0}^{x/2} xy^2 dx\right)dy$$



$$\int_{x=0}^{2}\left(\int_{y=0}^{x/2} xy^2 dy\right)dx = \int_{x=0}^{2} x\frac{y^3}{3}\Big|_{y=0}^{x/2} dx = \int_{x=0}^{2}\frac{x^4}{24}dx = \frac{x^5}{120}\Big|_{0}^{2} = \frac{4}{15}$$

```
a=0.0;
b=2.0;
ff=lambda x:x[0]*x[1]*x[1]
fc=lambda x:0.0
fd=lambda x:x/2.0
I=I(ff,a,b,fc,fd)
print("integral = : ",I)
```

```
runfile('D:/okul/SCO1/integral_3D_GL2.py', wdir='D:/okul/SCO1')
integral = :  0.26666666666665917
```

Now the same principle can be used for three dimensional integration process. A three dimensional integral can be defined as:

$$I = \int_{x=a}^{b}\int_{y=c(x)}^{d(x)}\int_{z=\alpha(x,y)}^{\beta(x,y)} f(x,y,z)dzdxdy$$

Gauss-Legendre integration will be used here as a solution method.
Example integral:

$$I = \int_0^1 \int_x^1 \int_0^y y^2 z dz dy dx$$

```
 #integral_4D_GL2.py
# x=x[0] y=x[1] z=x[2] integral(f(x,y,z)dxdydz)
from math import *
# M. Turhan Coban

def gauss_legendre_coefficients(x1,x2,n):
    #calculates legendre gauss-coefficients as coefficients of the integral
    #for n terms
    eps=3e-15
    m=int((n+1.0)/2.0)
    xm=0.5*(x2+x1)
    xl=0.5*(x2-x1)
    nmax=100
    a=[[0.0 for i in range(n)] for j in range(2)]
    for i in range(1,m+1):
        z=cos(pi*((i-0.25)/(n+0.5)))
        for ii in range(nmax):
            #while abs(z-z1) > eps:
            p1=1.0;
            p2=0.0;
            for j in range(1,n+1):
                p3=p2
                p2=p1
                p1=(float)(((2.0*j-1.0)*z*p2-(j-1.0)*p3)/j)
            pp=(float)(n*(z*p1-p2)/(z*z-1.0))
            z1=z;
            z=z1-p1/pp
            if abs(z-z1) < eps: break
        a[0][i-1]=xm-xl*z
        a[0][n-i]=xm+xl*z
        a[1][i-1]=2.0*xl/((1.0-z*z)*pp*pp)
        a[1][n-i]=a[1][i-1]
    return a

def I(f,a1,b1,fc,fd,falfa,fbeta,n,m,p):
    # integral f(x)dx where x= x[0],x[1],x[2]
    #integral of a function by using gauss-legendre quadrature
    rp=[0.0 for i in range(p)]
    cp=[0.0 for i in range(p)]
    nn=max(n,max(m,p))
    x=[0.0 for i in range(3)]
    a=a=[[0.0 for i in range(nn)] for j in range(2)]
    mm=max(max(n,m),p);
    a=gauss_legendre_coefficients(-1.0,1.0,mm)
    rn=rm=rp=a[0]
    cn=cm=cp=a[1]
    # double h1,h2,J,a1,b1,c1,d1,k1,k2,JX,JY,Q,l1,l2,alfa1,beta1
    h1=(b1-a1)/2.0
    h2=(b1+a1)/2.0
    J=0
    for i in range(m):
        JX=0.0
        x[0]=h1*rm[i]+h2
        d1=fd(x[0])
        c1=fc(x[0])
        k1=(d1-c1)/2.0
        k2=(d1+c1)/2.0
        #4=========================
        for j in range(n):
            JY=0;
            x[1]=k1*rn[j]+k2
            beta1=fbeta(x)
            alfa1=falfa(x)
            l1=(beta1-alfa1)/2.0
            l2=(beta1+alfa1)/2.0
            for k in range(p):
                x[2]=l1*rp[k]+l2
                Q=f(x)
                JY+=cp[k]*Q
            JX+=cn[j]*l1*JY;
```

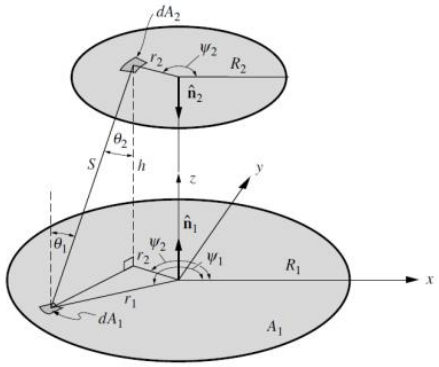
```
        #5=========================
        J=J+cm[i]*k1*JX
    J=h1*J
    return J

a=0.0
b=1.0
ff=lambda x:x[1]*x[1]*x[2]
fc=lambda x:x
fd=lambda x:1.0
falfa=lambda x:0.0
fbeta=lambda x:x[1]
n=60
m=60
p=60
I=I(ff,a,b,fc,fd,falfa,fbeta,n,m,p)
print("integral = : ",I)
```

```
runfile('D:/okul/SCO1/integral_4D_GL2.py', wdir='D:/okul/SCO1')
integral = :  0.08333333333333286
```

By using the same priciples, a fourth dimensional integral can also be solved.

$$I = \int_{x=a}^{b} \int_{y=c(x)}^{d(x)} \int_{z=\alpha(x,y)}^{\beta(x,y)} \int_{t=\gamma(x,y,z)}^{\theta(x,y,z)} f(x,y,z,t)dtdzdydx$$

As an example problem, let us look at radiation shape factor problem in radiative heat transfer. In this problem shape factor of two parallel round plate will be calculated.



Radiation shape factor is the percentage of radiative energy reached to surface A2, emitted from surface A1. If the Radius of plates are R1 and R2, and the distance in between is h, the radiation shape factor eqaution is given as:

$$F_{A1-A2} = \frac{1}{\pi^2 R_1^2} \int_{r1=0}^{R_1} \int_{r2=0}^{R_2} \int_{\psi_1=0}^{2\pi} \int_{\psi_2=0}^{2\pi} \frac{h^2 r_1 r_2 d\psi_1 d\psi_2 dr_1}{[h^2 + r_1^2 + r_2^2 - 2r_1 r_2 \cos(\psi_1 - \psi_2)]}$$

This equation has an analytical solution as:

$$F_{A1-A2} = \frac{R_2}{R_1}\left(X - \sqrt{X^2 - 1}\right)$$

$$X = \frac{h^2 + R_1^2 + R}{2R_1 R_2}$$

In the following program numerical solution by Gauss-Legendre integration is obtained and compared to the analytical solution.

```
 # -*- coding: utf-8 -*-
"""
Created on Sat Dec 25 05:48:08 2021

@author: M. Turhan Coban
"""
#integral_5D_GL2.py
# Original program is written by Hamza Taş in Java Language
# as a part of PhD. level numerical heat radiation class
```

```
from math import *
#2 dimensional function

def f(x):
    z=(h*h+x[0]*x[0]+x[1]*x[1]-2.0*x[0]*x[1]*cos(x[2]-x[3]))
    ff=1.0/(pi*pi*R1*R1)*h*h*x[0]*x[1]/(z*z)
    return ff

# analytical solution of the integral function
def F12(h,R1,R2):
    X=(h*h+R1*R1+R2*R2)/(2.0*R1*R2)
    F=R2/R1*(X-sqrt(X*X-1.0))
    return F

def gauss_legendre_coefficients(x1,x2,n):
    #calculates legendre gauss-coefficients as coefficients of the integral
    #for n terms
    eps=3e-15
    m=int((n+1.0)/2.0)
    xm=0.5*(x2+x1)
    xl=0.5*(x2-x1)
    nmax=100
    a=[[0.0 for i in range(n)] for j in range(2)]
    for i in range(1,m+1):
        z=cos(pi*((i-0.25)/(n+0.5)))
        for ii in range(nmax):
            #while abs(z-z1) > eps:
            p1=1.0;
            p2=0.0;
            for j in range(1,n+1):
                p3=p2
                p2=p1
                p1=(float)(((2.0*j-1.0)*z*p2-(j-1.0)*p3)/j)
            pp=(float)(n*(z*p1-p2)/(z*z-1.0))
            z1=z;
            z=z1-p1/pp
            if abs(z-z1) < eps: break
        a[0][i-1]=xm-xl*z
        a[0][n-i]=xm+xl*z
        a[1][i-1]=2.0*xl/((1.0-z*z)*pp*pp)
        a[1][n-i]=a[1][i-1]
    return a

def I(f,a1,b1,fc,fd,falfa,fbeta,fgama,fteta,n,m,p,s):
    rn=[0.0 for i in range(n)]
    rm=[0.0 for i in range(m)]
    rp=[0.0 for i in range(p)]
    rs=[0.0 for i in range(s)]
    cn=[0.0 for i in range(n)]
    cm=[0.0 for i in range(m)]
    cp=[0.0 for i in range(p)]
    cs=[0.0 for i in range(s)]
    #a=[[0.0 for i in range(n)] for j in range(2)]
    a=gauss_legendre_coefficients(-1.0,1.0,n)
    rn=a[0]
    cn=a[1]
    a=gauss_legendre_coefficients(-1.0,1.0,m)
    rm=a[0]
    cm=a[1]
    a=gauss_legendre_coefficients(-1.0,1.0,p)
    rp=a[0]
    cp=a[1]
    a=gauss_legendre_coefficients(-1.0,1.0,s)
    rs=a[0]
    cs=a[1]
    x=[0.0 for i in range(4)]
    h1=(b1-a1)/2.0
    h2=(b1+a1)/2.0
    J=0.0
    for i in range(m):
        JX=0.0
        x[0]=h1*rm[i]+h2
        d1=fd(x[0])
        c1=fc(x[0])
        k1=(d1-c1)/2.0
        k2=(d1+c1)/2.0
```

```
        for j in range(n):
            JY=0.0
            x[1]=k1*rn[j]+k2;
            beta1=fbeta(x);
            alfa1=falfa(x);
            l1=(beta1-alfa1)/2.0;
            l2=(beta1+alfa1)/2.0;
            for k in range(p):
                    JZ=0.0
                    x[2]=l1*rp[k]+l2
                    teta1=fteta(x)
                    gama1=fgama(x)
                    m1=(teta1-gama1)/2.0
                    m2=(teta1+gama1)/2.0
              for h in range(s):
                x[3]=m1*rs[h]+m2
                Q=f(x)
                JZ=JZ+cs[h]*Q
              JY=JY+cp[k]*m1*JZ
            JX=JX+cn[j]*l1*JY
        J=J+cm[i]*k1*JX
    J=h1*J
    return J

h=1.0
R1=1.0
R2=1.0

a=0.0
b=R1
fc=lambda x:0.0
fd=lambda x:R2
falfa=lambda x:0.0
fbeta=lambda x:2.0*pi
fgama=lambda x:0.0
fteta=lambda x:2.0*pi
n=60
m=60
p=60
s=60

I =I(f,a,b,fc,fd,falfa,fbeta,fgama,fteta,n,m,p,s)
I1=F12(h,R1,R2)
print("F_A1_A2 = ",I,"Numerical four directional Gauss-Legendre integration result")
print("F_A1_A2 = ",I1,"Analytical solution")
```

```
runfile('D:/okul/SCO1/#integral_5D_GL2.py', wdir='D:/okul/SCO1')
F_A1_A2 =  0.3819660112501026 Numerical four directional Gauss-Legendre integration result
F_A1_A2 =  0.3819660112501051 Analytical solution
```

DIRECT INTEGRATION OF DATA

If integration of a numerical data set is given two approches and be adapted for this process. The first one is to curve fit a function first and then integrate the fitted function. The second one is to apply an integration equation like Newton-Cotes equation directly to the data to achive integrated result without an intermediate process.

As an example to curve fitting integration a cubic spline integration will be described here. If a cubic polynomial of the following form is given as an interpolation formula:

$$s_k(x) = a_k(x - x_k) + b_k(x_{k+1} - x) + [(x - x_k)^3 c_{k+1} + (x_{k+1} - x)^3 c_k] / (6h_k) \quad 1 \le k \le n \quad (10.6 - 14)$$

Where x_k and y_k are data points. Derivatives of equation becomes

$$s'_k(x) = a_k - b_k + [(x - x_k)^2 c_{k+1} - (x_{k+1} - x)^2 c_k] / h_k \quad 1 \le k \le n \quad (10.6\text{-}15)$$

$$s''_k(x) = [(x - x_k) c_{k+1} - (x_{k+1} - x) c_k] / h_k \quad 1 \le k \le n$$

a_k ve b_k coefficients can be expressed as a function of c_k

$$b_k = \frac{[6y_k - h_k c_k]}{6h_k} \qquad 1 \le k \le n \qquad (10.6 - 16)$$

$$a_k = \frac{[6y_{k+1} - h_k^2 c_{k+1}]}{6h_k} \qquad 1 \le k \le n \qquad (10.6-17)$$

In this case only c_k terms left in the system of equations to be solved.

$$h_{k-1}c_{k-1} + 2(h_{k-1} - h_k)c_{k-1} + h_k c_{k+1} c_{k+1} = 6\left[\frac{y_{k+1} - y_k}{h_k} - \frac{y_k - y_{k-1}}{h_{k-1}}\right] \qquad 1 \le k \le n \qquad (10.6-18)$$

This system of equation has only n-2 terms to be solved. By making definition

$w_k = \frac{y_{k+1} - y_k}{h_k}$, $\quad 1 \le$ k $\le$ n (10.6-19)

System of equation becomes

$$\begin{bmatrix} 1 & & & & & & & \\ h_1 & 2(h_1+h_2) & h_2 & & & & & \\ & h_2 & 2(h_2+h_3) & & & & & \\ & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ & & & \cdots & 2(h_{n-3}+h_{n-2}) & h_{n-2} & & \\ & & & & h_{n-2} & 2(h_{n-2}+h_{n-1}) & h_{n-1} & \\ & & & & & & 1 & \end{bmatrix} \begin{Bmatrix} c_0 \\ c_1 \\ c_2 \\ \cdots \\ c_{n-2} \\ c_{n-1} \\ c_n \end{Bmatrix} = \begin{Bmatrix} A \\ 6(w_2 - w_1) \\ 6(w_3 - w_2) \\ \cdots \\ 6(w_{n-2} - w_{n-3}) \\ 6(w_{n-1} - w_{n-2}) \\ B \end{Bmatrix} \quad (10.6-20)$$

Where A and B are the second derivative end conditions. A and B should be defined by user. Another important property of the above matrix is that it is a band matrix, therefore less amount of calculation is required to solve it (by using band matrix algorithms such as Thomas algorithm) compare to gauss elimination type of solution methods.

Integration of the equation could be described with the equation

$$\int_a^b S_k(x)dx = \sum_{x=a}^{b} a_k \frac{(x-x_k)^2}{2} - b_k \frac{(x_{k+1}-x)^2}{2} + c_k \frac{(x-x_k)^4}{24h_k} - d_k \frac{(x_{k+1}-x)^4}{24h_k}$$

An example program using this approach is given as:

Note that data is created from the function $f(x) = x^2$ and integral is equivalent to

$$I = \int_{x=0}^{10} x^2 dx = \left.\frac{x^3}{3}\right|_0^{100} = 333.333$$

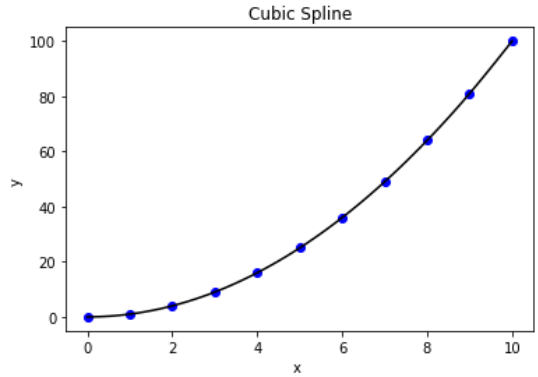
```
 from math import *
import numpy as np;
import matplotlib.pyplot as plt;
import os;

def thomas(f,e,g,r):
    n=len(f);
    x=[0.0 for i in range(n)];
    for k in range(0,n): e[k]=e[k]/f[k-1];f[k]=f[k]-e[k]*g[k-1];
    for k in range(1,n): r[k]=r[k]-e[k]*r[k-1];
    x[n-1]=r[n-1]/f[n-1];
    for k in range(n-2,-1,-1):  x[k]=(r[k]-g[k]*x[k+1])/f[k];
    return x;

def cubic_spline(xi,yi,c0,cn):
    n=len(xi);
    h=[0.0 for i in range(n)];
    w=[0.0 for i in range(n)];
    f=[0.0 for i in range(n)];
    e=[0.0 for i in range(n)];
    g=[0.0 for i in range(n)];
    d=[0.0 for i in range(n)];
    x=[0.0 for i in range(n)];
    S= [[0 for x in range(n)] for y in range(4)];
    for k in range(0,n-1):
        h[k]=xi[k+1]-xi[k];
        w[k]=(yi[k+1]-yi[k])/h[k];
    d[0]=c0;
    d[n-1]=cn;
    for k in range(0,n-1):
       d[k]=6.0*(w[k]-w[k-1]);
    f[0]=1.0;
    f[n-1]=1.0;
    g[0]=0.0;
```

```
    g[n-1]=0.0;
    e[0]=0.0;
    e[n-1]=0.0;
    for k in range(0,n-1):
        f[k]=2.0*(h[k]+h[k-1]);e[k]=h[k-1];g[k]=h[k];
    S[2]=thomas(f,e,g,d);
    S[3]=xi;
    for k in range(0,n-1):
       S[0][k]=(6.*yi[k+1]-h[k]*h[k]*S[2][k+1])/(6.0*h[k]);
       S[1][k]=(6.*yi[k]-h[k]*h[k]*S[2][k])/(6.0*h[k]);
    return S;


def funcSpline(S,x):
    n=len(S[0]);
    xx1=0;
    xx2=0;
    y=0;
    hk=0;
    for k in range(0,n-1):
       if S[3][k]<=x and x<=S[3][k+1]:
           hk=(S[3][k+1]-S[3][k]);
           xx1=(x-S[3][k]);
           xx2=(S[3][k+1]-x);
           y=S[0][k]*xx1+S[1][k]*xx2+(xx1*xx1*xx1*S[2][k+1]+xx2*xx2*xx2*S[2][k])/(6.0*hk);
           break;
    if y==0 and S[3][n-2]<=x:
        k=n-2;
        hk=(S[3][k+1]-S[3][k]);
        xx1=(x-S[3][k]);
        xx2=(S[3][k+1]-x);
        y=S[0][k]*xx1+S[1][k]*xx2+(xx1*xx1*xx1*S[2][k+1]+xx2*xx2*xx2*S[2][k])/(6.0*hk);
    return y;

def listSpline(xi,yi,numberofmidpoints,c0,cn):
    # numberofmidpoints :  in x--o--o--x--o--o--x chain if x's are esxperimental points
                # numberofmidpoints is 2
    n=len(xi);
    nn=(n-1)*(numberofmidpoints+1)+1;
    z=[[0 for x in range(nn)] for y in range(2)];
    S= [[0 for x in range(nn)] for y in range(4)];
    S=cubic_spline(xi,yi,c0,cn);
    dx=0;
    k=0;
    for i in range(0,n-1):
       z[0][k]=xi[i];z[1][k]=funcSpline(S,z[0][k]);k=k+1;
       for j in range(numberofmidpoints):
          dx=(xi[i+1]-xi[i])/(numberofmidpoints+1.0);
          z[0][k]=z[0][k-1]+dx;z[1][k]=funcSpline(S,z[0][k]);k=k+1;
    z[0][k]=xi[i+1];z[1][k]=funcSpline(S,z[0][k]);
    return z;
# Integration formulas
# ===================
def intSpline(S,a,b):
    # Spline coefficients a,b integral limits
    # cubic spline integration
    n=len(S[0])
    xx1=0.0
    xx2=0.0
    y1=0.0
    y2=0.0
    toplam=0.0
    for k in range(n-1):
       hk=(S[3][k+1]-S[3][k]);
       if a>S[3][k+1]:
          toplam=0.0
       elif S[3][k]<=a and a<=S[3][k+1] and b>S[3][k+1]:
          xx1=(a-S[3][k])
          xx2=(S[3][k+1]-a)
          y1=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk)
          xx1=hk
          xx2=0
          y2=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk)
          toplam=toplam+(y2-y1);
       elif S[3][k]<=a and a<=S[3][k+1] and S[3][k]<=b and b<=S[3][k+1]:
          xx1=(a-S[3][k])
```

```
            xx2=(S[3][k+1]-a)
            y1=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk)
            xx1=(b-S[3][k])
            xx2=(S[3][k+1]-b)
            y2=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk)
            toplam=toplam+(y2-y1)

        elif a<S[3][k] and  b>=S[3][k+1]:
            xx1=0.0
            xx2=hk
            y1=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk)
            xx1=hk
            xx2=0.0
            y2=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk)
            toplam=toplam+(y2-y1)
        elif S[3][k]<=b and b<=S[3][k+1]:
            xx1=0
            xx2=hk
            y1=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk)
            xx1=(b-S[3][k])
            xx2=(S[3][k+1]-b)
            y2=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk)
            toplam+=(y2-y1)
        else:
            break
    return toplam


def read_array2D(d):
    path = d
    file1 = open(path,'r')
    file1.seek(0,2)  #jumps to the end
    endLocation=file1.tell()
    file1.seek(0) #jumps to the start
    nn=0
    m=0
    while True:
        line=file1.readline()
        ee=line.split()
        nn+=1
        if  file1.tell()==endLocation:
            break
    m=len(ee)
    a=[[0.0 for i in range(nn)] for j in range(m)]
    file1.seek(0) #jumps to the start
    for i in range(nn):
        line=file1.readline()
        ee=line.split()
        for j in range(m):
            a[j][i]=float(ee[j])
    file1.close()
    return a

a=read_array2D('a.txt')
x=a[0];
print("x=",x)
y=a[1];
print("y=",y)
#x=[0.0,1.0,2.0,3.0,4.0,5.0,6.0,7.0,8.0,9.0,10.0]
#y=[0.0,1.0,4.0,9.0,16.0,25.0,36.0,49.0,64.0,81.0,100.0]
c0=2.0
cn=2.0
S=cubic_spline(x,y,c0,cn)
a=0
b=10
I=intSpline(S,a,b)
print("Integral=",I)
Z=listSpline(x,y,5,c0,c1)
plt.title('Cubic Spline')
plt.xlabel('x ')
plt.ylabel('y ');
plt.plot(x,y,'bo',Z[0],Z[1],'k');
```

```
runfile('E:/okul/SCO1/cubic_spline_integral.py', wdir='E:/okul/SCO1')
x= [0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0]
y= [0.0, 1.0, 4.0, 9.0, 16.0, 25.0, 36.0, 49.0, 64.0, 81.0, 100.0]
```

Integral= 333.33333333333337



MULTIDIMENSIONAL INTEGRATION BY USING CONTOUR INTEGRATION

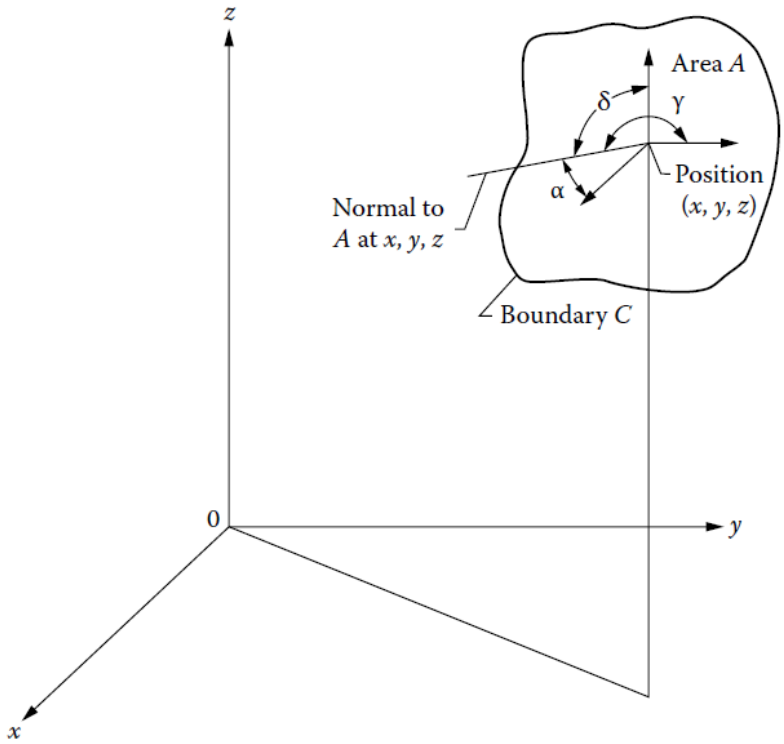
Stoke's theorem gives a way to reduce multiple integration to single integration. Consider the area A in the figure with countour boundary C. It is assumed that boundary is continuous or picewise-continuous. Let us also consider an arbitrary point on the surface as P(x,y,z). Normal to this point can be written as

Vector pointing from original to an arbitrary point $P(\mathrm{x},\mathrm{y},\mathrm{z}) = [x\vec{\imath} + y\vec{\jmath} + z\vec{k}]$

Surface normal for area: $n = [l\vec{\imath} + m\vec{\jmath} + n\vec{k}] = [\cos(\alpha)\vec{\imath} + \cos(\beta)\vec{\jmath} + \cos(\gamma)\vec{k}]$

Where l,m,n are direction cosines of the surface normal vector, α, β and γ are angles of normal with x,y and z axeses respectively. And also let P,Q, and W be any twice differentiable functions of (x,y,z). Stokes' theorem in three dimensions states that

$$\oint_C (Pdx + Qdy + Wdz) = \int_A \left[\left(\frac{\partial W}{\partial y} - \frac{\partial Q}{\partial z}\right) l + \left(\frac{\partial P}{\partial z} - \frac{\partial W}{\partial x}\right) m + \left(\frac{\partial Q}{\partial x} - \frac{\partial P}{\partial y}\right) n\right] dA$$



Now this theory can be apply to solve configuration factor problems. Let us consider **configuration factors between a differential area and a finite area**

$$\cos(\theta_i) = [(x_j - x_i)l_i + (y_j - y_i)m_i + (z_j - z_i)n_i]/R$$

$$\cos(\theta_j) = \left[(x_i - x_j)l_j + (y_i - y_j)m_j + (z_i - z_j)n_j\right]/R$$

And

$$R^2 = (x_j - x_i)^2 + (y_j - y_i)^2 + (z_j - z_i)^2$$

$$R = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2 + (z_j - z_i)^2}$$

$$F_{dA_i - A_j} = \int_{A_2} \frac{\cos(\theta_i)\cos(\theta_j)}{\pi R^2} dA_j$$

This particular integral is called shape factor. It is an important part of radiation heat transfer calculations. For further details of radiation shape factor, please refer to radştşon heat transfer books. Integral formula converted to contour integration as:

$$F_{dA_i - A_j} = \frac{1}{\pi} \int_{A_j} \frac{\left[(x_j - x_i)l_i + (y_j - y_i)m_i + (z_j - z_i)n_i\right]\left[(x_i - x_j)l_j + (y_i - y_j)m_j + (z_i - z_j)n_j\right]}{R^4} dA_j$$

Now let

$$f = \frac{\left[(x_j - x_i)l_i + (y_j - y_i)m_i + (z_j - z_i)n_i\right]}{\pi R^4}$$

then equation becomes

$$F_{dA_i - A_j} = \int_{A_j} \left[(x_i - x_j)fl_j + (y_i - y_j)fm_j + (z_i - z_j)fn_j\right] dA_j$$

Comparison of this equation with Stokes' equation gives

$$\left(\frac{\partial W}{\partial y} - \frac{\partial Q}{\partial z}\right) = (x_i - x_j)f$$

$$\left(\frac{\partial P}{\partial z} - \frac{\partial W}{\partial x}\right) = (y_i - y_j)f$$

$$\left(\frac{\partial Q}{\partial x} - \frac{\partial P}{\partial y}\right) = (z_i - z_j)f$$

Useful solution of these equations[98,99] are:

$$P = \frac{-m_i(z_j - z_i) + n_i(y_j - y_i)}{2\pi R^2}$$

$$Q = \frac{l_i(z_j - z_i) - n_i(x_j - x_i)}{2\pi R^2}$$

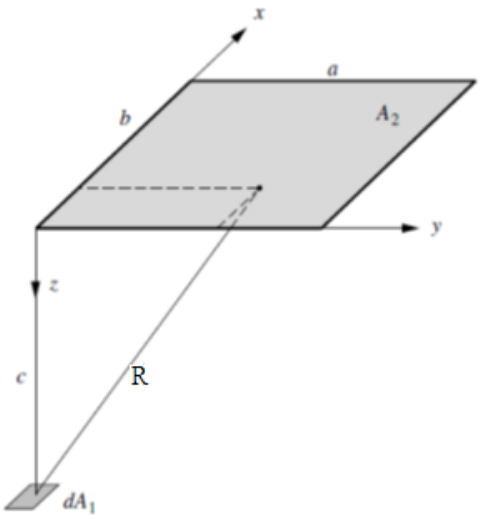
$$W = \frac{-l_i(y_j - y_i) + m_i(x_j - x_i)}{2\pi R^2}$$

$$F_{dA_i - A_j} = \oint_{C_j} (P dx_j + Q dy_j + W dz_j)$$

$$F_{dA_i - A_j} = \oint_{C_j} \left(\frac{-m_i(z_j - z_i) + n_i(y_j - y_i)}{2\pi R^2} dx_j + \frac{l_i(z_j - z_i) - n_i(x_j - x_i)}{2\pi R^2} dy_j + \frac{-l_i(y_j - y_i) + m_i(x_j - x_i)}{2\pi R^2} dz_j\right)$$

$$F_{dA_i - A_j} = \frac{l_i}{2\pi} \oint_{C_j} \frac{(z_j - z_i)dy_j - (y_j - y_i)dz_j}{R^2} + \frac{m_i}{2\pi} \oint_{C_j} \frac{(x_j - x_i)dz_j - (z_j - z_i)dx_j}{R^2} + \frac{n_i}{2\pi} \oint_{C_j} \frac{(y_j - y_i)dx_j - (x_j - x_i)dy_j}{R^2}$$

PROBLEM:



The shape in the left side is given. By using contour integration method find the shape factor analytically and numerically

$l_1 = m_1 = 0 \; \overrightarrow{n_1} = -\vec{k} \quad n_1 = -1 \quad F_{dA_1-A_2} = \frac{-1}{2\pi}\oint_{C_2} \frac{ydx-xdy}{R^2} \quad R^2 = x^2 + y^2 + c^2$

$$F_{dA_1-A_2} = \frac{-1}{2\pi}\left[\int_{x=0}^{b} \frac{y}{R^2}dx\Big|_{y=0} + \int_{y=0}^{a} \frac{-x}{R^2}dy\Big|_{x=b} + \int_{x=b}^{0} \frac{y}{R^2}dx\Big|_{y=a} + \int_{x=a}^{0} \frac{y}{R^2}dy\Big|_{x=0}\right]$$

For the configuration $F_{dA_1-A_2}$ given as

$$F_{dA_1-A_2} = \frac{1}{2\pi}\left(\int_{y=0}^{a} \frac{bdy}{b^2+y^2+c^2} + \int_{x=0}^{b} \frac{adx}{x^2+a^2+c^2}\right)$$

Analytical solution gives

$$F_{dA_1-A_2} = \frac{1}{2\pi}\left(\frac{b}{\sqrt{b^2+c^2}}tan^{-1}\left(\frac{a}{\sqrt{b^2+c^2}}\right) + \frac{a}{\sqrt{a^2+c^2}}tan^{-1}\left(\frac{b}{\sqrt{a^2+c^2}}\right)\right)$$

Calculate first equation by using numerical integration and compare it with the analytical result
a= 1 m b= 1 m and c= 1 m

```
from math import *
from if_x import *;

class fx(if_x):
   def __init__(self, ai,bi,ci):
      self.a = ai
      self.b=bi
      self.c=ci
   def func(self,x):
      y=1.0/(2.0*pi)*a/(x*x+a*a+c*c)
      return y

class fy(if_x):
   def __init__(self, ai,bi,ci):
      self.a = ai
      self.b=bi
      self.c=ci
   def func(self,y):
      y=1.0/(2.0*pi)*b/(b*b+y*y+c*c);
      return y

def gauss_legendre_coefficients(x1,x2,n):
   #calculates legendre gauss-coefficients as coefficients of the integral
   #for n terms
      eps=3e-15
      m=int((n+1.0)/2.0)
      xm=0.5*(x2+x1)
```

```
    xl=0.5*(x2-x1)
    nmax=100
    a=[[0.0 for i in range(n)] for j in range(2)]
    for i in range(1,m+1):
      z=cos(pi*((i-0.25)/(n+0.5)))
      for ii in range(nmax):
        #while abs(z-z1) > eps:
        p1=1.0;
        p2=0.0;
        for j in range(1,n+1):
          p3=p2
          p2=p1
          p1=(float)(((2.0*j-1.0)*z*p2-(j-1.0)*p3)/j)
        pp=(float)(n*(z*p1-p2)/(z*z-1.0))
        z1=z;
        z=z1-p1/pp
        if abs(z-z1) < eps: break
      a[0][i-1]=xm-xl*z
      a[0][n-i]=xm+xl*z
      a[1][i-1]=2.0*xl/((1.0-z*z)*pp*pp)
      a[1][n-i]=a[1][i-1]
    return a

def F(a,b,c):
  A=b/sqrt(b*b+c*c)
  B=a/sqrt(b*b+c*c)
  C=a/sqrt(a*a+c*c)
  D=b/sqrt(a*a+c*c)
  f=1.0/(2.0*pi)*(A*atan(B)+C*atan(D))
  return f


def integral(f,x1,x2,n):
  #integral func(x)dx
  #integral of a function by using gauss-legendre quadrature
  #between x1 and x2
  a=gauss_legendre_coefficients(x1,x2,n)
  z=0
  for i in range(n):
    z=z+a[1][i]*f.func(a[0][i])
  return z

a=1.0
b=1.0
c=1.0
x1=fx(a,b,c)
y1=fy(a,b,c)
I1=integral(y1,0.0,a,20)+integral(x1,0.0,b,20)
print("I1 (contour integral) : ",I1)
print("F (analytical integral solutionl) : ",F(a,b,c));
```

```
runfile('D:/okul/SCO1/integral_contour_GL.py', wdir='D:/okul/SCO1')
Reloaded modules: if_x
I1 (contour integral) :  0.13853160599489273
F (analytical integral solution) :  0.13853160599489298
```

Contour integration for configuration factor between finite areas

In order to calculate contour integration for finite areas, the previously find contour integration between differential and finite area eqution integrated for area

$$A_i F_{A_i-A_j} = A_j F_{A_j-A_i} = \int_{A_i} F_{dA_i-A_j}\, dA_i$$

$$A_i F_{A_i-A_j} = A_j F_{A_j-A_i} = \frac{1}{2\pi}\oint_{C_j}\int_{A_i}\left[\frac{-m_i(z_j-z_i)+n_i(y_j-y_i)}{R^2}dx_j + \frac{l_i(z_j-z_i)-n_i(x_j-x_i)}{R^2}dy_j + \frac{-l_i(y_j-y_i)+m_i(x_j-x_i)}{R^2}dz_j)\right] dA_i$$

$$A_iF_{A_i-A_j} = A_jF_{A_j-A_i} = \frac{1}{2\pi}\oint_{C_j}\left[\left(\int_{A_i}\frac{-m_i(z_j-z_i)+n_i(y_j-y_i)}{R^2}dA_i\right)dx_j + \left(\int_{A_i}\frac{l_i(z_j-z_i)-n_i(x_j-x_i)}{R^2}dA_i\right)dy_j + \left(\int_{A_i}\frac{-l_i(y_j-y_i)+m_i(x_j-x_i)}{R^2}dA_i\right)dz_j)\right]$$

where the integrals have been rearranged, and dx_j, dy_j, and dz_j have been factored out since they are independent of the area integration over $A1$. Stokes' theorem is applied in turn to each of the three area integrals. Consider area integral

$$\int_{A_i}\frac{-m_i(z_j-z_i)+n_i(y_j-y_i)}{R^2}dA_i$$

Integral in stokes theorem gives

$$\left(\frac{\partial W}{\partial y}-\frac{\partial Q}{\partial z}\right) = 0$$

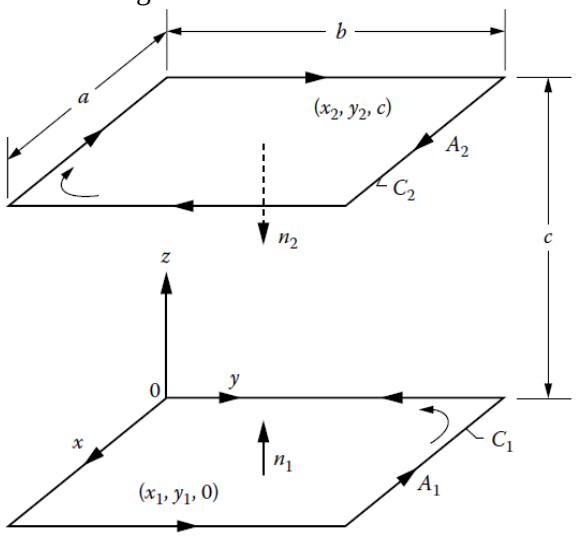
$$\left(\frac{\partial P}{\partial z}-\frac{\partial W}{\partial x}\right) = \frac{-(z_j-z_i)}{R^2}$$

$$\left(\frac{\partial Q}{\partial x}-\frac{\partial P}{\partial y}\right) = \frac{(y_j-y_i)}{R^2}$$

Solution of this set of partial differential equations is P=ln(R) Q=0 and W=0 and

$$A_iF_{A_i-A_j} = A_jF_{A_j-A_i} = \frac{1}{2\pi}\oint_{C_1}\oint_{C_2}\left[\ln(R)\,dx_jdx_i + \ln(R)\,dy_jdy_i + \ln(R)\,dz_jdz_i\right]$$

PROBLEM: Find configuration factor by using contour integration for two parallel rectangular shown in the figure



$$F_{A_1-A_2} = F_{A_2-A_i} = \frac{1}{2\pi ab}\oint_{C_1}\oint_{C_2}\begin{bmatrix}\ln\left([(x_2-x_1)^2+(y_2-y_1)^2+c^2]^{\frac{1}{2}}\right)dx_2dx_1 + \\ \ln\left([(x_2-x_1)^2+(y_2-y_1)^2+c^2]^{1/2}\right)dy_2dy_1\end{bmatrix}$$

Opening contour integral for area 2

$$F_{A_1-A_2} = F_{A_2-A_i}$$

$$= \frac{1}{2\pi ab}\oint_{C_1}\left[\begin{array}{l}\left(\int_{x_2=0}^{a}\ln\left([(x_2-x_1)^2+(b-y_1)^2+c^2]^{\frac{1}{2}}\right)dx_2\right)dx_1 + \left(\int_{x_2=a}^{0}\ln\left([(x_2-x_1)^2+(0-y_1)^2+c^2]^{\frac{1}{2}}\right)dx_2\right)dx_1 + \\ \left(\int_{y_2=0}^{b}\ln\left([(0-x_1)^2+(y_2-y_1)^2+c^2]^{1/2}\right)dy_2\right)dy_1 + \left(\int_{y_2=b}^{0}\ln\left([(a-x_1)^2+(y_2-y_1)^2+c^2]^{1/2}\right)dy_2\right)dy_1\end{array}\right]$$

Opening contour integral for area 2

$F_{A_1-A_2} = F_{A_2-A_1}$

$$= \frac{1}{2\pi ab}\left[\begin{array}{l}\int_{x_1=0}^{a}\left(\int_{x_2=0}^{a}\ln\left([(x_2-x_1)^2+(b-0)^2+c^2]^{\frac{1}{2}}\right)dx_2\right)dx_1+\int_{x_1=0}^{a}\left(\int_{x_2=a}^{0}\ln\left([(x_2-x_1)^2+(0-0)^2+c^2]^{\frac{1}{2}}\right)dx_2\right)dx_1+\\ \int_{x_1=a}^{0}\left(\int_{x_2=0}^{a}\ln\left([(x_2-x_1)^2+(b-b)^2+c^2]^{\frac{1}{2}}\right)dx_2\right)dx_1+\int_{x_{1=a}}^{0}\left(\int_{x_2=a}^{0}\ln\left([(x_2-x_1)^2+(0-b)^2+c^2]^{\frac{1}{2}}\right)dx_2\right)dx_1+\\ \int_{y_1=0}^{b}\left(\int_{y_2=0}^{b}\ln\left([(0-a)^2+(y_2-y_1)^2+c^2]^{\frac{1}{2}}\right)dy_2\right)dy_1+\int_{y_1=0}^{b}\left(\int_{y_2=b}^{0}\ln\left([(a-a)^2+(y_2-y_1)^2+c^2]^{\frac{1}{2}}\right)dy_2\right)dy_1+\\ \int_{y_1=b}^{0}\left(\int_{y_2=0}^{b}\ln\left([(0-0)^2+(y_2-y_1)^2+c^2]^{1/2}\right)dy_2\right)dy_1+\int_{y_1=b}^{0}\left(\int_{y_2=b}^{0}\ln\left([(a-0)^2+(y_2-y_1)^2+c^2]^{1/2}\right)dy_2\right)dy_1\end{array}\right]$$

$$F_{A_1-A_2} = F_{A_2-A_1} = \frac{1}{\pi ab}\left[\int_{x_1=0}^{a}\left(\int_{x_2=0}^{a}\ln\left(\frac{[(x_2-x_1)^2+b^2+c^2]^{\frac{1}{2}}}{[(x_2-x_1)^2+c^2]^{\frac{1}{2}}}\right)dx_2\right)dx_1 + \int_{y_1=0}^{b}\left(\int_{y_2=0}^{b}\ln\left(\frac{[a^2+(y_2-y_1)^2+c^2]^{\frac{1}{2}}}{[(y_2-y_1)^2+c^2]^{1/2}}\right)dx_2\right)dx_1\right]$$

Analytical solution of the equation:

$X = \frac{a}{c} \quad Y = \frac{b}{c}$

$F_{A_1-A_2} = F_{A_2-A_1} = \frac{2}{\pi XY}\left\{ln\left[\frac{(1+X^2)(1+Y^2)}{(1+X^2+Y^2)}\right]^{1/2} + X\sqrt{1+Y^2}tan^{-1}\left(\frac{X}{\sqrt{1+Y^2}}\right) + Y\sqrt{1+X^2}tan^{-1}\left(\frac{Y}{\sqrt{1+X^2}}\right) - Xtan^{-1}(X) - Ytan^{-1}(Y)\right\}$

Gauss random distribution function

When n is a big number

Average: $\mu = \frac{\sum_{i=1}^{n} x_i}{n}$

Standard deviation: $\sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i-\mu)^2}{n}}$

When n is relatively small number

Average: $\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}$

Standard deviation: $s = \sqrt{\frac{\sum_{i=1}^{n}(x_i-\bar{x})^2}{n-1}}$

Laplace-Gauss distribution:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

Normal distribution $(\sigma = 1 \quad \mu = 0)$

$$\varphi(x) = \frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}x^2}$$

Cumulative distribution function

The Laplace distribution is easy to integrate (if one distinguishes two symmetric cases) due to the use of the absolute value function. Its cumulative distribution function is as follows:

$$\phi(x) = \int_{x=-\infty}^{x} \varphi(t)dt = \frac{1}{\sqrt{2\pi}} \int_{x=-\infty}^{x} e^{-\frac{1}{2}t^2} dt$$

$$\phi(\infty) = \int_{x=-\infty}^{\infty} \varphi(t)dt = \frac{1}{\sqrt{2\pi}} \int_{x=-\infty}^{\infty} e^{-\frac{1}{2}t^2} dt = 1$$

Error function

$$erf(x) = \frac{1}{\sqrt{\pi}} \int_{x=0}^{\infty} e^{t^2} dt$$

MacLaurin series opening of error function

$$\mathrm{erf(x)} = \frac{2}{\sqrt{\pi}} \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{n!\,(2n+1)}$$

Error function- Cumulative distribution function relation

$$\phi(x) = \frac{1}{2}\left[1 + erf\left(\frac{x}{\sqrt{2}}\right)\right]$$

Laplace-Gauss distribution function:

$$F(x) = \phi\left(\frac{x-\mu}{\sigma}\right) = \frac{1}{2}\left[1 + erf\left(\frac{x-\mu}{\sigma\sqrt{2}}\right)\right]$$

```
# Gauss - Legendre integral 10 coefficients
# python version

from math import *

def fi(x):
   y= 1.0/(2.0*sqrt(pi))*exp(-0.5*x*x)
   return y

def erf1(x):
   plusminus=1
   power=x
   factorial=1
   total=0
   for n in range(100):
      k=2*n+1
      total+=plusminus*power/(factorial*k)
      plusminus*=-1
      power*=x*x
      factorial*=(n+1)
   return total*2.0/sqrt(pi)

def g(x):
   return 2.0/sqrt(pi) *exp(-x*x)

def erf2(x):
   return  integral(g,0,x)
  #gauss inetgral
def integral(f,a,b):
  #integral f1(x)dx
  #integral of a function by using gauss-legendre quadrature
  #coefficients are pre-calculated for 10 terms for [-1,1]
  #band then utilises variable transform
  r=[-0.973906528517171,-0.865063366688984,-0.679409568299024,-0.433395394129247,-
0.148874338981631,0.148874338981631,0.433395394129247,0.679409568299024,0.865063366688984,0.973906528517171]
```

```
c=[0.066671344308684,0.149451349150580,0.219086362515982,0.269266719309996,0.295524224714752,0.295524224714752,0.269266719309996,0.219086362515982,0.149451349150580,0.066671344308684]
  n=10
  z=0
  k1=(b-a)/2.0;
  k2=(b+a)/2.0;
  for i in range(n):
    x=k2+k1*r[i]
    y=f(x)
    z=z+k1*c[i]*y
  return z
# cumulative distribution functions
def fi1(x):
    y=0.5*(1.0+erf1(x/sqrt(2.0)))
    return y
def fi2(x):
    y=0.5*(1.0+erf2(x/sqrt(2.0)))
    return y
def fi3(x):
    y=0.5*(1.0+erf(x/sqrt(2.0)))
    return y

x=float(input("x="))
e1=erf1(x)
e2=erf2(x)
e3=erf(x)
print("erf1",e1,"erf2",e2,"erf3",e3)
fi1=fi1(x)
fi2=fi2(x)
fi3=fi3(x)
print("fi1",fi1,"fi2",fi2,"fi3",fi3)
```

```
runfile('E:/okul/SCO1/Laplace_gauss_didstribution.py', wdir='E:/okul/SCO1')

x=0.1
erf1 0.11246291601828491 erf2 0.11246291601828423 erf3 0.1124629160182849
fi1 0.539827837277029 fi2 0.5398278372770288 fi3 0.539827837277029
```

PROBLEMS (INTEGRATION & DERIVATION)

PROBLEM 1

of $y=f(x)=1/(1+x^2)$ **fonction between limits –1 ile 1 with 3 points Gauss-Legendre integration Formula**

Note: Required roots and coefficients :

coefficients	Roots
c_0 **0.555555555555553**	x_0 **0.774596669241483**
c_1 **0.888888888888889**	x_1 **0.000000000000000**
c_2 **0.555555555555553**	x_2 **0.774596669241483**

PROBLEM 2

$$\int_{-3}^{3} \frac{2}{1+2x^2}\,dx$$

solve with bole rule. Use n=2

Note: Bole rule for n=1

$$I=(b-a)[7f(x_0)+32f(x_1)+12f(x_2)+32f(x_3)+7f(x_4)]$$

Step size h=(b-a)/n

PROBLEM 3

$$f_{0-\lambda} = \frac{c_1}{\sigma} \int_{x=0}^{\lambda T} \frac{dx}{x^5 \left[exp\left(\frac{c_2}{x}\right) - 1 \right]}$$

Function is known as black body radiation shape function. The values of coefficients in the equation are:

c_1 =3.743 x10^8 W.μm^4/m^2, c_2 =1.4387x10^4 μm.K, σ=5.67x10^{-8} W/(m^2.K^4).

Use Gauss-Legendre integration and for a giving λT calculate black body radiation shape factor. Create a table for the values of blackbody radiation shape factor from 5000 to infinity(?)

PROBLEM 4

$$I = \int_{x=0}^{1} \int_{y=0}^{1} \frac{dydx}{(1+y)^2}$$

Solve the above integration by using analytical methods and Gauss-Legendre integration

PROBLEM 5

Solve the integration

$$I = \int_{x=0}^{1} \frac{1}{1+x^2} dx$$

a) by analytical methods

b) by Trapezoidal rule with n=2(divide region into 2 parts) and calculate % error

c) Trapezoidal rule with n=2(divide region into 2 parts) and calculate % error

PROBLEM 6

Solve the integration

$$I = \int_{x=0}^{2} \frac{1}{\sqrt{1+x^2}} dx$$

By using Gauss-Legendre formulation

PROBLEM 7

For x=1

$$f(x) = ln\left(\frac{1}{1+x^2}\right)$$

a) Calculate analytical derivative
b) Calculate derivative with 3 points central difference formula with h=0.1
c) Calculate derivative with 3 points central difference formula with h=0.05
d) Calculate derivative with Richardson extrapolation formula with h1=0.5 and h2=0.25

PROBLEM 8

$$f(x) = \int_{-1}^{1} xexp(x)dx$$

Calculate the integration by

a) Trapezoidal rule integrali
b) Simpson 3/8 integration

c) Gauss-Legendre integration
d) Romberg integration

PROBLEM 9

$$f(x) = \int_0^1 sin(1 - x + x^2)dx$$

Calculate the integration by

e) Trapezoidal rule integrali
f) Simpson 3/8 integration
g) Gauss-Legendre integration
h) Romberg integration

PROBLEM 10

$$f(x) = \int_{-1}^{1} \frac{ln(1 + x^2)}{\sqrt{1 - x^2}} dx$$

Calculate the integration by

i) Trapezoidal rule integrali
j) Simpson 3/8 integration
k) Gauss-Legendre integration
l) Romberg integration

PROBLEM 11

X	0.99	2.10	3.22	4.40	5.70	7.12	8.01	8.37	9.32	9.98
f(x)	4.90	5.70	4.20	7.04	8.31	7.82	5.97	7.01	6.68	4.79

Data given in the above table. Find the integration by using trapezoidal rule.

PROBLEM 12

Find the derivative of data given in problem 11 at point x=5

PROBLEM 13

Depth(H) and velocity(U) profile of a channel is given in below table

x, m	0	2	4	6	8	10	12	14	16	18	20
H, m	0	1.8	2	4	4	6	4	3.6	3.4	2.8	0
U m/s	0	0.03	0.045	0.055	0.065	0.12	0.07	0.06	0.05	0.04	0

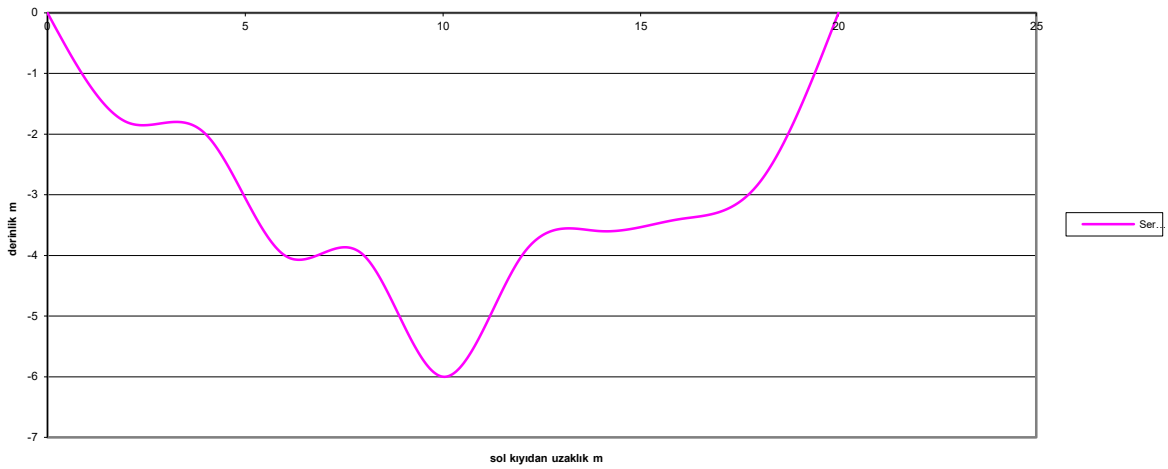
Cross sectional area of the channel can be calculated as :

$$A_c = \int_0^B H(x)dx$$

Flow rate of the channel will be

$$Q = \int_0^B U(x)H(x)dx$$

In this equations U is the velocity of the water measured at point x. Calculate cross sectional area and flow rate of the channel.



PROBLEM 14

Temperature profle of a solid body can be given as

$$\frac{dT}{dt} = -k(T - T_a)$$

In this equation T is the temperature, T_a is constant surface temperature.

A solid body with temperatures initially at T=90 C and, T_a =20 C put into a water container at temperature of T_a = 20 C. The temperature profile as a function of time is given at the table.

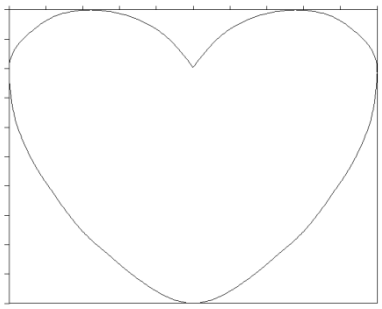
Zaman , t, s	0	5	10	15	20	25	30	35
Sicaklik T C	90	55	40	28	24	22	21.5	20.6

Calculate k thermal conductivity coefficient.

PROBLEM 15

Coordinate of a hard shape is given in the table. Find the area of the hard.

Xi	Yi	xi	yi	xi	yi	xi	Yi
241	127	331	157	215	246	149	120
258	107	323	178	199	232	160	106
283	96	312	197	184	217	175	97
307	97	298	217	170	197	199	96
322	106	283	232	159	178	224	107
333	120	267	246	151	157	241	127
334	138	241	258	148	138		



PROBLEM 16

$$\boldsymbol{I} = \int_{0.1}^{0.5} \int_{x^3}^{x^2} e^{y/x} dydx$$

Solve the integral by Gauss-Legendre method.

PROBLEM 17

$$\boldsymbol{I} = \int_{0.1}^{1} \int_{x}^{2x} (x^2 + y^2) dydx$$

Solve the equation with

a) 2 dimensional Gauss-Legendre integration
b) 2 dimensional simpson 1/3 integration
c) Develop an equation for 2 dimensional simpson. 3/8 integration and solve the integral

PROBLEM 18

$$\boldsymbol{I} = \int_{x=0}^{1} \int_{y=0}^{1} \frac{dydx}{(1+y)^2}$$

Solve the integral with Monte carlo method and check the results with Gauss-Legendre method.

PROBLEM 19

Show that equation below is correct

$$\pi = \int_{0}^{2} \sqrt{(4 - x^2)}\, dx$$

PROBLEM 20

$$I = \int_{0}^{1} e^x dx$$

Solve the integral for m=25,50,100,500,1000,10000,100000 with monte-carlo method and compare the result with the analytical solution.

PROBLEM 21

An angine of a vehicle with mass of 53400 kg and a velocity of v=30 m/is suddenly stopped at t=0. In this instant equation of the movement can be given as

$$5400v\frac{dv}{dx} = -8276v^2 - 2000$$

In the equation v(m/sn), is the velocity of the vehicle at time t. When velocity is 15 m/s calculate the total distance vehicle moved by using 4 point Gauss integration Formula .

PROBLEM 22

A vehicle is at complete stop. When it is started velocity of the vehicle is recorded for the first 5 minutes. Calculate the total distance vehicle moved by using trapezoidal rule as a direct data integration method.

t(minutes)	0	0,5	1,0	1,5	2,0	2,5	3,0	3,5	4,0	4,5	5,0
v (km/h)	0	4	7	11	15	21	25	30	32	35	40

Note : $\boldsymbol{v = \frac{dx}{dt}}$ $x - x_0 = \int_{t_0}^{t} v(t)dt$

PROBLEM 23

Upward velocity of a rocket can be calculated by using the formula

$$V = \frac{dz}{dt} = Uln\left(\frac{m_0}{m_0 - qt}\right) - gt$$

In this equation V is velocity, U is the exit velocity of burning gases from the rocket nozzle, m_o is the mass of the rocket at time t = 0 sn, q is fuel spending rate , g is garavitational acceleration (assume constant as, 9,8 m^2/s). If U = 2000 m/s, m_o = 150 000 kg, q = 2600 kg/s, Calculate the height rocket will climb in 30 seconds by using Romberg inntegration with k=4 [$O(h^8)$] order of error.

Note : $\boldsymbol{V = \frac{dH}{dt}}$ $H = \int_{t_{0=0}}^{t} V(t)dt$

PROBLEM 24

Calculate velocity and acceleration by using below data with

a) cental difference method [with $O(h^4)$ error level],
b) forward difference method [with $O(h^2)$ error level],
c) backward difference method [with $O(h^2)$ error level],

Time, t (s)	0	1	2	3	4	5	6	7	8
distance, x (m)	0	0.7	1.8	3.4	5.1	6.5	7.3	8.0	8.4

Note : Velocity $\boldsymbol{v = \frac{dx}{dt}}$Acceleration : $a = \frac{d^2x}{dt^2}$

PROBLEM 25

Integrate data by using trapezoidal rule

x	-3	-1	1	3	5	7	9	11
f(x)	1	-4	-9	2	4	2	6	-3

PROBLEM 26

In a river water height is measured in every 12.5 m distance. Water flow rate can be calculated by $Q = \int_{t_{0=0}}^{t} VdA$ integration. In this equation Q: water flow rate (m^3/s), V: water flow speed (m/s), A: cross-sectional area (m^2). If avarage water velocity is 0.4 m/s. Calculate water flow rate by using Simpson 1/3 rule

X(m)	0	12,5	25	37,5	50	62,5	75	87,5	100
f(X) m	11	7,5	5,5	0	1,5	4	5,5	8	11

PROBLEM 27

a)

$$\int_{0}^{10} \boldsymbol{f(x)dx}$$

calculate the integral value from the data given below

b) calculate the derivative of the function at x=3,4,5,6,7,8

X	0.99	2.10	3.22	4.40	5.70	7.12	8.01	8.37	9.32	9.98
f(x)	4.90	5.70	4.20	7.04	8.31	7.82	5.97	7.01	6.68	4.79

PROBLEM 28

By using Gauss-Legendre integral formulation calculate

$$I = \int_{-4}^{4} \frac{dx}{1+x^2}$$

integral for n=2,3,4,5 . Compare the result with analytical solution.

PROBLEM 29

By using **Adaptive** simpson 1/3 solve

$$I = \int_{-4}^{4} \frac{dx}{1+x^2}$$

integral with an error level of 1.0e-5

PROBLEM 30

By using Gauss-Chebychev integration equation solve the integral

$$I = \int_{-1}^{1} \frac{\cos(x)dx}{\sqrt{1-x^2}} = 2.40394$$

and compare the result with the exact value.

PROBLEM 31

By using trapezoidal rule calculate the integral

$$I = \int_{0}^{1} \frac{2dx}{2+\sin(10\pi x)} = 1.15470054$$

for n=2,4,8 ve 16

And then improve the result by using Richardson extrapolation Formula (Romberg integration).

PROBLEM 32

If velocity profile in a pipe is known, flow rate can be calculated as Q = ∫ v dA

In this equation v i the velocity of the fluid A is the cross-sectional area of the pipe. In a circular pipe A=(2 π r) and dA = (2 π r dr). If it is substituted into flow rate equation

Q =∫ v(2πr) dr is obtained. Where r is the radius. If the velocity distribution is given as a function of r as v = 3[1-(r/r$_0$)]$^{(1/7)}$ and if the radius of the pipe r$_0$= 0.1 m calculate mass flow rate. Select your own method.

PROBLEM 33

In the integrals below exact solutions are given. By using an adaptive integration method. Check out the results.

a)

$$I = \int_{0}^{1} \frac{4dx}{1+256(x-0.375)^2} = 0.7191938309210011$$

b)

$$I = \int_{0}^{1} x^{5/2}dx = \frac{2}{7}$$

c)

$$I = \int_0^1 x^{1/2} dx = \frac{2}{3}$$

d)

$$I = \int_0^1 \log(x) dx = -1$$

e)

$$I = \int_0^1 \log(|x - 0.7|) dx \,?\, 0.3\log(0.3) + 0.7\log(0.7) - 1.0$$

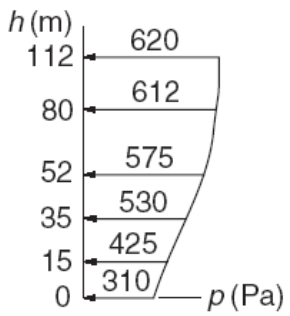
f)

$$I = \int_0^{12\pi} \mathrm{e}^{-x} \sin(50x) dx = \frac{50}{2501}(1 - e^{-2\pi})$$

g)

$$I = \int_{-9}^{10000} \frac{dx}{\sqrt{\lceil x \rceil}} = 206$$

PROBLEM 34



Wind pressure in a flat wall is measures as in the graphic. It is desired to find pressure effect center of the wall. It is calculated by using

$$\bar{h} = \frac{\int_0^{112} hp(h)dh}{\int_0^{112} p(h)dh}$$

Calculate the value.

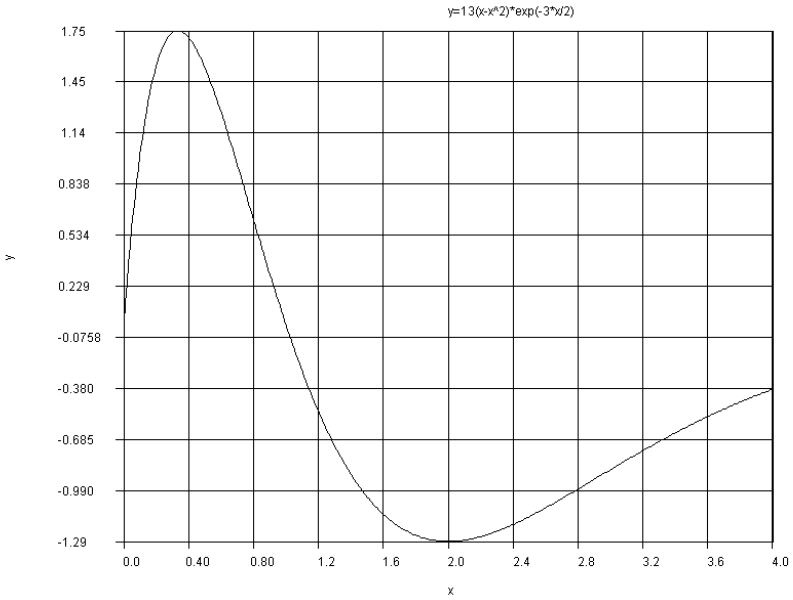
PROBLEM 35

For the analytically solved integration that result is given below:

Solve with with

a) Adaptive Simpson 1/3 method
b) Adaptive Gauss-Kronrod integral formula.

$$\int_0^4 13(x - x^2)e^{-3x/2} dx = \frac{4108e^{-6} - 52}{27} = -1.5487883725279481333$$



PROBLEM 36

v(m/s)	0	1	1.8	2.5	3.5	4.4	5.1	6
P(kW)	0	4.7	12.2	19.0	31.8	40.1	43.8	43.2

Power transmitted to the wheels of a vehicle is given in the table as a function of velocty. If the mass of the ar is m=2000 kg, Calculate the required time to accelarate the car from
0 m/s^2 to 6 m/s^2
Note :

$$\Delta t = m \int_{t=0\,s}^{6\,s} \frac{v}{P} dv$$

PROBLEM 37

$$I = \int_0^1 \int_1^2 \int_0^y e^{x+y+z} dzdydx$$

Calculate the integral by using 3 dimensional Gauss-Legendre integration

PROBLEM 38

$$I = \int_0^{\pi} \int_1^{x} \int_0^{yx} \frac{1}{y} sin\left(\frac{z}{y}\right) dzdydx$$

Calculate the integral by using 3 dimensional Gauss-Legendre integration

PROBLEM 39

Blackbody radiation function F(0- λT), λ(μm) T(degree Kelvin) can be calculated as

$$x = \frac{C_2}{\lambda T} = \frac{14387.69\ \mu mK}{\lambda T}$$

$$F(0 - \lambda T) = 1 - \frac{15}{\pi^4} \int_0^x \frac{x^3}{e^x - 1} dx$$

It can also be calculated from the series solution

$$F(0-\lambda T)=\frac{15}{\pi^4}\sum_{n=1}^{\infty}\left[\frac{e^{-nx}}{n}\left(x^3+\frac{3x^2}{n}+\frac{6x}{n^2}+\frac{6}{n^3}\right)\right]$$

By creating a program that calculates the result with using series solution and integration. Compare the results. Resulting tables are given as a reference

λ T μ mK	F(0-λT)	λ T μ mK	F(0-λT)	λ T μ mK	F(0-λT)	λ T μ mK	F(0-λT)	λ T μ mK	F(0-λT)	λ T μ mK	F(0-λT)
1000	0.00032	2900	0.25056	4800	0.60754	6900	0.8022	10700	0.92709	18000	0.98081
1050	0.00056	2950	0.26191	4850	0.61428	7000	0.80808	10800	0.92872	18200	0.98137
1100	0.00091	3000	0.27323	4900	0.62089	7100	0.81374	10900	0.93031	18400	0.98191
1150	0.00142	3050	0.28453	4950	0.62737	7200	0.81918	11000	0.93185	18600	0.98243
1200	0.00213	3100	0.29578	5000	0.63373	7300	0.82443	11200	0.9348	18800	0.98293
1250	0.00308	3150	0.30697	5050	0.63996	7400	0.82949	11400	0.93758	19000	0.98341
1300	0.00432	3200	0.3181	5100	0.64608	7500	0.83437	11600	0.94021	19200	0.98387
1350	0.00587	3250	0.32915	5150	0.65207	7600	0.83907	11800	0.9427	19400	0.98431
1400	0.00779	3300	0.34011	5200	0.65795	7700	0.8436	12000	0.94505	19600	0.98474
1450	0.01011	3350	0.35097	5250	0.66371	7800	0.84797	12200	0.94728	19800	0.98516
1500	0.01285	3400	0.36173	5300	0.66937	7900	0.85219	12400	0.94939	20000	0.98555
1550	0.01605	3450	0.37238	5350	0.67491	8000	0.85625	12600	0.95139	20500	0.98649
1600	0.01972	3500	0.38291	5400	0.68034	8100	0.86018	12800	0.95329	21000	0.98735
1650	0.02388	3550	0.39332	5450	0.68566	8200	0.86397	13000	0.95509	21500	0.98814
1700	0.02854	3600	0.4036	5500	0.69089	8300	0.86763	13200	0.95681	22000	0.98886
1750	0.03369	3650	0.41375	5550	0.696	8400	0.87116	13400	0.95843	22500	0.98952
1800	0.03934	3700	0.42377	5600	0.70102	8500	0.87457	13600	0.95998	23000	0.99014
1850	0.04549	3750	0.43364	5650	0.70594	8600	0.87787	13800	0.96145	23500	0.9907
1900	0.05211	3800	0.44338	5700	0.71077	8700	0.88105	14000	0.96285	24000	0.99123
1950	0.0592	3850	0.45297	5750	0.7155	8800	0.88413	14200	0.96419	24500	0.99172
2000	0.06673	3900	0.46241	5800	0.72013	8900	0.88711	14400	0.96546	25000	0.99217
2050	0.07469	3950	0.47172	5850	0.72468	9000	0.88999	14600	0.96667	26000	0.99297
2100	0.08306	4000	0.48087	5900	0.72914	9100	0.89278	14800	0.96783	27000	0.99368
2150	0.0918	4050	0.48987	5950	0.73351	9200	0.89547	15000	0.96893	28000	0.99429
2200	0.10089	4100	0.49873	6000	0.73779	9300	0.89808	15200	0.96999	29000	0.99482
2250	0.11031	4150	0.50744	6050	0.74199	9400	0.9006	15400	0.971	30000	0.99529
2300	0.12003	4200	0.516	6100	0.74611	9500	0.90305	15600	0.97196	32000	0.99607
2350	0.13002	4250	0.52442	6150	0.75015	9600	0.90541	15800	0.97289	34000	0.99669
2400	0.14026	4300	0.53269	6200	0.75411	9700	0.9077	16000	0.97377	36000	0.99719
2450	0.15071	4350	0.54081	6250	0.758	9800	0.90992	16200	0.97461	38000	0.99759
2500	0.16136	4400	0.54878	6300	0.76181	9900	0.91207	16400	0.97542	40000	0.99792
2550	0.17217	4450	0.55662	6350	0.76554	10000	0.91416	16600	0.9762	45000	0.99851
2600	0.18312	4500	0.56431	6400	0.76921	10100	0.91618	16800	0.97694	50000	0.9989
2650	0.19419	4550	0.57186	6450	0.7728	10200	0.91814	17000	0.97765	55000	0.99917
2700	0.20536	4600	0.57927	6500	0.77632	10300	0.92004	17200	0.97834	60000	0.99935
2750	0.2166	4650	0.58654	6600	0.78317	10400	0.92188	17400	0.97899		
2800	0.22789	4700	0.59367	6700	0.78976	10500	0.92367	17600	0.97962		
2850	0.23922	4750	0.60067	6800	0.7961	10600	0.9254	17800	0.98023		

PROBLEM 40

Integral of blackbody spectral radiation equation for all wavelengths will give

$$E_b=\left[\frac{C_1}{C_2^4}\int_{x=0}^{\infty}\frac{x^3}{e^x-1}dx\right]T^4=\sigma T^4$$

In this equation C_1=3.74177489x10^8 Wmm^4/m^2 and C_2=1.438769x10^4 mm K . □ is called Stefan_Boltzman constant. Calculate this integral by using Gauss-Leguerre integral formula. The value of the Stephan Boltzman constant is given σ =5.670x10^{-8} W/m^{2}K^4

PROBLEM 41 Bessel function $J_n(x)$ can be defined as:

$$J_n(x)=\frac{1}{\pi}\int_0^{\pi}cos(xsin(\theta)-n\theta)d\theta$$

Write a program to define such a function.

```
from scipy.special import *
from math import *
from if_x import *
```

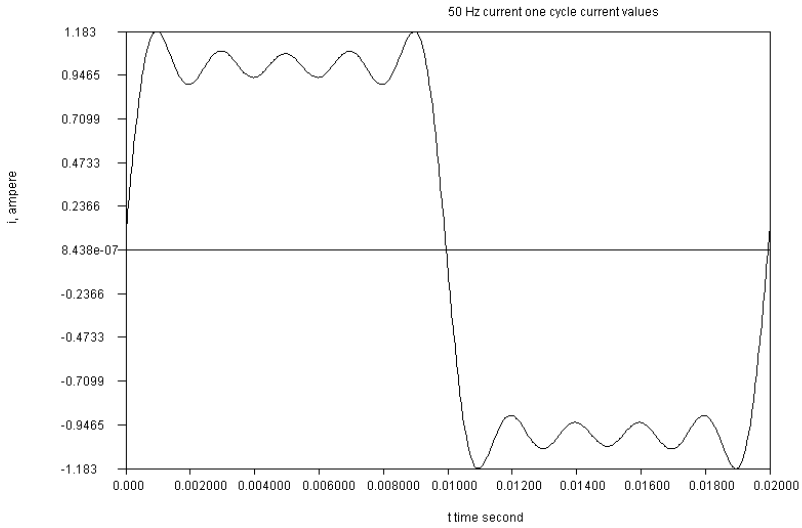
```
def gauss_legendre_coefficients(x1,x2,n):
   #calculates legendre gauss-coefficients as coefficients of the integral
   #for n terms
      eps=3e-15
      m=int((n+1.0)/2.0)
      xm=0.5*(x2+x1)
      xl=0.5*(x2-x1)
      nmax=100
      a=[[0.0 for i in range(n)] for j in range(2)]
      for i in range(1,m+1):
         z=cos(pi*((i-0.25)/(n+0.5)))
         for ii in range(nmax):
            #while abs(z-z1) > eps:
            p1=1.0;
            p2=0.0;
            for j in range(1,n+1):
               p3=p2
               p2=p1
               p1=(float)(((2.0*j-1.0)*z*p2-(j-1.0)*p3)/j)
            pp=(float)(n*(z*p1-p2)/(z*z-1.0))
            z1=z;
            z=z1-p1/pp
            if abs(z-z1) < eps: break
         a[0][i-1]=xm-xl*z
         a[0][n-i]=xm+xl*z
         a[1][i-1]=2.0*xl/((1.0-z*z)*pp*pp)
         a[1][n-i]=a[1][i-1]
      return a

def integral(f,x1,x2,n):
      #integral func(x)dx
      #integral of a function by using gauss-legendre quadrature
      #between x1 and x2
      a=gauss_legendre_coefficients(x1,x2,n)
      z=0
      for i in range(n):
         z=z+a[1][i]*f.func(a[0][i])
      return z
n=0
x=0.1
class f1(if_x):func=lambda self,tetha: cos(x*sin(tetha))-n*tetha
f=f1()
a=0
b=pi
bessel1=integral(f,a,b,50)/pi
print("bessel1  : ",bessel1)
bessel2=j0(x)
print("bessel1  : ",bessel2)
```

```
runfile('D:/okul/SCO1/integral_GL_P41.py', wdir='D:/okul/SCO1')
Reloaded modules: if_x
bessel1  :  0.997501562066038
bessel1  :  0.99750156206604
```

PROBLEM 42 In an AC-DC converter the signal defined with the following function is obtained. Signal frequency is 50 hertz (period T=1.0/50=0.02 second). A true root mean square current value for an alternative current signal is defined as:

$$I_{RMS} = \sqrt{\frac{1}{T}\int_0^T i^2(t)dt}$$

Calculate True RMS current for the given signal.



Note: In order to carry outintegral you should obtain $i(t)$ as a function y using data drom the plot above (digitizing and curve fitting)

PROBLEM 43 Work is defined as
Work =Force*Distance
In a real engineering problem force usually changes, Therefore Work for a variable force should be calculated as:

$$W = \int_{x_0}^{x_n} F(x)dx$$

In a system Force is measured as in the following table:

x(meter)	**0**	**5**	**10**	**15**	**20**	**25**	**30**	**35**	**40**
F(Newton)	**0**	**9**	**12**	**15**	**18**	**23**	**16**	**13**	**11**

Calculate the work done by the system.

PROBLEM 44 Euler's constant is defined by the following equations:

$$\gamma = \int_{x=0}^{\infty}\left(\frac{1}{1+x} - e^{-x}\right)\frac{dx}{x} = \sum_{n=1}^{\infty}\left[\frac{1}{n} - ln\left(\frac{n+1}{n}\right)\right]$$

where k > 0

$$\gamma = 0.577215664901532860606511 2 \ldots$$

Calculate **euler's** constant by integration and compare it with the series solution. You chose different integration methods to see the accuracy of the integration formula.

PROBLEM 45 Catalan's constant is defined by the following equations:

$$K = \int_{x=0}^{1}\left(\frac{\ln(x)}{1+x^2}\right)dx$$

Catalan's constant (K=0.915965594177) is defined by the following equations:

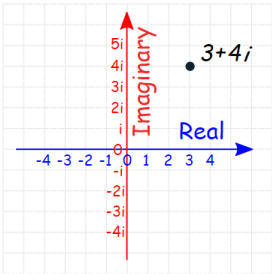
$$K = \frac{3}{64}\sum_{k=0}^{\infty}\frac{(-1)^k}{64^k}\left(\frac{32}{(12k+1)^2} - \frac{32}{(12k+2)^2} - \frac{32}{(12k+3)^2} - \frac{8}{(12k+5)^2} - \frac{16}{(12k+6)^2} - \frac{4}{(12k+7)^2} - \frac{4}{(12k+9)^2} - \frac{2}{(12k+10)^2} + \frac{1}{(12k+11)^2}\right)$$

Calculate **Catalan's** constant by integration and compare it with the series solution. You chose different integration methods to see the accuracy of the integration formula.

5. COMPLEX NUMBERS

$z = a + bi$ where $i = \sqrt{-1}$

A complex number can now be shown as a point:



The complex number 3 + 4**i**

Arithmetic of the complex numbers

$z_1 = a_1 + b_1 i \qquad z_2 = a_2 + b_2 i$

$z_1 + z_2 = (a_1 + a_2) + (b_1 + b_2)i$

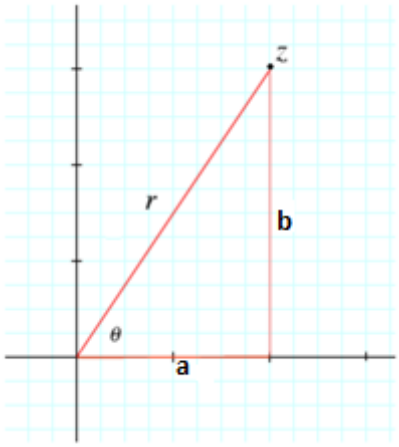
$z_1 - z_2 = (a_1 - a_2) + (b_1 - b_2)i$

Conjugate$(z) = a - bi$

$z_1 * z_2 = (a_1 * a_2 - b_1 * b_2) + (a_1 * b_2 + b_1 * a_2)i$

$$\frac{z_1}{z_2} = \frac{a_1 + b_1 i}{a_2 + b_2 i} = \frac{(a_1 + b_1 i)(a_2 - b_2 i)}{(a_2 + b_2 i)(a_2 - b_2 i)} = \frac{(a_1 * a_2 + b_1 * b_2) + (b_1 * a_2 - a_1 * b_2)i}{a_2^2 + b_2^2}$$

Complex numbers can also be expressed in polar coordinates



$z = a + bi = r(\cos(\theta) + i\sin(\theta))$

$r = \sqrt{a^2 + b^2} \quad \theta = \text{atan}\left(\frac{b}{a}\right)$

$a = r\cos(\theta) \quad b = \text{r}\sin(\theta)$

$e^{i\theta} = \cos(\theta) + i\sin(\theta)$

$e^{-i\theta} = \cos(\theta) - i\sin(\theta)$

$z = a + bi = r(\cos(\theta) + i\sin(\theta)) = re^{i\theta}$

This equation is called De Moivre formulation

Utilisation of polar coordinates makes power calculation of complex variables easier.

$z^n = (a + bi)^n = r^n(\cos(n\theta) + i\sin(n\theta)) = r^n e^{in\theta}$

$\sqrt{z} = (a + bi)^{0.5} = \sqrt{r}(\cos\left(\frac{\theta}{2}\right) + i\sin\left(\frac{\theta}{2}\right))$

Let us look at the roots of degree n

$z^{\frac{1}{n}} = (a+bi)^{\frac{1}{n}} = r^{\frac{1}{n}}(\cos\left(\frac{\theta}{n}\right) + i\sin\left(\frac{\theta}{n}\right)) = r^n e^{i\left(\frac{\theta}{n}\right)}$

But in this formula $\theta = \theta + 2k\pi$ If we substitude this into the equation

$z^{\frac{1}{n}} = (a+bi)^{\frac{1}{n}} = r^{\frac{1}{n}}(\cos\left(\frac{\theta+2k\pi}{n}\right) + i\sin\left(\frac{\theta+2k\pi}{n}\right)) = r^n e^{i\left(\frac{\theta+2k\pi}{n}\right)}$ k=0,1,2..(n-1)

So equation has n roots

Absulute value of complex numbers:

$|z| = r = \sqrt{a^2+b^2}$

Multiplication of complex numbers:

$z_1 * z_2 = \left(r_1 e^{i\theta_1}\right)\left(r_2 e^{i\theta_2}\right) = r_1 r_2 e^{i(\theta_1+\theta_2)}$

Division of complex numbers

$\frac{z_1}{z_2} = \frac{\left(r_1 e^{i\theta_1}\right)}{\left(r_2 e^{i\theta_2}\right)} = \frac{r_1}{r_2} e^{i(\theta_1-\theta_2)}$

Example:

$z_1 = 2+3i$

$z_2 = 4-5i$

$$z = \frac{z_1}{z_2} = \frac{2+3i}{4-5i} = \frac{2+3i}{4-5i}\frac{4+5i}{4+5i} = \frac{8+10i+12i+15i^2}{16+20i-20i-25i^2} = \frac{8+10i+12i-15}{16+20i-20i+25} = \frac{-7+22i}{41}$$
$$= \frac{-7}{41} + \frac{22}{41}i = -0.1707317 + 0.5365853i$$

In python complex numbers can directly be assigned and used.

```
z1=complex(2,3)
z2=complex(4,5)
z3=z1/z2
print("z3=",z3)
```

```
runfile('E:/okul/SCO1/complex1.py', wdir='E:/okul/SCO1')
z3= (-0.17073170731707318+0.5365853658536587j)
```

Example:

$z_1 = 2+3i$

$z_2 = z_1^2 = (2+3i\ \ )(2+3i\ \ ) = 4+6i+6i-9 = -5+12i$

Calculating by using polar coordinates:

$r = \sqrt{2^2+3^2} = \sqrt{13} = 3.605551 \quad \theta = \text{atan}\left(\frac{3}{2}\right) = 0.982$

$r^2 = 13 \quad \cos(2\,\theta) = -0.384615 \quad \sin(2\theta) = 0.923076$

$z_2 = z_1^2 = 13(-0.384616 + 0.923076i) = -5+12i$

```
from cmath import *

z1=complex(2,3)
z2=z1*z1
z3=pow(z1,2.0)
print("z2=",z2,"z3=",z3)
```

```
runfile('E:/okul/SCO1/complex1.py', wdir='E:/okul/SCO1')
z2= (-5+12j) z3= (-5+12j)
```

Example:

$z_1 = 2+3i$

$z_2 = \sqrt{z_1} = z_1^{0.5}$

Calculating by using polar coordinates:

$r = \sqrt{2^2+3^2} = \sqrt{13} = 3.605551 \quad \theta = \text{atan}\left(\frac{3}{2}\right) = 0.982$

$\sqrt{r} = 13 = 1.898828 \quad \cos(0.5\,\theta) = 0.881674598 \quad \sin(0.5\theta) = 0.471857925$

$z_2 = z_1^2 = 1.898828 \quad (0.881674598 + 0.471857925i) = 1.674149 + 0.895977i$

```
from cmath import *

z1=complex(2,3)
z2=sqrt(z1)
z3=pow(z1,0.5)
print("z2=",z2,"z3=",z3)
```

```
runfile('E:/okul/SCO1/complex1.py', wdir='E:/okul/SCO1')
z2= (1.6741492280355401+0.8959774761298381j) z3= (1.6741492280355401+0.895977476129838j)
```

Example:
$z_1 = 2 + 3i$
$z_2 = 4 - 5i$
$z_3 = {z_1}^{z_2}$

$\ln(z_3) = z_2 \ln(z_1)$

$z_3 = exp(z_2 \ln(z_1))$

```
from cmath import *

z1=complex(2,3)
z2=complex(4,-5)
z3=pow(z1,z2)
z4=exp(z2*log(z1))
print("z3=",z3,"z4=",z4)
```

```
z3= (-18175.487698625275-14117.567839250678j) z4= (-18175.487698625275-14117.56783925068j)
```

Example: Prove that following equation is correct

$$\frac{3i^{30} - i^{19}}{2i - 1} = 1 + i$$

$$\frac{3i^{30} - i^{19}}{2i - 1} = \frac{3 - i}{2i + 1} = \frac{(3 - i)}{(2i + 1)}\frac{(2i - 1)}{(2i - 1)} = \frac{(3 + 2) - i + 6i}{2^2 + 1^2} = \frac{5 + 5i}{5} = 1 + i$$

Example:

$$\sin(\theta) = \frac{1}{2i}\left(e^{i\theta} - e^{-i\theta}\right)$$
$$\cos(\theta) = \frac{1}{2}\left(e^{i\theta} + e^{-i\theta}\right)$$

$$\frac{d\sin(\theta)}{d\theta} = \frac{i}{2i}\left(e^{i\theta} + e^{-i\theta}\right) = \frac{1}{2}\left(e^{i\theta} + e^{-i\theta}\right) = \cos(\theta)$$

$$\frac{d\cos(\theta)}{d\theta} = \frac{i}{2}\left(e^{i\theta} - e^{-i\theta}\right) = \frac{i^2}{2i}\left(e^{i\theta} - e^{-i\theta}\right) = \frac{-1}{2i}\left(e^{i\theta} - e^{-i\theta}\right) = -\sin(\theta)$$

Numerical finite difference formula:

$\frac{df(x)}{dx} \cong \frac{f(x+dx)-f(x)}{dx}$ for $f(x) = \sin(x)$ and x=0, dx=0.0001

$$\frac{dsin(x)}{dx} \cong \frac{sin(x + dx) - \sin(x)}{dx} = \frac{sin(dx)}{dx} = \frac{9.9999999833333\text{e} - 5}{0.0001}$$

Sin(0.0001)=9.9999999833333e-5

$$\left.\frac{dsin(x)}{dx}\right|_{x=0} = 0.99999999833333 \cong \cos(0)$$

$$\int \sin(\theta)\, d\theta = \int \frac{1}{2i}(e^{i\theta} - e^{-i\theta})d\theta = \frac{1}{2i^2}(e^{i\theta} + e^{-i\theta}) = -\frac{1}{2}(e^{i\theta} + e^{-i\theta}) = -\cos(\theta)$$

So

$$\int_0^{\pi/2} \sin(\theta)\, d\theta = -\cos(\theta)|_0^{\pi/2} = -\cos\left(\frac{\pi}{2}\right) + \cos(0) = 1$$

If we use numerical integral (Gauss-Legendre) to calculate this:

```
 import cmath as cm
from math import *


def f(x):
    #sin(x) function in complex form
    x1=complex(0,x)
    z=2.0*complex(0,1)
    y=(cm.exp(x1)-cm.exp(-x1))/z
    return y

def gauss_legendre_coefficients(x1,x2,n):
   #calculates legendre gauss-coefficients as coefficients of the integral
   #for n terms
      eps=3e-15
      m=int((n+1.0)/2.0)
      xm=0.5*(x2+x1)
      xl=0.5*(x2-x1)
      nmax=100
      a=[[0.0 for i in range(n)] for j in range(2)]
      for i in range(1,m+1):
         z=cos(pi*((i-0.25)/(n+0.5)))
         for ii in range(nmax):
            #while abs(z-z1) > eps:
            p1=1.0;
            p2=0.0;
            for j in range(1,n+1):
               p3=p2
               p2=p1
               p1=(float)(((2.0*j-1.0)*z*p2-(j-1.0)*p3)/j)
            pp=(float)(n*(z*p1-p2)/(z*z-1.0))
            z1=z;
            z=z1-p1/pp
            if abs(z-z1) < eps: break
         a[0][i-1]=xm-xl*z
         a[0][n-i]=xm+xl*z
         a[1][i-1]=2.0*xl/((1.0-z*z)*pp*pp)
         a[1][n-i]=a[1][i-1]
      return a


def integral(f,x1,x2,n):
   #integral func(x)dx
   #integral of a function by using gauss-legendre quadrature
   #between x1 and x2
   a=gauss_legendre_coefficients(x1,x2,n)
   z=0
   for i in range(n):
      z=z+a[1][i]*f(a[0][i])
   return z


a=0
b=pi/2.0
I1=integral(f,a,b,20)
print("I1  : ",I1);
```

```
runfile('E:/okul/SCO1/integral_GL.py', wdir='E:/okul/SCO1')
I1  :  (0.9999999999999982+0j)
```

Same result is obtained

We will look at further utilisation of complex numbers in coming chapters.

6. LINEAR SYSTEM OF EQUATIONS

Lineear system of equations (or linear system) is a collection of one or more linear equations involving the same set of variables. For example

2x+3.5y=1.2

1.1x+1.5y=2

In this particular set, x and y is variables. System has two equations. It is a linear equations, becouse variables x and y linearly placed in the equation. The following set of equations is non-linear

$$2\sin(x) + 3.5e^{y} = 1.2$$
$$1.1\cos(x) + 1.5y = 2$$

In mathematics, linear system of equation is expressed as a matrix-vector format. For example above linear set can be written as

$$\begin{bmatrix} 2 & 3.5 \\ 1.1 & 1.5 \end{bmatrix} \begin{Bmatrix} x \\ y \end{Bmatrix} = \begin{Bmatrix} 1.2 \\ 2 \end{Bmatrix}$$

In some boooks the following formats of expressing the same system of linear equations can also be seen:

$$\left[\begin{array}{cc|c} 2 & 3.5 & 1.2 \\ 1.1 & 1.5 & 2 \end{array}\right]$$

Or simply

$$\begin{bmatrix} 2 & 3.5 & 1.2 \\ 1.1 & 1.5 & 2 \end{bmatrix}$$

In high school math, if it is only two equations and two unknown varibles variables, the following method of placement is shown to us to solve this system:

2x+3.5y=1.2

1.1x+1.5y=2

$x = (1.2 - 3.5y)/2$

$1.1\frac{(1.2-3.5y)}{2} + 1.5\text{y} = 2$

$0.66 - 1.925y + 1.5y = 2$

$-0.425y = 2 - 0.66$

$y =$-3.15294

Substituting to x gives

$x =$6.117647

6.1 GAUSS ELIMINATION METHOD

2x+3.5y=1.2

1.1x+1.5y=2

Can be written as

$$\begin{bmatrix} 2 & 3.5 \\ 1.1 & 1.5 \end{bmatrix} \begin{Bmatrix} x \\ y \end{Bmatrix} = \begin{Bmatrix} 1.2 \\ 2 \end{Bmatrix}$$

In general

$$\begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix} \begin{Bmatrix} x_0 \\ x_1 \end{Bmatrix} = \begin{Bmatrix} b_0 \\ b_1 \end{Bmatrix}$$

2x+3.5y+3z=1.2

1.1x+1.5y-1.5z=2

x+5y-2z=3

$$\begin{bmatrix} 2 & 3.5 & 3 \\ 1.1 & 1.5 & -1.5 \\ 1 & 5 & -2 \end{bmatrix} \begin{Bmatrix} x \\ y \\ z \end{Bmatrix} = \begin{Bmatrix} 1.2 \\ 2 \\ 3 \end{Bmatrix}$$

In general

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix} \begin{Bmatrix} x_0 \\ x_1 \\ x_2 \end{Bmatrix} = \begin{Bmatrix} b_0 \\ b_1 \\ b_2 \end{Bmatrix}$$

Upper triangle system

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} \\ 0 & a_{11} & a_{12} \\ 0 & 0 & a_{22} \end{bmatrix} \begin{Bmatrix} x_0 \\ x_1 \\ x_2 \end{Bmatrix} = \begin{Bmatrix} b_0 \\ b_1 \\ b_2 \end{Bmatrix}$$

Gauss elimination method: Gauss elimnation method is based on two charecteristics of matrices (or system of linear equations as well):

1. If one equation (on eline of system is multiplied with a number the solution set remain the same
2. If one row is substracted or added to another row, solution set will still be the same.

By using this two characteristics, system of equation will be reduced to upper triabgle system. And then by starting from the last equation, sytem will be solved by backsubstitution process.

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix} \begin{Bmatrix} x_0 \\ x_1 \\ x_2 \end{Bmatrix} = \begin{Bmatrix} b_0 \\ b_1 \\ b_2 \end{Bmatrix}$$

c1=a_{10}/a_{00} c2=a_{20}/a_{00}

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} - \mathrm{c1} * a_{00} & a_{11} - c1 * a_{01} & a_{12} - c1 * a_{02} \\ a_{20} - \mathrm{c2} * a_{00} & a_{21} - \mathrm{c2} * a_{01} & a_{22} - \mathrm{c2} ** a_{02} \end{bmatrix} \begin{Bmatrix} x_0 \\ x_1 \\ x_2 \end{Bmatrix} = \begin{Bmatrix} b_0 \\ b_1 - c1 * b_0 \\ b_2 - \mathrm{c2} * b_0 \end{Bmatrix}$$

Becomes

$$\begin{bmatrix} a'_{00} & a'_{01} & a'_{02} \\ 0 & a'_{11} & a'_{12} \\ 0 & a'_{21} & a'_{22} \end{bmatrix} \begin{Bmatrix} x_0 \\ x_1 \\ x_2 \end{Bmatrix} = \begin{Bmatrix} b'_0 \\ b'_1 \\ b'_2 \end{Bmatrix}$$

c3=a'_{21}/a'_{11}

$$\begin{bmatrix} a'_{00} & a'_{01} & a'_{02} \\ 0 & a'_{11} & a'_{12} \\ 0 & a'_{21} - c3 * a'_{11} & a'_{22} - c3 * a'_{11} \end{bmatrix} \begin{Bmatrix} x_0 \\ x_1 \\ x_2 \end{Bmatrix} = \begin{Bmatrix} b'_0 \\ b'_1 \\ b'_2 - c3 * b'_1 \end{Bmatrix}$$

Becomes

$$\begin{bmatrix} a''_{00} & a''_{01} & a''_{02} \\ 0 & a''_{11} & a''_{12} \\ 0 & 0 & a''_{22} \end{bmatrix} \begin{Bmatrix} x_0 \\ x_1 \\ x_2 \end{Bmatrix} = \begin{Bmatrix} b''_0 \\ b''_1 \\ b''_2 \end{Bmatrix}$$

Back substitution:

$$x_2 = b''_2/a''_{22}$$
$$x_1 = (b''_1 - a''_{12} * x_2)/a''_{11}$$
$$x_0 = (b''_0 - a''_{01} * x_1 - a''_{02} * x_2)/\, a''_{00}$$

In the program following set of system of equation is taken :

$$\begin{bmatrix} 1 & 2 & 5 \\ 2 & 1 & 2 \\ 3 & 1 & 1 \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix}$$

The solution set of this set of equation is 0,3 ve -1. As it is seen from the results, a small error is remained in the solution set. If the same elinination process is carried out by hand:

column 0(column indices starts from 0), row 1(raw indices starts from 0) multiplier $C_1=a_{10}/a_{00}$ = 2/1=2

a_{10} = a_{10} - C_1 * a_{00} = a_{10} - (a_{10}/a_{00})* a_{00} = 2 - 2*1 = 0

a_{11} = a_{11} - C_1 * a_{01} = a_{11} - (a_{10}/a_{00})* a_{01} = 1 - 2*2 = -3

a_{12} = a_{12} - C_1 * a_{02} = a_{12} - (a_{10}/a_{00})* a_{02} = 2 - 2*5 = -8

$b_2 = b_2 - C_1 * b_1 = b_2 - (a_{10}/a_{00})* b_1 = 1 – 2*1= -1$
column0 raw 1 elimination is completed
colun0 raw 2 elimination:
$C_1=a_{20}/a_{00} = 3/1=3$
$A_{20} = a_{20} – C_2 * a_{00} = a_{20} - (a_{20}/a_{00})* a_{00} = 3 - 3*1 = 0$
$a_{21} = a_{21} – C_2 * a_{01} = a_{21} - (a_{20}/a_{00})* a_{01} = 1 - 3*2 = -5$
$a_{22} = a_{22} – C_2 * a_{02} = a_{22} - (a_{20}/a_{00})* a_{02} = 2 - 3*5 = -14$
$b_2 = b_2 - C_2 * b_1 = b_2 - (a_{10}/a_{00})* b_1 = 1 – 3*1= -1$
column 1, row 2 multiplier C
$C_3=a_{21}/a_{11} = -5/-3=(5/3)$
$a_{21} = a_{21} – C_3 * a_{11} = a_{21} - (a_{21}/a_{11})* a_{11} = -5 – (5/3)*(-3) = 0$
$a_{22} = a_{22} - C_3 * a_{12} = a_{11} - (a_{21}/a_{11})* a_{01} = -14 – (5/3)*(-8) = -0.6666666666666$
$b_2 = b_2 - C_3 * b_1 = b_2 - (a_{21}/a_{11})* b_1 = -1 – (5/3)*(-1) = 0.6666666666666$
system of equation becomes:

$$\begin{bmatrix} 1 & 2 & 5 \\ 0 & -3 & -8 \\ 0 & 0 & -0.66666 \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \\ 0.66666 \end{bmatrix}$$

Back-substitution process :
$X_2 = b_2/a_{22} = 0.6666666666/(-0.6666666666) = -1$
$X_1 = (b_1 - a_{12}*X_2)/a_{11} = (-1 - (-8)*(-1))/(-3) =(-9)/(-3)=3$
$X_0 = (b_0 - a_{01}*X_1 - a_{02}*X_2)/a_{00} = (1 – 2*3-5*(-1))/1=0$

$$\begin{bmatrix} X_0 \\ X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 3 \\ -1 \end{bmatrix}$$

Another example for Gauss elimination with detailed steps :

$$\begin{bmatrix} 7 & 1 & 2 \\ -1 & 4 & -1 \\ 3 & 15 & 20 \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} 47 \\ 19 \\ 87 \end{bmatrix}$$

A B

	7	1	2	47
	-1	4	-1	19
	3	15	20	87

c				
	7	1	2	47
-0.14286	0	4.142857	-0.71429	25.71429
0.428571	0	14.57143	19.14286	66.85714

c				
	7	1	2	47
3.517241	0	4.142857	-0.71429	25.71429
	0	0	21.65517	-23.5862

X_0	**6.165605**
X_1	**6.019108**
X_2	**-1.08917**

As it is seen from the previous example some error existed in the solution set. It is called round off error. When gauss elimination process is used as it is defined above some error might be created by the process. In order to minimize the round off error, absolute value of the multiplier terms should be as big as possible. In the calculation process of the multiplier if divisor is zero term will go to infinity, if divisior is relatively small it will be a very big number, and this will cause an extra error in digital computers. In order to prevent this in a degree, elimination raw is interchanged with the raw with maximum absolute value for the multiplier term (in the diagonal). This process is called partial pivoting. Partial pivoting is shown in the example

$$\begin{bmatrix} 1 & 2 & 5 \\ 2 & 1 & 2 \\ 3 & 1 & 1 \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix}$$

Before starting the gauss elimination the column 0 elements are compared and raw with the largest absolute value of this column is replace with elimination raw (raw 0)

$$\begin{bmatrix} 3 & 1 & 1 \\ 2 & 1 & 2 \\ 1 & 2 & 5 \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix}$$

```
# Gauss Elimination with pivoting
# python version
from math import *

def gauss(a,b):
    #Gauss elimination with pivoting
  n=len(a)
  x=[0 for i in range(n)]
  for k in range(n):
    p=k
    buyuk=abs(a[k][k])
    for ii in range(k+1,n):
      dummy=abs(a[ii][k])
      if dummy>buyuk:
         buyuk=dummy
         p=ii
    if p!=k:
      for jj in range(k,n):
        dummy=a[p][jj]
        a[p][jj]=a[k][jj]
        a[k][jj]=dummy
      dummy=b[p]
      b[p]=b[k]
      b[k]=dummy
    for i in range(k+1,n):
       carpan=a[i][k]/a[k][k]
       a[i][k]=0;
       for j in range(k+1,n):
          a[i][j]=a[i][j]-carpan*a[k][j]
       b[i]   =b[i]-carpan*b[k]
  x[n-1]=b[n-1]/a[n-1][n-1]
  for i in range(n-1,-1,-1):
    toplam=0;
    for j in range(i+1,n):
      toplam=toplam+a[i][j]*x[j]
    x[i]=(b[i]-toplam)/a[i][i];
  return x;

def I(n):
    #unit matrix of dimension n
    B = [[0 for x in range(n)] for y in range(n)];
    for i in range(0,n):
        for j in range(0,n):
           if i==j: B[i][j]=1.0;
    return B;

def invA(A):
    #inverse matrix (n=m)
    n=len(A);
    Im= I(n);
    B = [[0 for x in range(n)] for y in range(n)];
    for i in range(n):
        B[i]=gauss(A,Im[i]);
    return B;
```

```
#gauss_test
from math import *
from gauss import *

a=[[1,2,5],[2,1,2],[3,1,1]]
b=[1,1,2]
x=gauss(a,b);
print("Gauss elimination solution vector :\nx = ",x)
```

```
runfile('E:/okul/SCO1/gauss_test.py', wdir='E:/okul/SCO1')
Reloaded modules: gauss
Gauss elimination solution vector :
x =  [7.401486830834377e-17, 2.9999999999999996, -0.9999999999999998]
```

```
#gauss_test
from math import *
from gauss import *

a=[[2,3.5],[1.1,1.5]]
```

```
b=[1.2,2]
x=gauss(a,b);
print("Gauss elimination solution vector :\nx = ",x)
```

```
runfile('E:/okul/SCO1/gauss_test.py', wdir='E:/okul/SCO1')
Reloaded modules: fi_xi, gauss
Gauss elimination solution vector :
x =  [6.117647058823525, -3.152941176470586]
```

6.2 GAUSS-JORDAN METHOD

Gauss Jordan elimination method is similar to Gauss elimination with exception of a normalisation process before elimination and application of the elimination for all raws (instead of only lower lows of diagonal reference row in gauss elimination) except the diagonal reference raw. The result is conversion of the right hand side into the solution without any back substitution process. Example program is given below

$$\begin{bmatrix} 1 & 0.3333 & -0.3333 \\ 1 & 4 & 1 \\ 2 & 1 & 2 \end{bmatrix}\begin{bmatrix} X_0 \\ X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} 0.6667 \\ 12 \\ 10 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0.3333 & -0.3333 \\ 0 & 3.6667 & 1.3333 \\ 2 & 1 & 2 \end{bmatrix}\begin{bmatrix} X_0 \\ X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} 0.6667 \\ 11.333 \\ 10 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0.3333 & -0.3333 \\ 0 & 3.6667 & 1.3333 \\ 0 & 0.3333 & 2.6667 \end{bmatrix}\begin{bmatrix} X_0 \\ X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} 0.6667 \\ 11.333 \\ 8.6667 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0.3333 & -0.3333 \\ 0 & 1 & 0.3636 \\ 0 & 0.3333 & 2.6667 \end{bmatrix}\begin{bmatrix} X_0 \\ X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} 0.6667 \\ 3.0909 \\ 8.6667 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & -0.4545 \\ 0 & 1 & 0.3636 \\ 0 & 0.3333 & 2.6667 \end{bmatrix}\begin{bmatrix} X_0 \\ X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} -0.3636 \\ 3.0909 \\ 8.6667 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & -0.4545 \\ 0 & 1 & 0.3636 \\ 0 & 0 & 2.5455 \end{bmatrix}\begin{bmatrix} X_0 \\ X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} -0.3636 \\ 3.0909 \\ 7.6365 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & -0.4545 \\ 0 & 1 & 0.3636 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} X_0 \\ X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} -0.3636 \\ 3.0909 \\ 3.0000 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0.3636 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} X_0 \\ X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} 1.0000 \\ 3.0909 \\ 3.0000 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} X_0 \\ X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} 1.0000 \\ 2.0001 \\ 3.0000 \end{bmatrix}$$

Solution vector

$$\begin{bmatrix} X_0 \\ X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} 1.0000 \\ 2.0001 \\ 3.0000 \end{bmatrix}$$

Exact solution :

$$\begin{bmatrix} X_0 \\ X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} 1.0000 \\ 2.0000 \\ 3.0000 \end{bmatrix}$$

Another example for Gauss-Jordan elimination with detailed steps :

$$\begin{bmatrix} 7 & 1 & 2 \\ -1 & 4 & -1 \\ 3 & 15 & 20 \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} 47 \\ 19 \\ 87 \end{bmatrix}$$

Gauss-Jordan

A				B
7	1	2	X_0	47
-1	4	-1	X_1	19
3	15	20	X_2	87
Normalisation				
1	0.142857	0.285714	X_0	6.714286
-1	4	-1	X_1	19
3	15	20	X_2	87
Elimination				
1	0.142857	0.285714	X_0	6.714286
0	4.142857	-0.71429	X_1	25.71429
0	14.57143	19.14286	X_2	66.85714
Normalisation				
1	0.142857	0.285714	X_0	6.714286
0	1	-0.17241	X_1	6.206897
0	14.57143	19.14286	X_2	66.85714
Elimination				
1	0	0.310345	X_0	5.827586
0	1	-0.17241	X_1	6.206897
0	0	21.65517	X_2	-23.5862
Normalisation				
1	0	0.310345	X_0	5.827586
0	1	-0.17241	X_1	6.206897
0	0	1	X_2	-1.08917
Elimination				
1	0	0	X_0	**6.165605**
0	1	0	X_1	**6.019108**
0	0	1	X_2	**-1.08917**

```
# Gauss Elimination with pivoting
# python version
from math import *

def gauss_jordan(a,b):
  n=len(a)
  x=[0 for i in range(n)]
  print("origial matrix :\na =",a,"\nb =",b)
  for k in range(n):
    p=k
    buyuk=abs(a[k][k])
    for ii in range(0,n):
      dummy=abs(a[ii][k])
      if dummy>buyuk:
         buyuk=dummy
         p=ii
    if p!=k:
      for jj in range(0,n):
        dummy=a[p][jj]
        a[p][jj]=a[k][jj]
        a[k][jj]=dummy
      dummy=b[p]
      b[p]=b[k]
      b[k]=dummy
      # normalisation process
      for jj in range(0,n):
        a[k][jj]=a[k][jj]/a[k][k]
    print("pivoted matrix :\na =",a,"\nb =",b)
    for i in range(k+1,n):
       carpan=a[i][k]
       a[i][k]=0;
       for j in range(0,n):
          a[i][j]=a[i][j]-carpan*a[k][j]
       b[i]   =b[i]-carpan*b[k]
  print("eliminated matrix :\na =",a,"\nb =",b)
  return b;

a=[[1,2,5],[2,1,2],[3,1,1]]
b=[1,1,2]
x=gauss_jordan(a,b);
print("Gauss-Jordan elimination solution vector :\nx = ",x)
```

```
runfile('E:/okul/SCO1/gauss_jordan.py', wdir='E:/okul/SCO1')
```

```
origial matrix :
a = [[1, 2, 5], [2, 1, 2], [3, 1, 1]]
b = [1, 1, 2]
pivoted matrix :
a = [[1.0, 1.0, 1.0], [2, 1, 2], [1, 2, 5]]
b = [2, 1, 1]
pivoted matrix :
a = [[1.0, 1.0, 1.0], [-2.0, -1.0, 0.0], [-1.0, 1.0, 4.0]]
b = [2, -3, -1]
pivoted matrix :
a = [[1.0, 1.0, 1.0], [-2.0, -1.0, 0.0], [1.0, 1.0, 4.0]]
b = [2, -3, 2.0]
eliminated matrix :
a = [[1.0, 1.0, 1.0], [-2.0, -1.0, 0.0], [1.0, 1.0, 4.0]]
b = [2, -3, 2.0]
Gauss-Jordan elimination solution vector :
x =  [2, -3, 2.0]
```

6.3 LU-DECOMPOSITION METHODS

6.3.1 gauss(Dolittle) method

In Gauss or Gauss-Jordan elimination processes left hand matrix A and right hand vector b is eliminated together. Therefore if a new left hand side vector b1 will be considered to be used in as second set with the same A matrix, Matrix A is reprocessed. This will spend valuable calculation time for a repeated process. In order to avoid this repetation, Matrix A is decomposed into an upper triangle(U) and a lower triangle (L) matrices so that multiplication of this two matricess will give the original A matrix

For example for N=5

$$[L] = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ l_{21} & 1 & 0 & 0 & 0 \\ l_{31} & l_{32} & 1 & 0 & 0 \\ l_{41} & l_{42} & l_{43} & 1 & 0 \\ l_{51} & l_{52} & l_{53} & l_{54} & 1 \end{bmatrix}$$

$$[U] = \begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} & u_{15} \\ 0 & u_{22} & u_{23} & u_{24} & u_{25} \\ 0 & 0 & u_{33} & u_{34} & u_{35} \\ 0 & 0 & 0 & u_{44} & u_{45} \\ 0 & 0 & 0 & 0 & u_{55} \end{bmatrix}$$

To solve system of equation with a B vector following process applies

[A]=[L][U]

[A]{X}={B}

[L][U]{X}={B}

[L]{D}={B}

[U]{X}={D}

In Gauss(Doolittle) decomposition process matrix U is the same as Gauss elimination upper triangualar matrix. L matrix is consist of the multipliers calculated in Gauss elimination process. Due to diagonal element of matrix L is 1, LU matrix can be represented in one matrix if space saving is desired. Example code is given below

```
 # Gauss Elimination with pivoting
# python version
from math import *

def gaussLU(ai):
   #Gauss elimination with pivoting
  n=len(ai)
  x=[0 for i in range(n)]
  a=[[0 for i in range(n)] for j in range(n)]
  carpan=0
  toplam=0
  for i in range(n):
     for j in range(n):
        a[i][j]=ai[i][j]
  for k in range(n-1):
     for i in range(k+1,n):
        carpan=a[i][k]/a[k][k];
        a[i][k]=carpan
```

```
        for j in range(k+1,n):
            a[i][j]-=carpan*a[k][j]
  return a


def I(n):
    #unit matrix of dimension n
    B = [[0 for x in range(n)] for y in range(n)];
    for i in range(0,n):
        for j in range(0,n):
            if i==j: B[i][j]=1.0;
    return B;

def back_substitution(ai,bi):
    #gauss LU linear system of  equation solution
    n=len(ai)
    x=[0 for i in range(n)]
    a=[[0 for i in range(n)] for j in range(n)]
    b=[0 for i in range(n)]
    for i in range(n):
        for j in range(n):
            a[i][j]=ai[i][j]
        b[i]=bi[i]
    for i in range(n):
        toplam=b[i]
        for j in range(i):
            toplam-=a[i][j]*b[j]
        b[i]=toplam
    x[n-1]=b[n-1]/a[n-1][n-1];
    for i in range((n-2),-1,-1):
        toplam=0
        for j in range(i+1,n):
            toplam+=a[i][j]*x[j]
        x[i]=(b[i]-toplam)/a[i][i]
    return x

def back_substitution_n(ai,bi):
    #gauss LU linear system of  equation solution
    n=len(ai)
    m=len(bi)
    x=[[0 for i in range(n)] for j in range(m)]
    a=[[0 for i in range(n)] for j in range(n)]
    b=[[0 for i in range(n)] for j in range(m)]
    for i in range(n):
        for j in range(n):
            a[i][j]=ai[i][j]
            b[i][j]=bi[i][j]
    for k in range(m):

        for i in range(n):
            toplam=b[i][k]
            for j in range(i):
                toplam =toplam - a[i][j]*b[j][k]
            b[i][k]=toplam
        x[n-1][k]=b[n-1][k]/a[n-1][n-1];
        for i in range((n-2),-1,-1):
            toplam=0
            for j in range(i+1,n):
                toplam+=a[i][j]*x[j][k]
            x[i][k]=(b[i][k]-toplam)/a[i][i]
    return x

def AXB(a,b):
    n=len(a)
    c=gaussLU(a)
    s=back_substitution(c,b)
    return s;
def inverse(a):
    # Inverse matrix
    n=len(a)
    I1=I(n)
    print("n = ",n,"I1=",I1[0])
    c=gaussLU(a)
    s=back_substitution_n(c,I1)
    return s
a=[[1,2,5],[2,1,2],[3,1,1]]
```

```
b=[1,1,2]
x=AXB(a,b);
print("GaussLU elimination solution vector :\nx = ",x)
```

```
runfile('D:/okul/SCO1/gaussLU.py', wdir='D:/okul/SCO1')
GaussLU elimination solution vector :
x =  [-8.881784197001252e-16, 3.000000000000003, -1.000000000000001]
```

Gauss LU decompositionDolittle method) example in excel:

A

1	2	5
2	1	2
3	1	1

B

1
1
2

c

1	2	5
0	-3	-8
0	-5	-14

2
3

c

1	2	5
0	-3	-8
0	0	-0.6667

1.667

U

1	2	5
0	-3	-8
0	0	-0.6667

L

1		
2	1	
3	1.6667	1

d0 =
d1 =
d2 =

1
1
2

1
-1
0.6667

U

1	2	5
0	-3	-8
0	0	-0.6667

x0 = 1
x1 = -1
x2 = 0.6667

0
3
-1

6.3.2 Craut LU decomposition method

In the Gauss (Dolittle) LU decomposition diagonal of matrix L was setted as 1, In the Craut LU decompositon diagonal of matrix U is given as 1. For example for a 5 by 5 system

$$[L] = \begin{bmatrix} l_{11} & 0 & 0 & 0 & 0 \\ l_{21} & l_{22} & 0 & 0 & 0 \\ l_{31} & l_{32} & l_{33} & 0 & 0 \\ l_{41} & l_{42} & l_{43} & l_{44} & 0 \\ l_{51} & l_{52} & l_{53} & l_{54} & l_{55} \end{bmatrix}$$

$$[U] = \begin{bmatrix} 1 & u_{12} & u_{13} & u_{14} & u_{15} \\ 0 & 1 & u_{23} & u_{24} & u_{25} \\ 0 & 0 & 1 & u_{34} & u_{35} \\ 0 & 0 & 0 & 1 & u_{45} \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Solution process is similar

[A]=[L][U]
[A]{X}={B}
[L][U]{X}={B}
[L]{D}={B}
[U]{X}={D}
Algorithm of Craut is different compare to Dolittle
$l_{i,1} = a_{i,1}$ loop i=1,2,….,n

$u_{1,j} = \dfrac{a_{1,j}}{l_{11}}$ loop i=2,3,….,n

loop j=2,3,….,n – 1

$l_{i,j} = a_{i,j} - \sum_{k=1}^{j-1} l_{i,k} u_{k,j}$ loop k=j,j+2,….,n

$u_{i,j} = \frac{a_{i,j} - \sum_{i=1}^{j-1} l_{j,i} u_{i,k}}{l_{j,j}}$ loop k=j+1,j+2,….,n

End of loop $l_{n,n} = a_{n,n} - \sum_{k=1}^{j-1} l_{n,k} u_{k,n}$

Algorithm is converted to programming code in the following example

6.3.3 cholesky symmetrıc matrix LU decomposition method Cholesky method is similar to Dolittle(Gauss) LU method. This method is applied to symmetric matrix. Therefore, instead of LU, it decomposes to $[L][L]^T$ form.

André-Louis Cholesky

$[A]=[L][L]^T$

For the decomposition process following steps are used

$l_{k,i} = \frac{a_{k,i} - \sum_{j=1}^{i-1} l_{i,j} l_{k,j}}{l_{i,i}}$ k=1,2,….,k-1

And

$$l_{k,k} = \sqrt{a_{k,k} - \sum_{j=0}^{n} l_{k,j} l_{k,j}}$$

Backsubstitution is similar to Dolittle method.

```
# -*- coding: utf-8 -*-
"""
Created on Thu Aug 23 09:46:37 2018

@author: Mustafa Turhan Çoban
"""
from math import *
def choleskyLU(c):
#cholesky symmetric matrix LU
#symmetric matrices only
    factor=0;
    n=len(c);
    d=[[0.0 for i in range(n)] for j in range(n)];
    for i in range(n):
        for j in range(n):
            d[i][j]=c[i][j];
    for k in range(1,n+1):
        for i in range(1,k):
            sum=0;
            for j in range(1,i):
                sum+=d[i-1][j-1]*d[k-1][j-1];
            d[k-1][i-1]=(d[k-1][i-1]-sum)/d[i-1][i-1];
        sum=0.0;
        for j in range(1,k):
            sum+=d[k-1][j-1]*d[k-1][j-1];
        d[k-1][k-1]=sqrt((d[k-1][k-1]-sum));
        for j in range(k+1,n+1):
            d[k-1][j-1]=0.0;
    return d;
```

```
def T(A):
    #transpose matrix
    n=len(A);
    m=len(A[0]);
    B = [[0 for x in range(m)] for y in range(n)];
    for i in range(n):
        for j in range(m):
            B[i][j]=A[j][i];
    return B;

def substitutecholesky(c,b):
  #solving of system of equation by using cholesky decomposed matrix
  U=T(c);
  L=c;
  n=len(b);
  sum=0;
  x=[0.0 for j in range(n)];
  d=[0.0 for j in range(n)];
  #forward substituting
  toplam=0;
  for i in range(n):
      toplam=0;
      for j in range(i):
          toplam+=L[i][j]*d[j];
      d[i]=(b[i]-toplam)/L[i][i];
  #backward substitution
  x[n-1]=d[n-1]/U[n-1][n-1];
  for i in range(n-2,-1,-1):
      toplam=0;
      for j in range(i+1,n):
          toplam+=U[i][j]*x[j];
      x[i]=(d[i]-toplam)/U[i][i];
  return x;

a=[[4.0,-1.0,1.0],[-1.0,4.25,2.75],[1.0,2.75,3.5]];
b=[1.0,1.0,1.0];
L=choleskyLU(a);
print("L=\n",L);
x=substitutecholesky(L,b);
print("x=\n",x);
```

```
runfile('E:/okul/SCO1/choleskyLU.py', wdir='E:/okul/SCO1')
L=
 [[2.0, 0.0, 0.0], [-0.5, 2.0, 0.0], [0.5, 1.5, 1.0]]
x=
 [0.41015625, 0.453125, -0.1875]
```

Let us solve the same problem step by step by hand:

$$\begin{bmatrix} 4 & -1 & 1 \\ -1 & 4.25 & 2.75 \\ 1 & 2.75 & 3.5 \end{bmatrix}$$

$$l_{11} = \sqrt{a_{11}} = \sqrt{4} = 2$$

$$l_{21} = \frac{a_{21}}{l_{11}} = \frac{-1}{2} = -0.5$$

$$l_{22} = \sqrt{a_{22} - l_{21}^2} = \sqrt{4.25 - (-0.5)^2} = 2$$

$$l_{32} = \frac{a_{32} - l_{21} - l_{31}}{l_{22}} = \frac{2.75 - (-0.5) * 0.5}{2} = 1.5$$

$$l_{33} = \sqrt{a_{33} - l_{31}^2 - l_{32}^2} = \sqrt{3.5 - (0.5)^2 - (1.5)^2} = 1$$

so cholesky decomposed L matrix :

$$[L] = \begin{bmatrix} 2 & 0 & 0 \\ -0.5 & 2 & 0 \\ 0.5 & 1.5 & 1 \end{bmatrix}$$

$$[U] = [L]^T = \begin{bmatrix} 2 & -0.5 & 0.5 \\ 0 & 2 & 1.5 \\ 0 & 0 & 1 \end{bmatrix}$$

$$[L]\{D\} = \{B\}$$

$$\begin{bmatrix} 2 & 0 & 0 \\ -0.5 & 2 & 0 \\ 0.5 & 1.5 & 1 \end{bmatrix} \begin{Bmatrix} d_1 \\ d_2 \\ d_3 \end{Bmatrix} = \begin{Bmatrix} 1 \\ 1 \\ 1 \end{Bmatrix}$$

d_1=1/2=0.5

d_2=(1-(-0.5)*0.5)/2=0.625

d_3=(1-0.5*0.5-1.5*0.625)/1=-0.1875

$$[U]\{x\} = \{D\}$$

$$\begin{bmatrix} 2 & -0.5 & 0.5 \\ 0 & 2 & 1.5 \\ 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 0.5 \\ 0.625 \\ -0.1875 \end{Bmatrix}$$

x_3=-0.1875

x_2=(0.625-1.5*(-0.1875))/2=0.453125

x_1=(0.5-(-0.5)*0.453125-0.5*(-0.1875))/2=0.41015625

Cholesky backsubstitution example with excel:

Cholesky decomposition

4	-1	1
-1	4.25	2.75
1	2.75	3.5

LT

2	-0.5	0.5
	2	1.5
		1

L

2			d1	1
-0.5	2		d2	1
0.5	1.5	1	d3	1

d1	0.5
d2	0.625
d3	-0.1875

LT

2	-0.5	0.5	x1	0.5
0	2	1.5	x2	0.625
0	0	1	x3	-0.1875

x1	0.41016
x2	0.45313
x3	-0.1875

6.4 JACOBI GAUSS -SEIDEL AND SUCCESSIVE RELAXATION METHODS

In this section, iterative solution methods will be described. Iterative methods could be important in solution of big matrices. Gauss elimination requires $n^3/3$, Gauss-Jordan method requires $n^3/2$ process. When n is big, the total number of process will climb up. In this case the total number of processes in iterative methods will be much smaller. The simplest of the iterative methods are Jacobi method. In this method, a set of x solution vector is selected and substituted into the system of equation in the following form :

$$x_i^{k+1} = \frac{1}{A_{ij}}\left(b_i - \sum_{j \neq i} A_{ij} x_j^k\right) \quad 1 \leq i \leq n \qquad (3.9.1)$$

When a new set of x values is found, it will be substituted in the the equation as the next iteration step. To ensure the convergence matrix should be diagonally dominant

$$\sum_{i=1}^{n} |A_{ii}| \geq \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} |A_{i,j}| \qquad (3.9.2)$$

Diagonal dominancy is the important condition for all iterative formulations. Unfortunately, very big matrices solved in engineering problems are in general fits to this definition.

In order to reduce number of iteration steps, every new calculated x value in the x vector set can be substited into the equation as soon as it is calculated and is not waited the complete set to be calculated as in Jacobi method. This form of iterative process is called Gauss-Seidel iteration.

Philipp Ludwig von Seidel

$$x_i^{k+1} = \frac{1}{A_{ij}}\left(b_i - \sum_{j=1}^{i-1} A_{ij}x_j^{k+1} - \sum_{j=i+1}^{n} A_{ij}x_j^{k}\right) \quad 1 \le i \le n \qquad (3.9.3)$$

Gauss-Seidel and Jacobi methods can be merged to define an intermediate method between two by defining a relaxation coefficient, α, in the equation.

$$x_i^{k+1} = (1-\alpha)x_i^{k+1} + \frac{\alpha}{A_{ij}}\left(b_i - \sum_{j=1}^{i-1} A_{ij}x_j^{k+1} - \sum_{j=i+1}^{n} A_{ij}x_j^{k}\right) \quad 1 \le i \le n \quad (3.9.4)$$

When α is equal to one equation coverged to the Gauss-Seidel method.

```
# -*- coding: utf-8 -*-
"""
Created on Thu Aug 30 09:51:34 2018

@author: Mustafa Turhan Çoban
"""

def gauss_seidel(a,b,x,lembda):
    lamd=1.2;
    n=len(a);
    imax=500;#maximum number of iterations
    es=1.0e-20;
    dummy=0;
    ea=1e-1;
    sum=0;
    old=0;
    iter=0;
    sentinel=0;
    for i in range(n):
        dummy=a[i][i];
        for j in range(n):
            a[i][j]/=dummy;
        b[i]/=dummy;
    for i in range(n):
        sum=b[i];
        for j in range(n):
            if i!=j:sum-=a[i][j]*x[j];
        x[i]=sum;
    iter=iter+1;
    while iter<imax:
      sentinel=1;
      for i in range(n):
          old=x[i];
          sum=b[i];
          for j in range(n):
              if i!=j:
                  sum-=a[i][j]*x[j];
          x[i]=lamd*sum+(1.0-lamd)*old;
          if sentinel==1 and x[i]!=0.0: ea=abs((x[i]-old)/x[i])*100.0;
          if ea<es: return x;
      iter+=1;
    if iter>=imax: print("Maximum number of iteration is exceed, result might not be valid iter = ",iter);
    return x;
```

```
a=[[3,-0.1,-0.2],[0.1,7,-0.3],[0.3,-0.2,10]];
b=[7.85,-19.3,71.4];
x=[0,0,0];
lamd=0.0;
print("iterative relaxation  method",gauss_seidel(a,b,x,lamd));
```

```
runfile('E:/okul/SCO1/gauss_seidel.py', wdir='E:/okul/SCO1')
iterative relaxation  method [3.0000000000000004, -2.5, 7.000000000000001]
```

Jacobi and Gauss-Seidel methods in excel:

iterative method

3	-0.1	-0.2	x1	7.85
0.1	7	-0.3	x2	-19.3
0.3	-0.2	10	x3	71.4

Jacobi		**Gauss-Seidel**	
Initial guess		Initial guess	
x1	0	x1	0
x2	0	x2	0
x3	0	x3	0
iter	1	iter	1
x1	2.616667	x1	2.616667
x2	-2.75714	x2	-2.79452
x3	7.14	x3	7.00561
iter	2	iter	2
x1	3.000762	x1	2.990557
x2	-2.48852	x2	-2.49962
x3	7.006357	x3	7.000291
iter	3	iter	3
x1	3.000806	x1	3.000032
x2	-2.49974	x2	-2.49999
x3	7.000207	x3	6.999999
iter	4	iter	4
x1	3.000022	x1	3
x2	-2.5	x2	-2.5
x3	6.999981	x3	7

```
a=[[10,2,-1,3,1],[2,10,2,-1,3],[-1,2,10,2,-1],[3,-1,2,10,2],[1,3,-1,2,10]];
b=[5,-2,4,-2,5];
x=[0,0,0,0,0];
lamd=0.0;
print("Gauss seidel \n",gauss_seidel(a,b,x,lamd));
```

```
runfile('D:/okul/SCO1/gauss_seidel.py', wdir='D:/okul/SCO1')
Gauss seidel
[1.0, -1.0, 0.9999999999999998, -1.0, 1.0]
```

6.5 METHODS FOR BAND MATRICES

A three-band matrix can be shown as:

$$\begin{bmatrix} f_1 & g_1 & & & & & & \\ e_2 & f_2 & g_1 & & & & & \\ & e_3 & f_3 & g_1 & & & & \\ & & e_4 & f_4 & g_1 & & & \\ & & & e_5 & f_5 & g_1 & & \\ & & & & . & . & . & \\ & & & & & & f_{n-1} & g_{n-1} \\ & & & & & & e_n & f_n \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ . \\ x_{n-1} \\ x_n \end{Bmatrix} = \begin{Bmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \\ . \\ r_{n-1} \\ r_n \end{Bmatrix}$$

This kind of matrix can be solved by using decomposition back and front substitution processes. Following form of basic solition process is called Thomas algorithm

Decomposition step :

for(int k=1;k<n;k++)

```
    {e[k]=e[k]/f[k-1];
     f[k]=f[k]-e[k]*g[k-1];
    }
```

Forward substitution :

```
  for(int k=1;k<n;k++)
    {r[k]=r[k]-e[k]*r[k-1];
    }
```

Backward substitution:

```
  x[n-1]=r[n-1]/f[n-1];
  for(int k=(n-2);k>=0;k--)
    {x[k]=(r[k]-g[k]*x[k+1])/f[k];}
```

An example with detailed calculation steps:

$$\begin{bmatrix} f_1 & g_1 & 0 \\ e_2 & f_2 & g_2 \\ 0 & e_3 & f_3 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} r_1 \\ r_2 \\ r_3 \end{Bmatrix}$$

$$\begin{bmatrix} 0.8 & -0.4 & 0 \\ -0.4 & 0.8 & -0.4 \\ 0 & -0.4 & 0.8 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 41 \\ 25 \\ 105 \end{Bmatrix}$$

Decomposition:

$$e_k = \frac{e_k}{f_{k-1}}$$

$$f_k = f_k - e_k g_{k-1}$$

$$e_2 = \frac{e_2}{f_1} = \frac{-0.4}{0.8} = -0.5$$

$$f_2 = f_2 - e_2 g_1 = 0.8 - (-0.5)(-0.4) = 0.6$$

$$e_3 = \frac{e_3}{f_3} = \frac{-0.4}{0.6} = -0.666666666$$

$$f_3 = f_3 - e_3 g_3 = 0.8 - (-0.666666666)(-0.4) = 0.533333333$$

Matrix took the form of

$$\begin{bmatrix} 0.8 & -0.4 & 0 \\ -0.4 & 0.6 & -0.4 \\ 0 & -0.666666666 & 0.533333333 \end{bmatrix}$$

LU decomposed form of the matrix :

$$[A] = [L][U] = \begin{bmatrix} 1 & 0 & 0 \\ -0.5 & 1 & 0 \\ 0 & -0.66666 & 1 \end{bmatrix} \begin{bmatrix} 0.8 & -0.4 & 0 \\ 0 & 0.6 & -0.4 \\ 0 & 0 & 0.53333 \end{bmatrix}$$

Forward substitution

$$[L]\{D\} = \{B\}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ -0.5 & 1 & 0 \\ 0 & -0.66666 & 1 \end{bmatrix} \begin{Bmatrix} d_1 \\ d_2 \\ d_3 \end{Bmatrix} = \begin{Bmatrix} 41 \\ 25 \\ 105 \end{Bmatrix}$$

d_1=41

d_2=25-(-0.5)*41=45.5

d_3=105-(-0.66666)*45.5=135.3333333

Backward substitution

$$[U]\{x\} = \{D\}$$

$$\begin{bmatrix} 0.8 & -0.4 & 0 \\ 0 & 0.6 & -0.4 \\ 0 & 0 & 0.53333 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 41 \\ 45.5 \\ 135.33333 \end{Bmatrix}$$

X_3=135.33333/0.53333=253.75

X_2=(45.5-(-0.4)* 253.75)/0.6=245

X_1=(41-(-0.4)*245)/0.8=173.75

Python version of the Thomas algorithm is given below:

Program 6.5.1 Thomas algorithm for band matrices (python version)

```
# -*- coding: utf-8 -*-
"""
Created on Sun Jun  3 21:38:38 2018

@author: Mustafa Turhan Çoban
"""
def thomas(a,r):
    n=len(a);
    f=[0.0 for i in range(n)];
    e=[0.0 for i in range(n)];
    g=[0.0 for i in range(n)];
    x=[0.0 for i in range(n)];
    for i in range(0,n): f[i]=a[i][i];
    for i in range(0,n-1): g[i]=a[i][i+1];
    for i in range(0,n-1): e[i+1]=a[i+1][i];
    for k in range(0,n): e[k]=e[k]/f[k-1];f[k]=f[k]-e[k]*g[k-1];
    for k in range(1,n): r[k]=r[k]-e[k]*r[k-1];
    x[n-1]=r[n-1]/f[n-1];
    for k in range(n-2,-1,-1):  x[k]=(r[k]-g[k]*x[k+1])/f[k];
    return x;

def thomas_n(f,e,g,r):
    n=len(f);
    x=[0.0 for i in range(n)];
    for k in range(0,n): e[k]=e[k]/f[k-1];f[k]=f[k]-e[k]*g[k-1];
    for k in range(1,n): r[k]=r[k]-e[k]*r[k-1];
    x[n-1]=r[n-1]/f[n-1];
    for k in range(n-2,-1,-1):  x[k]=(r[k]-g[k]*x[k+1])/f[k];
    return x;

a=[[0.8,-0.4,0],[-0.4,0.8,-0.4],[0,-0.4,0.8]];
b=[41,25,105];
x=thomas(a,b);
print("x=",x);
```

```
runfile('D:/okul/SCO1/thomas.py', wdir='D:/okul/SCO1')
x= [173.75, 244.99999999999997, 253.74999999999997]
```

6.6 CONJUGATE GRADIENT METHOD

If vector x^* is a solution of positive and well defined system of equation Ax = b, this solution is valid when the vector x^* is defined as minimum of function $g(x) = x^T Ax - 2b^T x$. The x vector is defined as a different vector then the solution vector x* . In addition a v vector with values different than 0 and a real number α is defined.

$$g(x + \alpha v) = g(x) + 2\alpha[v^T(Ax - B)] + \alpha^2[v^T(Av)]$$

In this equation, if v and x vector is known, for some value of α function will be minimum (One dimensional minimisation problem). If it is assumed that only independent variable of the function is α $h(\alpha) = g(x + \alpha v)$ can be written. The minimum of the function, can be defined as the root of the derivative of the function with respect to α

$$h'^{(\alpha)} = 2[v^T(Ax - B)] + 2\alpha[v^T(Av)] = 0$$

From this equation

$$\alpha = \frac{v^T(Ax - B)}{v^T(Av)}$$

To simplify the equation a little bit further definition $r = b - Ax$
r = b – A x can be applied. If this definition is substituted into the previous equation:

$$\alpha = \frac{v^T r}{v^T(Av)}$$

If α is substituted into the iterative process iterative process

$$x_k = x_{k-1} + \alpha_k v_k$$

and for $r_k = b - Ax_k$ value would be

$$\alpha_k = \frac{v^T r_k}{v_k^T(Av_k)}$$

New x_k vector should be give solution to the $Ax_k = b$ equation . When a solution is obtained, it is known that equation will converged to

$$Ax_k = A(x_{k-1} + \alpha_k v_k) = b$$

But we still do not know the search direction vector, v_k . As a starting search direction **steepest descent** optimization direction can be selected. This direction is defined as

$$\nabla g(x) = 2(ax - b) = -2r$$

In this case v_{k+1} can be selected as

$$v_{k+1} = -\nabla g(x_k) = r_k$$

This direction gives the maximum negative change in $g(x_k)$. An alternative selection to the direction vector can be carried out such a vector that for this vector condition $v_i^T A v_j = 0 \quad i \neq j$ will be valid for a non-zero v vector. This type of vectors are called orthogonal vectors. If some orthogonal v vectors can be found, it will simplify our solution. In orthogonal sets terms out of diagonal will be zero so only diagonal terms will remain in the equation. Therefore process of solving matrix system will be converted to simple multiplication and division processes.
The orthogonal set suggested by Hesteness and Stiefel has the following form :

$$p_k = \frac{r_k^T r_k}{r_{k-1}^T r_{k-1}}$$

$$v_{k+1} = r_k + p_k v_k$$

As it is seen from the equation, values are derived from the previous sets. In summary this method can be applied as:
In the first iteration steepest descent value is taken.
$r_0 = b - Ax_0$ and $v_0 = r_0$(steepest descent value) .
then for the next iteration sets calculates the values as
for k=1 to n

$$\alpha_k = \frac{r_{k-1}^T r_k}{v_k^T A v_k}$$

$$x_k = x_{k-1} + \alpha_k v_k$$

$$r_k = b - Ax_k$$

$$p_k = \frac{r_k^T r_k}{r_{k-1}^T r_{k-1}}$$

$$v_{k+1} = r_k + p_k v_k$$

As it is seen process is quite a complex process compared to Gauss elimination type of solution. It is not preffered to Gauss elimination in solution of the small systems, but when the system becomes larger, it will be prefferable.

Numerical example

To illustrate the conjugate gradient method, we will complete a simple example. Considering the linear system Ax = b given by

$$Ax = \begin{bmatrix} 4 & 1 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

we will perform two steps of the conjugate gradient method beginning with the initial guess

$$x_0 = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

in order to find an approximate solution to the system.

Solution

```
# -*- coding: utf-8 -*-
"""
Created on Sun Jan  2 06:44:53 2022
CONUGATE GRADIENT METHOD
@author: Mustafa Turhan ÇOBAN
"""

from math import *
def VT_V(left):
    #multiplies a vector transpose with a vector
    n=len(left);
    tot=0;
    for i in range(n): tot+=left[i]*left[i];
    return tot;
```

```
def V_VT(left):
    #multiplys a vector with a vector              transpose
    n=len(left);
    aa = [[0 for x in range(n)] for y in range(n)];
    for i in range(n):
        for j in range(n): aa[i][j]=left[i]*left[j];
    return aa;

def  VT_X(left,right):
    #multiplys a vector transpose with a vector
    n=len(left)
    tot=0.0
    for i in range(n):
        l1=left[i]
        l2=right[i]
        tot=tot+left[i]*right[i]
    return tot

def multiply_c_m(left,right):
    #multiplying a matrix with a constant
     n=len(rigth);
     m=len(right[0]);
     b = [[0 for x in range(m)] for y in range(n)];
     for i in range(n):
         for j in range(m):
             b[i][j]=right[i][j]*left;
     return b;

def multiply_v_m(left,right):
    #multiplication of one vector with one matrix
    m2=len(right[0]);
    n2=len(right);
    m1=len(left);
    b = [0.0 for i in range(m1)];
    if n2 != m1:
        print("inner matrix dimensions must agree");
        for ii in range(n2):b[ii]=0;
        return b;
    for i in range(m2):
        b[i]=0;
        for k in range(m1):
            b[i]+=right[i][k]*left[k];
    return b;

def multiply_m_v(left,right):
    #multiplication of one matrix with one vector
    m1=len(left[0]);
    n1=len(left);
    m2=len(right);
    b = [0.0 for i in range(m2)];
    if n1 != m2:
        print("inner matrix dimensions must agree");
        for ii in range(n1):b[ii]=0;
        return b;
    for i in range(m1):
        b[i]=0;
        for k in range(n1):
            b[i]+=left[i][k]*right[k];
    return b;

def multiply_c_v(left,right):
    #multiplying a vector with a constant
    n=len(right);
    b = [0.0 for i in range(n)];
    for i in range(n):
        b[i]=left*right[i];
    return b;

def substract(left,right):
    #addition of two vectors
    n1=len(left);
    n2=len(right);
    if n1>=n2: nMax=n1;
    else     : nMax=n2;
    b = [0.0 for i in range(nMax)];
    for i in range(n1):
```

```
        b[i]=b[i]+left[i];
    for i in range(n2):
        b[i]=b[i]-right[i];
    return b;

def add(left,right):
    #addition of two vectors
    n1=len(left);
    n2=len(right);
    if n1>=n2: nMax=n1;
    else     : nMax=n2;
    b = [0.0 for i in range(nMax)];
    for i in range(n1):
        b[i]=b[i]+left[i];
    for i in range(n2):
        b[i]=b[i]+right[i];
    return b;

def equals(left):
    #addition of two vectors
     n1=len(left);
     b = [0.0 for i in range(n1)];
     for i in range(n1):
         b[i]=left[i];
     return b;

def norm(v):
    return sqrt(normkare(v));

def normkare(v):
    #vector norm
    total=0;
    n=len(v);
    for i in range(n):
        total+=v[i]*v[i];
    return total;

def VT_G_V(V,G):
  x1=multiply_v_m(V,G);
  return VT_X(V,x1);

def I(n):
    #unit matrix
    b = [0.0 for i in range(n)];
    for i in range(n):
        b[i][i]=1.0;
    return b;

def D(A):
    #Diagonal elements of matrix
    n=len(A);
    b = [[0 for x in range(n)] for y in range(n)];
    for i in range(n):
        b[i][i]=A[i][i];
    return b;


def conjugate_gradient(x,A,b):
    n=len(A);
    return conjugate_gradient1(x,A,b);

def conjugate_gradient(A,b):
    n=len(A);
    x = [0.0 for i in range(n)];
    return conjugate_gradient1(x,A,b);

def conjugate_gradient1(x,A,b):
    #iterative system of equation solution
    #x is the initial guess
    #C is the conditioning matrix
    eps=1.0e-15;
    maxiter=500;
    n=len(x);
    # x = [0.0 for i in range(n)];
    r = [0.0 for i in range(n)]
    v = [0.0 for i in range(n)]
```

```
    k=0
    r=substract(b,multiply_m_v(A,x))
    print("A=\n",A,"b=\n",b,"x=\n",x,"r=",r)
    v=equals(r)
    beta=0
    nr=norm(r)
    while (nr>eps and k<maxiter):
        k=k+1;
        if norm(v)<eps: return x
        re=equals(r)
        alpha=VT_V(re)/VT_G_V(v,A)
        print("alpha=",alpha)
        x=add(x,multiply_c_v(alpha,v));
        r=substract(r,multiply_v_m(multiply_c_v(alpha,v),A))
        print("k=",k,"r=",r)
        nr=norm(r);
        if beta<eps:
            nr=norm(r);
            if nr<eps: return x;
        p=VT_V(r)/VT_V(re)
        print("p=",p,"v=",v)
        v=add(r,multiply_c_v(p,v));
    return x;

A=[[4.0,1.0],[1.0,3.0]]
b=[1.0,2.0]
x=[0.0,0.0]
x=conjugate_gradient1(x,A,b)
print("conjugate gradient method : \n",x)
```

```
runfile('D:/okul/SCO1/conjuge_gradient1.py', wdir='D:/okul/SCO1')
A=
 [[4.0, 1.0], [1.0, 3.0]] b=
 [1.0, 2.0] x=
 [0.0, 0.0] r= [1.0, 2.0]
alpha= 0.25
k= 1 r= [-0.5, 0.25]
p= 0.0625 v= [1.0, 2.0]
alpha= 0.36363636363636365
k= 2 r= [0.0, 2.7755575615628914e-17]
conjugate gradient method :
 [0.09090909090909091, 0.6363636363636364]
```

Let us apply to another example:

```
# -*- coding: utf-8 -*-
"""
Created on Sun Jan  2 06:44:53 2022
CONUGATE GRADIENT METHOD
@author: Mustafa Turhan ÇOBAN
"""

from math import *
def VT_V(left):
    #multiplies a vector transpose with a vector
    n=len(left);
    tot=0;
    for i in range(n): tot+=left[i]*left[i];
    return tot;

def V_VT(left):
    #multiplys a vector with a vector              transpose
    n=len(left);
    aa = [[0 for x in range(n)] for y in range(n)];
    for i in range(n):
        for j in range(n): aa[i][j]=left[i]*left[j];
    return aa;

def  VT_X(left,right):
    #multiplys a vector transpose with a vector
    n=len(left)
    tot=0.0
    for i in range(n):
        l1=left[i]
```

```
        l2=right[i]
        tot=tot+left[i]*right[i]
    return tot

def multiply_c_m(left,right):
    #multiplying a matrix with a constant
     n=len(rigth);
     m=len(right[0]);
     b = [[0 for x in range(m)] for y in range(n)];
     for i in range(n):
         for j in range(m):
             b[i][j]=right[i][j]*left;
     return b;

def multiply_v_m(left,right):
    #multiplication of one vector with one matrix
    m2=len(right[0]);
    n2=len(right);
    m1=len(left);
    b = [0.0 for i in range(m1)];
    if n2 != m1:
        print("inner matrix dimensions must agree");
        for ii in range(n2):b[ii]=0;
        return b;
    for i in range(m2):
        b[i]=0;
        for k in range(m1):
            b[i]+=right[i][k]*left[k];
    return b;

def multiply_m_v(left,right):
    #multiplication of one matrix with one vector
    m1=len(left[0]);
    n1=len(left);
    m2=len(right);
    b = [0.0 for i in range(m2)];
    if n1 != m2:
        print("inner matrix dimensions must agree");
        for ii in range(n1):b[ii]=0;
        return b;
    for i in range(m1):
        b[i]=0;
        for k in range(n1):
            b[i]+=left[i][k]*right[k];
    return b;

def multiply_c_v(left,right):
    #multiplying a vector with a constant
    n=len(right);
    b = [0.0 for i in range(n)];
    for i in range(n):
        b[i]=left*right[i];
    return b;

def substract(left,right):
    #addition of two vectors
    n1=len(left);
    n2=len(right);
    if n1>=n2: nMax=n1;
    else     : nMax=n2;
    b = [0.0 for i in range(nMax)];
    for i in range(n1):
        b[i]=b[i]+left[i];
    for i in range(n2):
        b[i]=b[i]-right[i];
    return b;

def add(left,right):
    #addition of two vectors
    n1=len(left);
    n2=len(right);
```

```
    if n1>=n2: nMax=n1;
    else     : nMax=n2;
    b = [0.0 for i in range(nMax)];
    for i in range(n1):
        b[i]=b[i]+left[i];
    for i in range(n2):
        b[i]=b[i]+right[i];
    return b;

def equals(left):
    #addition of two vectors
     n1=len(left);
     b = [0.0 for i in range(n1)];
     for i in range(n1):
         b[i]=left[i];
     return b;

def norm(v):
    return sqrt(normkare(v));

def normkare(v):
    #vector norm
    total=0;
    n=len(v);
    for i in range(n):
        total+=v[i]*v[i];
    return total;

def VT_G_V(V,G):
  x1=multiply_v_m(V,G);
  return VT_X(V,x1);

def I(n):
    #unit matrix
    b = [0.0 for i in range(n)];
    for i in range(n):
        b[i][i]=1.0;
    return b;

def D(A):
    #Diagonal elements of matrix
    n=len(A);
    b = [[0 for x in range(n)] for y in range(n)];
    for i in range(n):
        b[i][i]=A[i][i];
    return b;


def conjugate_gradient(x,A,b):
    n=len(A);
    return conjugate_gradient1(x,A,b);

def conjugate_gradient(A,b):
    n=len(A);
    x = [0.0 for i in range(n)];
    return conjugate_gradient1(x,A,b);

def conjugate_gradient1(x,A,b):
    #iterative system of equation solution
    #x is the initial guess
    #C is the conditioning matrix
    eps=1.0e-15;
    maxiter=500;
    n=len(x);
    # x = [0.0 for i in range(n)];
    r = [0.0 for i in range(n)]
    v = [0.0 for i in range(n)]
    k=0
    r=substract(b,multiply_m_v(A,x))
    v=equals(r)
    beta=0
```

```
    nr=norm(r)
    while (nr>eps and k<maxiter):
        k=k+1;
        if norm(v)<eps: return x
        re=equals(r)
        x1=VT_V(re)
        x2=VT_G_V(v,A)
        alpha=VT_V(re)/VT_G_V(v,A)
        x=add(x,multiply_c_v(alpha,v))
        r=substract(r,multiply_v_m(multiply_c_v(alpha,v),A))
        nr=norm(r);
        if beta<eps:
            nr=norm(r);
            if nr<eps: return x;
        p=VT_V(r)/VT_V(re)
        v=add(r,multiply_c_v(p,v));
    return x;

A=[[0.2,0.1,1,1,0],[0.1,4,-1,1,-1],[1,-1,60,0,-2],[1,1,0,8,4],[0,-1,-2,4,700]]
b=[1.0,2.0,3.0,4.0,5.0]
x=conjugate_gradient(A,b);
print("conjugate gradient method : \n",x)
```

```
runfile('D:/okul/SCO1/conjuge_gradient1.py', wdir='D:/okul/SCO1')
conjugate gradient method :
 [7.85971307544586, 0.42292640829500766, -0.07359223902404631, -0.5406430168946267, 0.010626162854036317]
```

The result, $\mathbf{x}_2$, is a "better" approximation to the system's solution than $\mathbf{x}_1$ and $\mathbf{x}_0$. If exact arithmetic were to be used in this example instead of limited-precision, then the exact solution would theoretically have been reached after $n = 2$ iterations (n being the order of the system).

We will now extend the conjugate gradient method to include ***preconditioning***. If the matrix A is ill-conditioned, the conjugate gradient method is highly susceptible to rounding errors. So, although the exact answer should be obtained in n steps, this is not usually the case. As a direct method the conjugate gradient method is not as good as Gaussian elimination with pivoting. The main use of the conjugate gradient method is as an iterative method applied to a better-conditioned system. To apply the method to a better-conditioned system, we want to select a nonsingular conditioning matrix C so that system of equation can be converted to a better conditioned system.

$$A^* = C^{-1}AC^{-1} \qquad x^* = C^{-1}x \qquad b^* = C^{-1}b \quad A^*x^* = b^*$$

As a C conditioning matrix, a matrix with the rootsquare values of the diagonal elements and 0 values for the rest of the elements can be selected. For such a matrix, inverse matrix can be produced just by dividing numbers in the matrix by 1. Another possibility for the conditioning matrix could be unit matrix.

If the whole procedure is given as a recipe:

Input : Matrix A,b and x , C^{-1} inverse of positive conditioning matrix

Step 1 : $r_0 = b - A\,x$

$w_0 = C^{-1}r_0$

$v_0 = C^{-1}\,w_0$

$$\gamma_0 = \sum_{j=1}^{n} w_j^2$$

Step 2 : k=1

Step 3 : while(k<=N) do adim 4-5

Step 4 : $u=Av_k$

$$\alpha_k = \frac{\gamma_{k-1}}{\sum_{j=1}^{n} v_j u_j}$$

$x_k=x_{k-1} + \alpha_k v_k$

$r_k=r_{k-1} + \alpha_k u_k$

$w_k = C^{-1}r_k$

$$\gamma_k = \sum_{j=1}^{n} w_j^2$$

Step 5 : $p_k = \beta_k/\gamma_k$

$v_{k+1} = C^{-1} w_k + p_k v_k$

$\gamma_k = \beta_k$

$k=k+1$

Conditioned conjugate-gradient method to solve system of equations (python version)

```
# -*- coding: utf-8 -*-
"""
Created on Thu Aug 23 11:49:55 2018

@author: Mustafa Turhan Çoban
"""
from math import *
def VT_V(left):
    #multiplies a vector transpose with a vector
    n=len(left);
    tot=0;
    for i in range(n): tot+=left[i]*left[i];
    return tot;

def V_VT(left):
    #multiplys a vector with a vector            transpose
    n=len(left);
    aa = [[0 for x in range(n)] for y in range(n)];
    for i in range(n):
        for j in range(n): aa[i][j]=left[i]*left[j];
    return aa;

def  VT_X(left,right):
    #multiplys a vector transpose with a vector
    n=len(left);
    tot=0;
    for i in range(n):tot+=left[i]*right[i];
    return tot;

def multiply_c_m(left,right):
    #multiplying a matrix with a constant
     n=len(rigth);
     m=len(right[0]);
     b = [[0 for x in range(m)] for y in range(n)];
     for i in range(n):
         for j in range(m):
             b[i][j]=right[i][j]*left;
     return b;

def multiply_v_m(left,right):
    #multiplication of one vector with one matrix
    m2=len(right[0]);
    n2=len(right);
    m1=len(left);
    b = [0.0 for i in range(m1)];
    if n2 != m1:
        print("inner matrix dimensions must agree");
        for ii in range(n2):b[ii]=0;
        return b;
    for i in range(m2):
        b[i]=0;
        for k in range(m1):
            b[i]+=right[i][k]*left[k];
    return b;

def multiply_m_v(left,right):
    #multiplication of one matrix with one vector
    m1=len(left[0]);
    n1=len(left);
    m2=len(right);
    b = [0.0 for i in range(m2)];
    if n1 != m2:
        print("inner matrix dimensions must agree");
        for ii in range(n1):b[ii]=0;
        return b;
    for i in range(m1):
        b[i]=0;
```

```
        for k in range(n1):
            b[i]+=left[i][k]*right[k];
    return b;

def multiply_c_v(left,right):
    #multiplying a vector with a constant
    n=len(right);
    b = [0.0 for i in range(n)];
    for i in range(n):
        b[i]=left*right[i];
    return b;

def substract(left,right):
    #addition of two vectors
    n1=len(left);
    n2=len(right);
    if n1>=n2: nMax=n1;
    else     : nMax=n2;
    b = [0.0 for i in range(nMax)];
    for i in range(n1):
        b[i]=b[i]+left[i];
    for i in range(n2):
        b[i]=b[i]-right[i];
    return b;

def add(left,right):
    #addition of two vectors
    n1=len(left);
    n2=len(right);
    if n1>=n2: nMax=n1;
    else     : nMax=n2;
    b = [0.0 for i in range(nMax)];
    for i in range(n1):
        b[i]=b[i]+left[i];
    for i in range(n2):
        b[i]=b[i]+right[i];
    return b;

def equals(left):
    #addition of two vectors
     n1=len(left);
     b = [0.0 for i in range(n1)];
     for i in range(n1):
         b[i]=left[i];
     return b;

def norm(v):
    return sqrt(normkare(v));

def normkare(v):
    #vector norm
    total=0;
    n=len(v);
    for i in range(n):
        total+=v[i]*v[i];
    return total;

def VT_G_V(V,G):
  x1=multiply_v_m(V,G);
  return VT_X(V,x1);

def I(n):
    #unit matrix
    b = [0.0 for i in range(n)];
    for i in range(n):
        b[i][i]=1.0;
    return b;

def D(A):
    #Diagonal elements of matrix
    n=len(A);
    b = [[0 for x in range(n)] for y in range(n)];
    for i in range(n):
        b[i][i]=A[i][i];
    return b;
```

```
def invsqrtD(A):
    #square root of Diagonal elements of a matrix
    n=len(A);
    b = [[0 for x in range(n)] for y in range(n)];
    for i in range(n):
        b[i][i]=1.0/sqrt(A[i][i]);
    return b;

def conjugate_gradient(x,A,b):
    n=len(A);
    C = [[0 for x in range(n)] for y in range(n)];
    C=invsqrtD(A);
    return conjugate_gradient1(x,A,b,C);

def conjugate_gradient(A,b):
    n=len(A);
    C = [[0 for x in range(n)] for y in range(n)];
    C=invsqrtD(A);
    x = [0.0 for i in range(n)];
    return conjugate_gradient1(x,A,b,C);

def conjugate_gradient1(x,A,b,C):
    #iterative system of equation solution
    #x is the initial guess
    #C is the conditioning matrix
    eps=1.0e-15;
    maxiter=500;
    n=len(x);
    x = [0.0 for i in range(n)];
    r = [0.0 for i in range(n)];
    v = [0.0 for i in range(n)];
    w = [0.0 for i in range(n)];
    u = [0.0 for i in range(n)];
    k=0;
    r=substract(b,multiply_m_v(A,x));
    w=multiply_m_v(C,r);
    v=multiply_m_v(C,w);
    beta=0;
    gamma=normkare(w);
    nr=norm(r);
    while (nr>eps and k<maxiter):
        k=k+1;
        if norm(v)<eps: return x;
        re=equals(r);
        u=multiply_m_v(A,v);
        alpha=gamma/VT_X(v,u);
        x=add(x,multiply_c_v(alpha,v));
        r=substract(r,multiply_c_v(alpha,u));
        nr=norm(r);
        w=multiply_m_v(C,r);
        beta=normkare(w);
        if beta<eps:
            nr=norm(r);
            if nr<eps: return x;
        p=beta/gamma;
        v=add(multiply_m_v(C,w),multiply_c_v(p,v));
        gamma=beta;
    return x;

A=[[0.2,0.1,1,1,0],[0.1,4,-1,1,-1],[1,-1,60,0,-2],[1,1,0,8,4],[0,-1,-2,4,700]];
b=[1,2,3,4,5];
x=conjugate_gradient(A,b);
print("conditioned conjugate gradient method : \n",x);
```

```
runfile('D:/okul/SCO1/conjugate_gradient.py', wdir='D:/okul/SCO1')
conditioned conjugate gradient method :
 [7.859713075445861, 0.4229264082950077, -0.07359223902404635, -0.5406430168946268, 0.010626162854036319]
```

6.7 UNIT MATRIX, TRANPOSE MATRIX, DETERMINANT AND MATRIX INVERSION

Unit matrix is a matrix with all diagonal elements are one and all the remaining elements are zero

$$a_{ii} = 1$$
$$a_{ij,i\neq j} = 0$$

As an example for n=4

$$[I] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Transpose matrix is the matrix with rows and colums are interchanged. Mathematically, it can be shown with a T sign at the top of the matrix. Transpose of a matrix can be taken as in the following method:

```
def Transpose(A):
    n=len(A)
    B=[[0 for i in range(n)] for j in range(n)]
    for i in range(n):
        for j in range(n):
            B[j][i]=A[i][j]
    return B

A=[[1,2,3],[4,5,6],[7,8,9]]
B=Transpose(A)
print("A=",A)
print("B=",B)
```

```
runfile('E:/okul/SCO1/Transpose.py', wdir='E:/okul/SCO1')
A= [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
B= [[1, 4, 7], [2, 5, 8], [3, 6, 9]]
```

Inverse matrix is the matrix when multiplied with the original matrix will give the unit matrix

$$[A][A]^{-1} = [I]$$

```
 # Gauss(DooLittle) LU decomposition
# python version
from math import *
# Gauss(DoLittle) LU decomposition
def gaussLU(ai):
    #Gauss elimination with pivoting
  n=len(ai)
  x=[0 for i in range(n)]
  a=[[0 for i in range(n)] for j in range(n)]
  carpan=0
  toplam=0
  for i in range(n):
      for j in range(n):
          a[i][j]=ai[i][j]
  for k in range(n-1):
      for i in range(k+1,n):
          carpan=a[i][k]/a[k][k];
          a[i][k]=carpan
          for j in range(k+1,n):
              a[i][j]-=carpan*a[k][j]
  return a

# inverse matrix
def I(n):
    #unit matrix of dimension n
    B = [[0 for x in range(n)] for y in range(n)];
    for i in range(0,n):
        for j in range(0,n):
            if i==j: B[i][j]=1.0;
    return B;
# bi is vector
def back_substitution(ai,bi):
    #gauss LU linear system of  equation solution
    n=len(ai)
    x=[0 for i in range(n)]
    a=[[0 for i in range(n)] for j in range(n)]
    b=[0 for i in range(n)]
    for i in range(n):
        for j in range(n):
            a[i][j]=ai[i][j]
        b[i]=bi[i]
```

```
    for i in range(n):
        toplam=b[i]
        for j in range(i):
            toplam-=a[i][j]*b[j]
        b[i]=toplam
    x[n-1]=b[n-1]/a[n-1][n-1];
    for i in range((n-2),-1,-1):
        toplam=0
        for j in range(i+1,n):
           toplam+=a[i][j]*x[j]
        x[i]=(b[i]-toplam)/a[i][i]
    return x
# bi is matrix
def back_substitution_n(ai,bi):
    #gauss LU linear system of  equation solution
    n=len(ai)
    m=len(bi)
    x=[[0 for i in range(n)] for j in range(m)]
    a=[[0 for i in range(n)] for j in range(n)]
    b=[[0 for i in range(n)] for j in range(m)]
    for i in range(n):
        for j in range(n):
            a[i][j]=ai[i][j]
            b[i][j]=bi[i][j]
    for k in range(m):

        for i in range(n):
            toplam=b[i][k]
            for j in range(i):
                toplam =toplam - a[i][j]*b[j][k]
            b[i][k]=toplam
        x[n-1][k]=b[n-1][k]/a[n-1][n-1];
        for i in range((n-2),-1,-1):
            toplam=0
            for j in range(i+1,n):
               toplam+=a[i][j]*x[j][k]
            x[i][k]=(b[i][k]-toplam)/a[i][i]
    return x

def AXB(a,b):
    n=len(a)
    c=gaussLU(a)
    s=back_substitution(c,b)
    return s;
def inverse(a):
    # Inverse matrix
    n=len(a)
    I1=I(n)
    c=gaussLU(a)
    s=back_substitution_n(c,I1)
    return s

A=[[-8.0,1.0,2.0],[5.0,7.0,-3.0],[2.0,1.0,-2.0]]
#A=[[3.556,-1.778,0],[-1.778,3.556,-1.778],[0,-1.778,3.556]]
print("matrix = \n",A);
AI=inverse(A)
print("inverse matrix = \n",AI);
```

```
runfile('D:/okul/SCO1/gaussLU.py', wdir='D:/okul/SCO1')
matrix =
 [[-8.0, 1.0, 2.0], [5.0, 7.0, -3.0], [2.0, 1.0, -2.0]]
inverse matrix =
 [[-0.14864864864864866, 0.05405405405405404, -0.22972972972972974], [0.05405405405405406, 0.16216216216216214, -
0.1891891891891892], [-0.12162162162162161, 0.13513513513513511, -0.8243243243243243]]
```

```
A=[[3.556,-1.778,0],[-1.778,3.556,-1.778],[0,-1.778,3.556]]
print("matrix = \n",A);
AI=inverse(A)
```

```
runfile('D:/okul/SCO1/gaussLU.py', wdir='D:/okul/SCO1')
matrix =
 [[3.556, -1.778, 0], [-1.778, 3.556, -1.778], [0, -1.778, 3.556]]
inverse matrix =
 [[0.421822272215973, 0.281214848143982, 0.140607424071991], [0.281214848143982, 0.562429696287964, 0.281214848143982],
[0.140607424071991, 0.281214848143982, 0.421822272215973]]
```

Determinant of a matrix can be defined as follows: if a matrix A is given as

$$[A] = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix}$$

Determinat of this matrix determined as:

$$D[A] = \begin{vmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{vmatrix} = a_{00}\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} - a_{01}\begin{vmatrix} a_{10} & a_{12} \\ a_{20} & a_{22} \end{vmatrix} + a_{02}\begin{vmatrix} 0 & a_{11} \\ a_{20} & a_{21} \end{vmatrix}$$

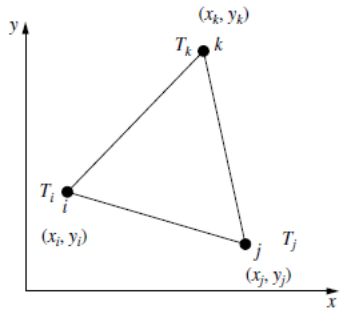
$$D[A] = a_{00}(a_{11}a_{22} - a_{12}a_{21}) - a_{01}(a_{10}a_{22} - a_{12}a_{20}) + a_{02}(a_{10}a_{22} - a_{12}a_{20})$$

If matrix is converted to a upper triangular form Determinant will simply be the product of diagonals

$$[A] = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ 0 & a_{11} & a_{12} \\ 0 & 0 & a_{22} \end{bmatrix}$$

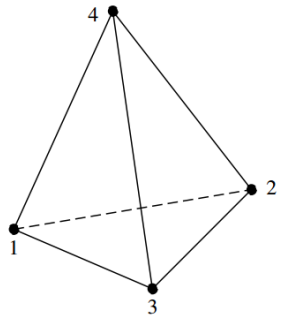
$$D[A] = a_{00}a_{11}a_{11}$$

Additional note: Area of a triangle



$$2A = det\begin{bmatrix} 1 & x_i & y_i \\ 1 & x_j & y_j \\ 1 & x_k & y_k \end{bmatrix} = (x_i y_j - x_j y_i) + (x_k y_j - x_i y_k) + (x_j y_k - x_k y_j)$$

Volume of a tetrahedron



$$6V = det\begin{bmatrix} 1 & x_1 & y_1 & z_1 \\ 1 & x_2 & y_2 & z_2 \\ 1 & x_3 & y_3 & z_3 \\ 1 & x_4 & y_4 & z_4 \end{bmatrix}$$

```
def D(a):
  n=len(a)
  D=1.0
  #gauss elimination
  for k in range(n):
    for i in range(k+1,n):
      carpan=a[i][k]/a[k][k]
      a[i][k]=0;
      for j in range(k+1,n):
        a[i][j]=a[i][j]-carpan*a[k][j]
  for i in range(n):
     D*=a[i][i]
  return D

A=[[1,0,0],[1,1,0],[1,0,1]]
```

```
Area=0.5*D(A)
print("Area=",Area)
```

```
runfile('E:/okul/SCO1/Determinant.py', wdir='E:/okul/SCO1')
Area= 0.5
```

CRAMER'S RULE

Another method to solve linear system of equations is Cramer's rule. The rule based on solutions of determinants. This solution method is not preferred much as a computer solution method, but it is practical for hand solutions of small systems. If a system of equation given as:

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{Bmatrix} = \begin{Bmatrix} b_1 \\ b_2 \\ \dots \\ b_n \end{Bmatrix}$$

then solution can be given as:

$$D = \begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{vmatrix} \quad x_1 = \begin{vmatrix} b_1 & a_{12} & \dots & a_{1n} \\ b_2 & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ b_n & a_{n2} & \dots & a_{nn} \end{vmatrix} \frac{1}{D} \quad x_1 = \begin{vmatrix} a_{11} & b_1 & \dots & a_{1n} \\ a_{21} & b_2 & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & b_n & \dots & a_{nn} \end{vmatrix} \frac{1}{D} \quad \dots.$$

Due to fact that solution of the determinat requires solution of the system of equations in computer environment. Cramer's rule is not very practical numerical method to solve system of equations, but could be an easy solution method for hand calculations.

PROBLEMS

PROBLEM 1

Solve system of equation by using Gauss elimination method.

$$\begin{bmatrix} 7 & 1 & 2 \\ -1 & 4 & -1 \\ 3 & 15 & 20 \end{bmatrix} \begin{Bmatrix} x_0 \\ x_1 \\ x_2 \end{Bmatrix} = \begin{Bmatrix} 47 \\ 19 \\ 87 \end{Bmatrix}$$

PROBLEM 2

Solve system of equation by using

a) By using Gauss elimination
b) By using Gauss-Jordan elimination

$$\begin{bmatrix} 1.1348 & 3.8226 & 1.1651 & 3.4017 \\ 0.5301 & 1.7875 & 2.5330 & 1.5435 \\ 3.4129 & 4.9317 & 8.7643 & 1.3142 \\ 1.2371 & 4.9998 & 10.6721 & 0.0147 \end{bmatrix} \begin{Bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 9.5342 \\ 6.3941 \\ 18.4231 \\ 16.9237 \end{Bmatrix}$$

Exact solution: $\begin{Bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{Bmatrix}$

PROBLEM 3

Solve system of equation by using

a) By using Gauss elimination
b) By using Gauss-Jordan elimination

$$\begin{bmatrix} 4 & 3 & -1 \\ 7 & -2 & 3 \\ 5 & -18 & 13 \end{bmatrix} \begin{Bmatrix} x_0 \\ x_1 \\ x_2 \end{Bmatrix} = \begin{Bmatrix} 6 \\ 9 \\ 3 \end{Bmatrix}$$

PROBLEM 4

Solve system of equation by using

a) By using Gauss elimination
b) By using Gauss-Jordan elimination
c) LU decomposition (Dolittle)
d) LU Craut Decomposition

$$\begin{bmatrix} 1 & 1/2 & 1/3 & 1/4 & 1/5 \\ 1/2 & 1/3 & 1/4 & 1/5 & 1/6 \\ 1/3 & 1/4 & 1/5 & 1/6 & 1/7 \\ 1/4 & 1/4 & 1/6 & 1/7 & 1/8 \\ 1/5 & 1/6 & 1/7 & 1/8 & 1/9 \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix} = \begin{bmatrix} 137/60 \\ 87/60 \\ 459/60 \\ 743/840 \\ 1879/2520 \end{bmatrix}$$

PROBLEM 5

Solve system of equation by using

a) By using Gauss elimination
b) Gauss-Seidel iteration
c) Conjugate gradient method

$$\begin{bmatrix} 8 & 2 & 3 \\ 3 & -9 & 2 \\ 2 & 3 & 6 \end{bmatrix} \begin{Bmatrix} x_0 \\ x_1 \\ x_2 \end{Bmatrix} = \begin{Bmatrix} 30 \\ 1 \\ 31 \end{Bmatrix}$$

PROBLEM 6 Solve system of equation by using

a) By using Gauss elimination
b) Gauss-Seidel with relaxation
c) Conjugate gradient method

$$\begin{bmatrix} 3 & -5 & 47 & 20 \\ 11 & 16 & 17 & 10 \\ 56 & 22 & 11 & -18 \\ 17 & 66 & -12 & 7 \end{bmatrix} \begin{Bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 18 \\ 26 \\ 34 \\ 82 \end{Bmatrix}$$

PROBLEM 7 Solve system of equation by using Thomas algorithm (3 band matrix method)

$$\begin{bmatrix} -2 & 1 & & & & & & & \\ 1 & -2 & 1 & & & & & & \\ & 1 & -2 & 1 & & & & & \\ & & & \dots & & & & & \\ & & & 1 & -2 & 1 & & & \\ & & & & 1 & -2 & 1 & & \\ & & & & & 1 & -2 & 1 & \\ & & & & & & 1 & -2 & \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ \\ X_7 \\ X_8 \\ X_9 \end{bmatrix} = \begin{bmatrix} -0.5 \\ -1.5 \\ -1.5 \\ \dots \\ -1.5 \\ -1.5 \\ 0.5 \end{bmatrix}$$

PROBLEM 8 Solve system of equation by using Thomass algorithm (3 band matrix method)

$$\begin{bmatrix} -4 & 1 & & & & & & & \\ 1 & -4 & 1 & & & & & & \\ & 1 & -4 & 1 & & & & & \\ & & & \dots & & & & & \\ & & & 1 & -4 & 1 & & & \\ & & & & 1 & -4 & 1 & & \\ & & & & & 1 & -4 & 1 & \\ & & & & & & 1 & -4 & \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ \\ X_7 \\ X_8 \\ X_9 \end{bmatrix} = \begin{bmatrix} -27 \\ -15 \\ -15 \\ \dots \\ -15 \\ -15 \\ -15 \end{bmatrix}$$

PROBLEM 9

Solve the following system of equation

$$\begin{bmatrix} 3 & -5 & 6 & 4 & -2 & -3 & 8 \\ 1 & 1 & -9 & 15 & 1 & -9 & 2 \\ 2 & -1 & 7 & 5 & -1 & 6 & 11 \\ -1 & 1 & 3 & 2 & 7 & -1 & -2 \\ 4 & 3 & 1 & -7 & 2 & 1 & 1 \\ 2 & 9 & -8 & 11 & -1 & -4 & -1 \\ 7 & 2 & -1 & 2 & 7 & -1 & 9 \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \\ X_6 \end{bmatrix} = \begin{bmatrix} 47 \\ 17 \\ 24 \\ 8 \\ 13 \\ -10 \\ 34 \end{bmatrix}$$

PROBLEM 10

Solve the following system of equation

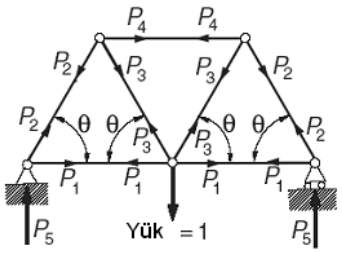
$$\begin{bmatrix} 1 & -1 & 2 & 5 & -7 & -8 \\ 3 & -9 & 1 & -1 & 8 & 1 \\ -1 & 1 & 9 & -9 & 2 & 3 \\ 1 & 7 & 2 & -3 & -1 & 4 \\ 7 & 1 & -2 & 4 & 1 & -1 \\ 2 & 3 & -9 & 12 & -2 & 7 \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \end{bmatrix} = \begin{bmatrix} -12 \\ 8 \\ 22 \\ 41 \\ 15 \\ 50 \end{bmatrix}$$

PROBLEM 11

Solve the following system of equation

$$\begin{bmatrix} 10 & -2 & -1 & 2 & 3 & 1 & -4 & 7 \\ 5 & 11 & 3 & 10 & -3 & 3 & 3 & -4 \\ 7 & 12 & 1 & 5 & 3 & -12 & 2 & 3 \\ 8 & 7 & -2 & 1 & 3 & 2 & 2 & 4 \\ 2 & -15 & -1 & 1 & 4 & -1 & 8 & 3 \\ 4 & 2 & 9 & 1 & 12 & -1 & 4 & 1 \\ -1 & 4 & -7 & -1 & 1 & 1 & -1 & -3 \\ -1 & 3 & 4 & 1 & 3 & -4 & 7 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \end{bmatrix} = \begin{bmatrix} 0 \\ 12 \\ -5 \\ 3 \\ -25 \\ -26 \\ 9 \\ -7 \end{bmatrix}$$

PROBLEM 12

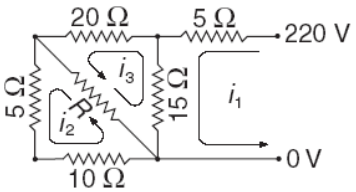


Load Matrix of the figure can be given with the following system of eqautions

$$\begin{bmatrix} c & 1 & 0 & 0 & 0 \\ 0 & s & 0 & 0 & 1 \\ 0 & 0 & 2s & 0 & 0 \\ 0 & -c & c & 1 & 0 \\ 0 & s & s & 0 & 0 \end{bmatrix} \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \\ P_5 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

Where c=cos(θ),s=sin(θ) . For θ =60º find the load distribution profile

PROBLEM 13



For the given electrical circuit current density-Voltage eqautions can be given as:

$$5i_1 + 15(i_1 - i_3) = 220\ V$$
$$5i_1 + 10i_2 + R(i_2 - i_3) = 0$$
$$20i_3 + R(i_3 - i_2) + 15(i_3 - i_1) = 0$$

Calculate current densities for R=10 Ω.

PROBLEM 14

$-x_1 + 3x_2 + 5x_3 + x_4 = 8$

$x_1 + 9x_2 + 3x_3 + 4x_4 = 10$

$x_2 + x_4 = 2$

$3x_1 + x_2 + x_3 - x_4 = -4$

Solve the given system of equations. Determine the best method to solve it.

PROBLEM 15 Solve the following system of equation by using Doolittle LU method

$$\begin{bmatrix} 1 & 3 & 5 & 7 \\ 11 & 16 & 17 & 10 \\ 0 & 0 & 2 & 5 \\ -2 & -6 & -3 & 1 \end{bmatrix} \begin{Bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{Bmatrix}$$

PROBLEM 16 Solve system of equation by using

a) By using Gauss elimination
b) By using Gauss-Jordan elimination
c) LU decomposition (Dolittle)
d) LU Craut Decomposition

$3x_1 + 5x_2 + 4x_3 + 2x_4 = 11$
$6x_1 + 14x_2 + 11x_3 + 6x_4 = 26$
$9x_1 + 11x_2 + 16x_3 + 5x_4 = 68$
$3x_1 + 13x_2 + 17x_3 + 12x_4 = 43$

PROBLEM 17 Solve system of equation by using

a) By using Gauss elimination
b) By using Gauss-Jordan elimination
c) LU decomposition (Dolittle)
d) LU Craut Decomposition
e) Gauss Seidel iteration
f) Conjugate-Gradian method

$$x_1 + 3x_2 + x_3 + 5x_4 = 4$$
$$2x_1 + x_2 + 3x_4 = 5$$
$$4x_1 + 2x_2 + 2x_3 + x_4 = 11$$
$$-3x_1 + x_2 + 3x_3 + 2x_4 = 3$$

PROBLEM 18 Solve system of equation by using

a) By using Gauss elimination
b) By using Gauss-Jordan elimination
c) LU decomposition (Dolittle)
d) LU Craut Decomposition

$x_1 + x_2 - x_3 = -3$
$6x_1 + 2x_2 + 2x_3 = 2$
$-3x_1 + 4x_2 + x_3 = 1$

PROBLEM 19 Find infinite norm two norm and m=3 norm of the matrix

$$\begin{bmatrix} 7 & 1 & 2 \\ -1 & 4 & -1 \\ 3 & 15 & 20 \end{bmatrix}$$

PROBLEM 20 Find the inverse matrix

$$\begin{bmatrix} 1.1348 & 3.8326 & 1.1651 & 3.4017 \\ 0.5301 & 1.7875 & 2.5330 & 1.5435 \\ 3.4129 & 4.9317 & 8.7643 & 1.3142 \\ 1.2371 & 4.9998 & 10.6721 & 0.0147 \end{bmatrix}$$

PROBLEM 21 Find the inverse matrix

$$\begin{bmatrix} 4 & 3 & -1 \\ 7 & -2 & 3 \\ 5 & -18 & 13 \end{bmatrix}$$

PROBLEM 22

Find two norm and inverse matrix

$$\begin{bmatrix} 1 & 1/2 & 1/3 & 1/4 & 1/5 \\ 1/2 & 1/3 & 1/4 & 1/5 & 1/6 \\ 1/3 & 1/4 & 1/5 & 1/6 & 1/7 \\ 1/4 & 1/4 & 1/6 & 1/7 & 1/8 \\ 1/5 & 1/6 & 1/7 & 1/8 & 1/9 \end{bmatrix}$$

PROBLEM 23

Find two norm, infinite norm and and inverse matrix

$$\begin{bmatrix} 8 & 2 & 3 \\ 1 & -9 & 2 \\ 2 & 3 & 6 \end{bmatrix}$$

PROBLEM 24

Find two norm, infinite norm and and inverse matrix

$$\begin{bmatrix} 3 & -5 & 47 & 20 \\ 11 & 16 & 17 & 10 \\ 56 & 22 & 11 & -18 \\ 17 & 66 & -12 & 7 \end{bmatrix}$$

PROBLEM 25

Find two norm, infinite norm and and inverse matrix

$$\begin{bmatrix} -2 & 1 & & & & & & \\ 1 & -2 & 1 & & & & & \\ & 1 & -2 & 1 & & & & \\ & & & & \dots & & & \\ & & & & 1 & -2 & 1 & \\ & & & & & 1 & -2 & 1 \\ & & & & & & 1 & -2 \end{bmatrix}$$

7. ROOT FINDING OF NON_LINEAR EQUATIONS

Root finding is one of the most used numerical methods. For the one independent variable functions, it requires to find x value where the function is zero f(x)=0. If multidimensional (has more than one independent value) functions are involved, it requires to find $x_1, x_2, x_3, \ldots, x_n$ roots of n equation $f_i(x_1, x_2, x_3, \ldots, x_n)=0$, i=1..n

Some of the root finding methods will be investigated. As a starting point, we will start a simpler form of a function y=f(x). In this equation you have one root value..

7.1 BISECTION METHOD

In the bisection method root of the functions is searched in a given region $a \leq x \leq b$ If a single root is existed in the given region equation $f(a)f(b) < 0$ (2.1.1) will be satisfied. If this condition is not satisfied, it can be assumed that there is no root in the given region. Of course the reason will be existance of double root in the given region. If the root existed region is divide to two equal parts as

$$p = \frac{a+b}{2} \quad (7.1.2)$$

and function is evaluated in the middle. If

$$f(a) * f(p) < 0 \quad (7.1.3)$$

root should be located in a-p else it is located in p-b region. Rarely it is also possible that

$$f(a) * f(p) = 0 \quad (7.1.4)$$

can happen, that will indicate the root is located at point p. The new root region is used in further iterations. In order to use iterative process an exceptible criteria for error limit can be assumed

$$\left|\frac{(b-a)}{(b+a)}\right| < \epsilon \qquad \text{or} \qquad |f(p)| < \epsilon \qquad (7.1.5)$$

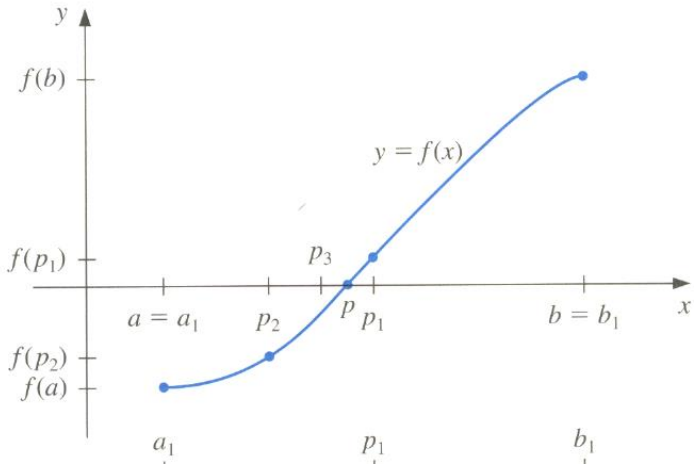


Figure 7.1 Convergence of bisection Method

An example code is created for the Bisection method.

Pyton definition of abstract if_x class

Program 7.1-1e Interface(abstract class) if_x in python language

```
 from abc import ABC, abstractmethod
from math import *

class if_x(ABC):

   @abstractmethod
   def func(self,x):
      pass;

   def dfunc(self,x,n):
      dx=0.0001;
      if n==0: df=self.func(x);
      elif n==1: df=(3.0*self.func(x-4.0*dx)-32.0*self.func(x-3.0*dx)+168.0*self.func(x-2.0*dx)-672.0*self.func(x-
dx)+672.0*self.func(x+dx)-168.0*self.func(x+2.0*dx)+32.0*self.func(x+3.0*dx)-3.0*self.func(x+4.0*dx))/840.0/dx;
      elif n==2: dx=0.001;df=(-14350.0*self.func(x)-9.0*self.func(x-4.0*dx)+128.0*self.func(x-3.0*dx)-1008.0*self.func(x-
2.0*dx)+8064.0*self.func(x-dx)+8064.0*self.func(x+dx)-1008.0*self.func(x+2.0*dx)+128.0*self.func(x+3.0*dx)-
9.0*self.func(x+4.0*dx))/5040.0/dx/dx;
      elif n==3: dx=0.001;df=(-7.0*self.func(x-4.0*dx)+72.0*self.func(x-3.0*dx)-338.0*self.func(x-2.0*dx)+488.0*self.func(x-dx)-
488.0*self.func(x+dx)+338.0*self.func(x+2.0*dx)-72.0*self.func(x+3.0*dx)+7.0*self.func(x+4.0*dx))/240.0/dx/dx/dx;
      elif n==4: dx=0.001;df=(2730.0*self.func(x)+7.0*self.func(x-4.0*dx)-96.0*self.func(x-3.0*dx)+676.0*self.func(x-2*dx)-
1952.0*self.func(x-dx)-1952.0*self.func(x+dx)+676.0*self.func(x+2.0*dx)-
96.0*self.func(x+3.0*dx)+7.0*self.func(x+4.0*dx))/240.0/dx/dx/dx/dx;
      elif n==5: dx=0.01;df=(self.func(x-4.0*dx)-9.0*self.func(x-3.0*dx)+26.0*self.func(x-2.0*dx)-29.0*self.func(x-
dx)+29.0*self.func(x+dx)-26.0*self.func(x+2.0*dx)+9.0*self.func(x+3.0*dx)-self.func(x+4.0*dx))/6.0/dx/dx/dx/dx/dx;
      elif n==6: dx=0.01;df=(-150.0*self.func(x)-self.func(x-4.0*dx)+12.0*self.func(x-3.0*dx)-52.0*self.func(x-
2.0*dx)+116.0*self.func(x-dx)+116.0*self.func(x+dx)-52.0*self.func(x+2.0*dx)+12.0*self.func(x+3.0*dx)-
self.func(x+4.0*dx))/4.0/dx/dx/dx/dx/dx/dx;
      elif n==7: dx=0.01;df=(-self.func(x-4.0*dx)+6.0*self.func(x-3.0*dx)-14.0*self.func(x-2.0*dx)+14.0*self.func(x-dx)-
14.0*self.func(x+dx)+14.0*self.func(x+2.0*dx)-6.0*self.func(x+3.0*dx)+self.func(x+4.0*dx))/2.0/dx/dx/dx/dx/dx/dx/dx;
      elif n==8: dx=0.1;df=(70.0*self.func(x)+self.func(x-4.0*dx)-8.0*self.func(x-3.0*dx)+28.0*self.func(x-2.0*dx)-56.0*self.func(x-
dx)-56.0*self.func(x+dx)+28.0*self.func(x+2.0*dx)-8.0*self.func(x+3.0*dx)+self.func(x+4.0*dx))/dx/dx/dx/dx/dx/dx/dx/dx;
      else: df=0;
      return df;

   def gauss_legendre_coefficients(self,x1,x2,n):
      #calculates legendre gauss-coefficients as coefficients of the integral
      #for n terms
      eps=3e-15
      m=int((n+1.0)/2.0)
      xm=0.5*(x2+x1)
      xl=0.5*(x2-x1)
      nmax=100
      a=[[0.0 for i in range(n)] for j in range(2)]
      for i in range(1,m+1):
         z=cos(pi*((i-0.25)/(n+0.5)))
         for ii in range(nmax):
            #while abs(z-z1) > eps:
            p1=1.0;
            p2=0.0;
            for j in range(1,n+1):
```

```
                p3=p2
                p2=p1
                p1=(float)(((2.0*j-1.0)*z*p2-(j-1.0)*p3)/j)
            pp=(float)(n*(z*p1-p2)/(z*z-1.0))
            z1=z;
            z=z1-p1/pp
            if abs(z-z1) < eps: break
        a[0][i-1]=xm-xl*z
        a[0][n-i]=xm+xl*z
        a[1][i-1]=2.0*xl/((1.0-z*z)*pp*pp)
        a[1][n-i]=a[1][i-1]
    return a

  def integral(self,x1,x2,n):
    #integral func(x)dx
    #integral of a function by using gauss-legendre quadrature
    #between x1 and x2
    a=self.gauss_legendre_coefficients(x1,x2,n)
    z=0
    for i in range(n):
        z=z+a[1][i]*self.func(a[0][i])
    return z
```

Program 7.1-2f Bisection method in python language class f1 is defined as seperate entity

```
 from math import *
from if_x import *;

class f1(if_x):
  def func(self,x):
     y=x*x-2;
     return y;

def bisection(f,xl,xu):
  # bisection root finding method
  maxit=100
  iter=0
  es=0.0000001
  ea=1.1*es
  s=""
  fxl= f.func(xl)
  fxu= f.func(xu)
  xr=xl
  while ea>es and iter<maxit:
    xold=xr
    xr=(xl+xu)/2.0;
    fxr= f.func(xr);
    iter=iter+1;
    if xr!=0:
      ea=abs((xr-xold)/xr)*100
    test= fxl*fxr;
    if test==0.0 :      ea=0;
    elif test<0.0: xu=xr;fxu=fxr;
    elif test>0:    xl=xr;fxl=fxr;
    else :          ea=0;
    if iter>=maxit: print("Maximum number of iteration is exceeded \n result might not be valid")
  return xr

f=f1();
a=float(input("enter lower limit : "))
b=float(input("enter upper limit : "))
r=bisection(f,a,b);
print(" ROOT  : ",r,"\nFUNCTION VALUE : ",f.func(r));
```

Pyton version of the bisection program: Note that function f1 is given as lambda definition but not as a separate class

Program 7.1-3g Bisection method in python language

```
from math import *

def bisection(f,xl,xu):
  # bisection root finding method
  maxit=100
  iter=0
```

```
  es=0.0000001
  ea=1.1*es
  s=""
  fxl= f(xl)
  fxu= f(xu)
  xr=xl
  while ea>es and iter<maxit:
    xold=xr
    xr=(xl+xu)/2.0;
    fxr= f(xr);
    iter=iter+1;
    if xr!=0:
      ea=abs((xr-xold)/xr)*100
    test= fxl*fxr;
    if test==0.0 :      ea=0;
    elif test<0.0: xu=xr;fxu=fxr;
    elif test>0:    xl=xr;fxl=fxr;
    else :          ea=0;
    if iter>=maxit: print("Maximum number of iteration is exceeded \n result might not be valid")
  return xr

f1 = lambda x: x*x-2;
a=float(input("enter lower limit : "))
b=float(input("enter upper limit : "))
r=bisection(f1,a,b);
print(" ROOT  : ",r,"\nFUNCTION VALUE : ",f1(r));x = np.arange(0, 3, 0.1);
```

```
runfile('D:/okul/SCO1/bisection.py', wdir='D:/okul/SCO1')
enter lower limit : 1
enter upper limit : 3
ROOT  :  1.4142135614529252
FUNCTION VALUE :  -2.6026334420947705e-09
```

Program 7.1-3h Bisection method in python language class f1 is defined internally with lambda variable

```
class f1(if_x):func=lambda self,x:x*x-2;
a=float(input("enter lower limit : "))
b=float(input("enter upper limit : "))
r=bisection(f,a,b);
print(" ROOT  : ",r,"\nFUNCTION VALUE : ",f.func(r));
```

Bisection method require a high number of iterations

```
# M. Turhan Ä‡OBAN
# 23.01.2022

# bisection root finding methods

# Hasan, Mohammed A., On the Derivation of Higher Order Root-finding Methods,
# Proceedings of the 2007 American Control Conference, ThA07.3, New York City,
# USA, July 11-13 2007

from math import *
from if_x import *

def root_bisection(f,a,b):
  # Hasan root finding method
  nmax=100
  tolerance=1.0e-20
  i=0
  fa= f.func(a)
  r=(a+b)/2.0
  fr=f.func(r)
  while abs(fr)>tolerance and i<nmax:
     if fa*fr<0:
        b=r
     else:
        a=r
        fa=fr
     r=(a+b)/2.0
     fr=f.func(r)
     print("i=",i,"fr=",fr)
     i=i+1
```

```
  return r


class f1(if_x):func=lambda self,x:x*x-exp(x)-3.0*x+2.0
f=f1()
a=0
b=3.6
r2=root_bisection(f,a,b)
print(" ROOT bisection : ",r2,"\nFUNCTION VALUE : ",f.func(r2))
```

```
runfile('D:/okul/SCO1/root_bisection1.py', wdir='D:/okul/SCO1')
Reloaded modules: if_x
i= 0 fr= -2.3496031111569504
i= 1 fr= -0.7158121854901691
i= 2 fr= 0.12330228380813546
i= 3 fr= -0.300033358391973
i= 4 fr= -0.0894331962288657
i= 5 fr= 0.016652993318758025
i= 6 fr= -0.036458691717268454
i= 7 fr= -0.009920223410241569
i= 8 fr= 0.003362013244826434
i= 9 fr= -0.0032801944982123743
i= 10 fr= 4.0636579967223696e-05
i= 11 fr= -0.0016198471025550276
i= 12 fr= -0.0007896223040164685
i= 13 fr= -0.00037449712356352194
i= 14 fr= -0.00016693133729006604
i= 15 fr= -6.314764504811166e-05
i= 16 fr= -1.1255599138948469e-05
i= 17 fr= 1.4690473764567002e-05
i= 18 fr= 1.7174331503611029e-06
i= 19 fr= -4.769084034794702e-06
i= 20 fr= -1.5258257022310318e-06
i= 21 fr= 9.580365878392172e-08
i= 22 fr= -7.150110379328112e-07
i= 23 fr= -3.096036937932922e-07
i= 24 fr= -1.0690001861490828e-07
i= 25 fr= -5.548180137537884e-09
i= 26 fr= 4.512773932319192e-08
i= 27 fr= 1.9789779592827017e-08
i= 28 fr= 7.120799949689172e-09
i= 29 fr= 7.86309906075644e-10
i= 30 fr= -2.3809354487980272e-09
i= 31 fr= -7.973124382942842e-10
i= 32 fr= -5.501377131622576e-12
i= 33 fr= 3.9040415344970825e-10
i= 34 fr= 1.9245138815904284e-10
i= 35 fr= 9.347500551371013e-11
i= 36 fr= 4.39870362356487e-11
i= 37 fr= 1.9243051596617988e-11
i= 38 fr= 6.870948254800169e-12
i= 39 fr= 6.845635169838715e-13
i= 40 fr= -2.4082957850168896e-12
i= 41 fr= -8.615330671091215e-13
i= 42 fr= -8.881784197001252e-14
i= 43 fr= 2.9820590441431705e-13
i= 44 fr= 1.0480505352461478e-13
i= 45 fr= 8.215650382226158e-15
i= 46 fr= -4.04121180963557e-14
i= 47 fr= -1.5987211554602254e-14
i= 48 fr= -3.9968028886505635e-15
i= 49 fr= 1.9984014443252818e-15
i= 50 fr= -8.881784197001252e-16
i= 51 fr= 6.661338147750939e-16
i= 52 fr= 0.0
 ROOT bisection :  0.2575302854398608
FUNCTION VALUE :  0.0
```

PROBLEM: Following equation is given:

$$f(x) = xtan(x) - Bi$$

For Bi=0.1 Find the multiple roots of the given equation

```
from math import *
from f_x import *;
```

```
import matplotlib.pyplot as plt


def bisection(f,a,b):
    b1=1.1*b
    r=(a+b)/2.0
    fr=f.func(r)
    fa=f.func(a)
    eps=1.0e-6;
    nmax=100;
    i=1;
    while abs(fr)>eps and i<nmax:
        if fa*fr<0:
            b=r
        else:
            a=r
            fa=fr
        r=(a+b)/2.0;
        fr=f.func(r)
        i=i+1
    if i>=nmax:
       r=bisection(f,a,b1)
    return r

def ksi(n,Bi):
    # eigenvalue function to be solved
    x1=n*pi+0.00000001
    x2=x1+(pi/2.0-0.00000001)
    class f1(f_x):func=lambda self,x: x*tan(x)-Bi
    f = f1()
    y= bisection(f,x1,x2)
    return y

Bi=0.1
for n in range(20):
    print("Bi = ",Bi," n = ",n," x =",ksi(n,Bi))
```

```
runfile('E:/okul/SCO1/bisection_test.py', wdir='E:/okul/SCO1')
Reloaded modules: f_x
Bi =  0.1  n =  0  x = 0.3110535614945277
Bi =  0.1  n =  1  x = 3.1730969436232437
Bi =  0.1  n =  2  x = 6.299059359233245
Bi =  0.1  n =  3  x = 9.435375957842044
Bi =  0.1  n =  4  x = 12.574323093800821
Bi =  0.1  n =  5  x = 15.714326807706442
Bi =  0.1  n =  6  x = 18.854859512786156
Bi =  0.1  n =  7  x = 21.995694913169565
Bi =  0.1  n =  8  x = 25.13671943960106
Bi =  0.1  n =  9  x = 28.27787017490028
Bi =  0.1  n =  10  x = 31.419109293857616
Bi =  0.1  n =  11  x = 34.56041264076988
Bi =  0.1  n =  12  x = 37.701764252275126
Bi =  0.1  n =  13  x = 40.843152893162554
Bi =  0.1  n =  14  x = 43.984570651971254
Bi =  0.1  n =  15  x = 47.12601177065568
Bi =  0.1  n =  16  x = 50.26747180194493
Bi =  0.1  n =  17  x = 53.408947468902504
Bi =  0.1  n =  18  x = 56.550436103165865
Bi =  0.1  n =  19  x = 59.691935691759696
```

PROBLEM: Following equation is given:

$$f(x) = xJ_1(x) - BiJ_0(x) = 0$$

Where $J_0(x)$ and $J_1(x)$ first and second order Bessel functions

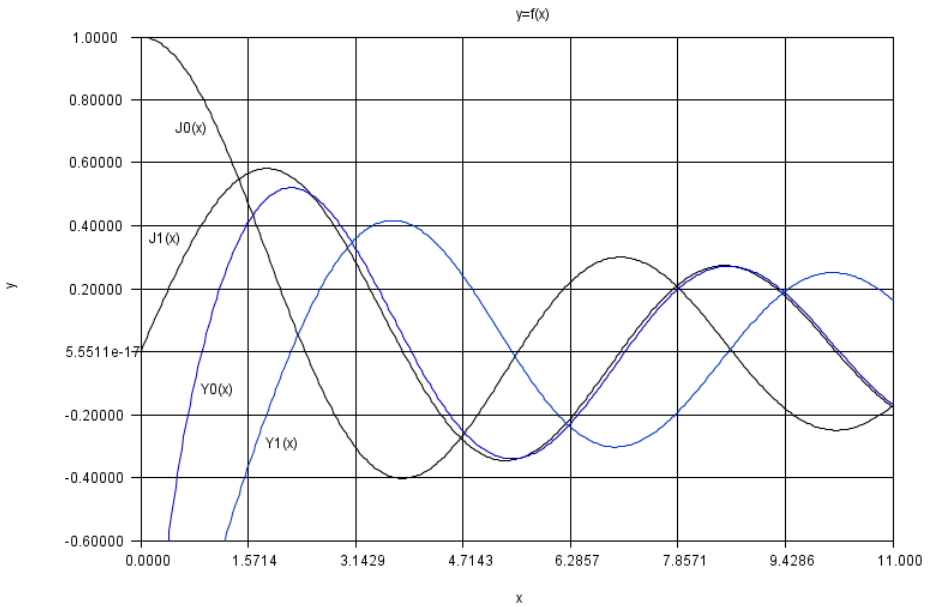
Bessel function $J_\nu(z)$ can be defined by using the following series:

$$J_\nu(z) = \left(\frac{z}{2}\right)^\nu \sum_{k=0}^{\infty} (-1)^k \frac{\left(\frac{z}{2}\right)^{2\kappa}}{k!\,\Gamma(\nu+k+1)}$$

Where gamma function defined as

$$\Gamma(z)=\int_{z=0}^{\infty} t^{z-1}e^{-t}dt=\frac{e^{-\gamma t}}{t}\prod_{n=1}^{\infty}\left(1+\frac{t}{n}\right)^{-1}e^{t/n}$$

Where γ=0.57721566490153286... is the Euler-Mascheroni constant. The Bessel function $Y_v(z)$ is given by:



For Bi=0.1 Find the multiple roots of the given equation

```
from math import *
from f_x import *;
import matplotlib.pyplot as plt

def J(v,x):
    if x==0.0: x=1.0e-10
    JJ=0.0
    a=1.0
    fact=1
    b=0.0
    x1=0.0
    if x <=20:
        x5=1.0
        lna=0.0;
        x1=log(0.5*x)
        x1=x1*v
        lna1=log(0.25*x*x)
        for k in range(0,100):
            x2=lna
            # lgamma=ln(gamma) is a math library function
            x3=lgamma(k+1)
            x4=lgamma(v+k+1)
            b=x1+x2-x3-x4
            b=x5*exp(b)
            lna=lna+lna1
            x5=-1*x5
            JJ=JJ+b
    elif (v==0 and x==0):
        JJ=1.0
    else:
        JJ=sqrt(2.0/pi/x)*cos(x-0.5*v*pi-0.25*pi)
    return JJ

def bisection(f,a,b):
    b1=1.1*b
    r=(a+b)/2.0
    fr=f.func(r)
    fa=f.func(a)
    eps=1.0e-6;
    nmax=100;
    i=1;
```

```
    while abs(fr)>eps and i<nmax:
        if fa*fr<0:
            b=r
        else:
            a=r
            fa=fr
        r=(a+b)/2.0;
        fr=f.func(r)
        i=i+1
    if i>=nmax:
       r=bisection(f,a,b1)
    return r

def enlarge(f,x0,dx):
    # enlarge region untill a root is existed
    x1=x0;
    x2=x1+dx
    NTRY=200
    a=[0.0 for i in range(n)]
    FACTOR=1.001
    j=0
    if x1 == x2: print("input variables are wrong")
    f1=f.func(x1)
    f2=f.func(x2)
    for j in range(NTRY+1):
        if f1*f2 < 0.0:
            break
        else:
            x2=x2+dx
            f2=f.func(x2)
    return x2


Bi=0.1
class f1(f_x):func=lambda self,xx: xx*J(1,xx)-Bi*J(0,xx)
f = f1()
x0=0.0
dx=0.001
for n in range(1,20):
    x1=enlarge(f,x0,dx)
    ksi=bisection(f,x0,x1)
    x0=ksi+dx
    print("Bi = ",Bi," n = ",n," x =",ksi)
```

```
runfile('E:/okul/SCO1/bisecion_test1.py', wdir='E:/okul/SCO1')
Reloaded modules: f_x
Bi =  0.1  n =  1  x = 0.44168142409737154
Bi =  0.1  n =  2  x = 3.85771028940425
Bi =  0.1  n =  3  x = 7.029825369592585
Bi =  0.1  n =  4  x = 10.183292774584878
Bi =  0.1  n =  5  x = 13.331195186460318
Bi =  0.1  n =  6  x = 16.476700564557774
Bi =  0.1  n =  7  x = 19.620955665797776
Bi =  0.1  n =  8  x = 22.780936100092504
Bi =  0.1  n =  9  x = 25.921997117309406
Bi =  0.1  n =  10  x = 29.0631727331714
Bi =  0.1  n =  11  x = 32.204429688792445
Bi =  0.1  n =  12  x = 35.34574648462637
Bi =  0.1  n =  13  x = 38.4871082430234
Bi =  0.1  n =  14  x = 41.628504916998224
Bi =  0.1  n =  15  x = 44.76992896512047
Bi =  0.1  n =  16  x = 47.91137518640969
Bi =  0.1  n =  17  x = 51.052839435342094
Bi =  0.1  n =  18  x = 54.19431840018331
Bi =  0.1  n =  19  x = 57.33580996009236
```

7.2 FALSE POSITION METHOD (REGULA FALSI)

In bisection method search region was devided by two in each time. It is caused a relatively long interpolation period. In order to improve interpolation time, two points can be joined with a line and root of the line can be taken as the next iteration point .

$$p = b - f(b)\frac{(a-b)}{f(a)-f(b)} = \frac{bf(a)-af(b)}{f(a)-f(b)} \quad (2.2.1)$$

Region selection process is same as bisection process. If f(a)*f(p) < 0 root should be located in a-p else it is located in p-b region. The new root region is used in further iterations. In order to use iterative process an exceptible criteria for error limit can be assumed

$$\left|\frac{(b-a)}{(b+a)}\right| < \epsilon \quad \text{or} \quad |f(p)| < \epsilon \qquad (2.2.2)$$

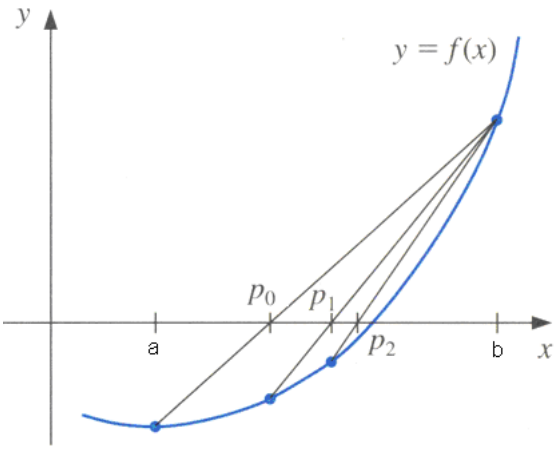


Figure 2.2 Convergence of the False position Method

Steps of bisection method is shown in detail in the following table(a spreadsheet solution) for function **f(x)=x^2-2**

Program 2.1-2b Bisection method in excel environment

xl	xu	xr	fxl	fxu	fxr	fxr*fxl
1	2	1.5	-1	2	0.25	-0.25
1	1.5	1.25	-1	0.25	-0.4375	0.4375
1.25	1.5	1.375	-0.4375	0.25	-0.10938	0.047852
1.375	1.5	1.4375	-0.10938	0.25	0.066406	-0.00726
1.375	1.4375	1.40625	-0.10938	0.066406	-0.02246	0.002457
1.40625	1.4375	1.421875	-0.02246	0.066406	0.021729	-0.00049
1.40625	1.421875	1.4140625	-0.02246	0.021729	-0.00043	9.6E-06
1.4140625	1.421875	1.41796875	-0.00043	0.021729	0.010635	-4.5E-06
1.4140625	1.41796875	1.416015625	-0.00043	0.010635	0.0051	-2.2E-06
1.4140625	1.416015625	1.415039063	-0.00043	0.0051	0.002336	-1E-06
1.4140625	1.415039063	1.414550781	-0.00043	0.002336	0.000954	-4.1E-07
1.4140625	1.414550781	1.414306641	-0.00043	0.000954	0.000263	-1.1E-07
1.4140625	1.414306641	1.41418457	-0.00043	0.000263	-8.2E-05	3.5E-08
1.41418457	1.414306641	1.414245605	-8.2E-05	0.000263	9.06E-05	-7.4E-09

As it is seen from the plot and table for iteration steps 2.0 is always stays and root search goes on between this two points. Even if solution is close to first point iterations did not cut from the left side of the search area. It may cause a long number of iteration steps.

```
from math import *

def false_position(f,xl,xu):
  # bisection root finding method
  maxit=100
  iter=0
  es=0.0000001
  ea=1.1*es
  s=""
  fxl= f(xl)
  fxu= f(xu)
  xr=xl
  while ea>es and iter<maxit:
    xold=xr
    xr=xu-fxu*(xl-xu)/(fxl-fxu);
    fxr= f(xr);
    iter=iter+1;
    if xr!=0:
      ea=abs((xr-xold)/xr)*100
    test= fxl*fxr;
```

```
    if test==0.0 :      ea=0;
    elif test<0.0: xu=xr;fxu=fxr;
    elif test>0:    xl=xr;fxl=fxr;
    else :          ea=0;
    if iter>=maxit: print("Maximum number of iteration is exceeded \n result might not be valid")
  return xr

f1 = lambda x: x*x-2;
a=float(input("enter lower limit : "))
b=float(input("enter upper limit : "))
r=false_position(f1,a,b);
print(" ROOT  : ",r,"\nFUNCTION VALUE : ",f1(r))
```

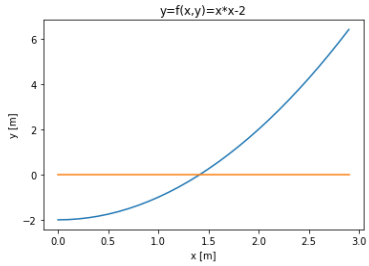
```
---------- Capture Output ----------
> "D:\co\anaconda\pythonw.exe" false_position.py
enter lower limit : 1
enter upper limit : 3
 ROOT  :  1.4142135617510516
FUNCTION VALUE :  -1.759404621992644e-09

> Terminated with exit code 0.
```

In order to overcome the difficulty of one sided cut of the region a bisection step can be put in with some regular intervals. For example once in every 4 false position steps. This is called **modified false position** method. The next example looks the root of $f(x)=x^2-2$ with modified false position method.

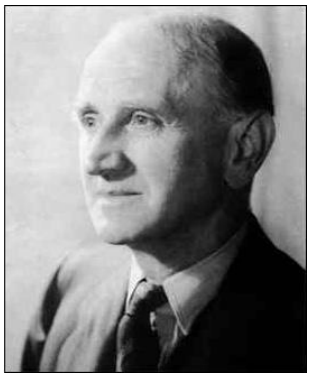
```
import numpy as np
import matplotlib.pyplot as plt
from math import *

def f1(x):
  return x*x-2.0

def false_position(xl,xu):
  # bisection root finding method
  maxit=100
  iter=0
  es=0.0000001
  ea=1.1*es
  s=""
  fxl= f1(xl)
  fxu= f1(xu)
  xr=xl
  while ea>es and iter<maxit:
    xold=xr
    if iter%4!=0:
      xr=xu-fxu*(xl-xu)/(fxl-fxu);
    else:
      xr=(xl+xu)/2.0;
    fxr= f1(xr);
    iter=iter+1;
    if xr!=0:
      ea=abs((xr-xold)/xr)*100
    test= fxl*fxr;
    if test==0.0 :      ea=0;
    elif test<0.0: xu=xr;fxu=fxr;
    elif test>0:    xl=xr;fxl=fxr;
    else :          ea=0;
    if iter>=maxit: print("Maximum number of iteration is exceeded \n result might not be valid")
  return xr

x = np.arange(0, 3, 0.1);
y = f1(x)
y1=0.0*x
plt.title('y=f(x,y)=x*x-2')
plt.xlabel('x [m]')
plt.ylabel('y [m]')
plt.plot(x, y,x,y1)
print(" Modified false position: ")
a=float(input("enter lower limit : "))
b=float(input("enter upper limit : "))
r=false_position(a,b);
print(" ROOT  : ",r,"\nFUNCTION VALUE : ",f1(r))
```

```
Modified false position:
```

```
enter upper limit : 3
 ROOT  :  1.414213562372946
FUNCTION VALUE :  -4.2166270475263445e-13
```



7.3 FIXED POSITION METHOD WITH AITKEN EXTRAPOLATION (STEPHENSEN METHOD)



Alexander Craig Aitken

Fixed position method is relatively simple. When a function f(x) is given to find the root, the function f(x) converted with algebraic manupulations and converted to f(x)=g(x) – x form. Since this equation can be written in as g(x)=x, starting from an initial guess, iteration continued till x and g(x) gives the same x value. Even though the relative simplicity, this method is not used much due to high failure possibility. The equation can diverge from the root sometimes. In order to overcome this problem aitken extrapolation process will be combined with fixed point , iteration. Method is called Stephensen method.

Aitken extrapolation process is a error reduction method. For a very big k number

$$\frac{x_{k+2}-a}{x_{k+1}-a} \approx \frac{x_{k+1}-a}{x_k-a} \quad k \gg 1 \qquad (7.3.1)$$

Relation can be valid. If the value of fourth point desired to be calculated from this equation

$$x = \frac{x_k x_{k+2} - x_{k+1}^2}{2x_k + 2x_{k+1} + 2x_{k+2}} \qquad (7.3.2)$$

When 3 points are previously known, a third one can be evaluated

$$x_k = x_{k+1} - x_k \qquad (7.3.3)$$

$$\Delta^2 x_k = x_{k+2} - 2x_{k+1+} x_k \quad (7.3.4)$$

$$x_e = x_{k+2} - \frac{(\Delta x_k)^2}{\Delta^2 x_k} \qquad (7.3.5)$$

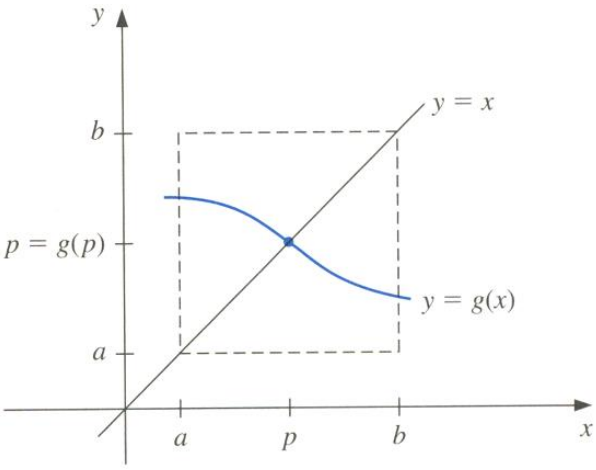


Figure 7.3.1 Convergence of the Fixed iteration Method

Stephensen Method is simple enough to calculate in excel format: The example problem is :

$$f(x) = x - \sqrt{\frac{5}{x+1}} = x - g(x) \qquad\qquad g(x) = \sqrt{\frac{5}{x+1}}$$

Program 7.3-1 : Fixed point iteration (Stephensen method) with excel:

Fixedİteration

f(x)=x-(5/(x+1)^0.5

Aitken		dxk	d2xk	xe=xk-(dxk)^2/d2xk	g(x)=(5/(x+1))^0.5
	x				
0	0.5				1.825741858
0	1.825741858	1.325741858			1.330205563
0	1.330205563	-0.495536296	-1.821278154	1.465031877	1.464832223
1	1.465031877	0.134826314	0.63036261	1.436194292	1.424209021
0	1.424209021	-0.040822856	-0.17564917	1.433696715	1.436150556
0	1.436150556	0.011941535	0.052764391	1.433447971	1.432626367
0	1.432626367	-0.003524189	-0.015465724	1.433429427	1.433663727
1	1.433429427	0.00080306	0.004327249	1.433280394	1.433427145
0	1.433427145	-2.28264E-06	-0.000805343	1.433427151	1.433427817
0	1.433427817	6.72303E-07	2.95494E-06	1.433427664	1.433427619
0	1.433427619	-1.98012E-07	-8.70315E-07	1.433427664	1.433427677
1	1.433427664	4.50514E-08	2.43064E-07	1.433427656	1.433427664
0	1.433427664	-7.32747E-15	-4.50514E-08	1.433427664	1.433427664
0	1.433427664	1.9984E-15	9.32587E-15	1.433427664	1.433427664
0	1.433427664	0	-2.44249E-15	1.433427664	1.433427664

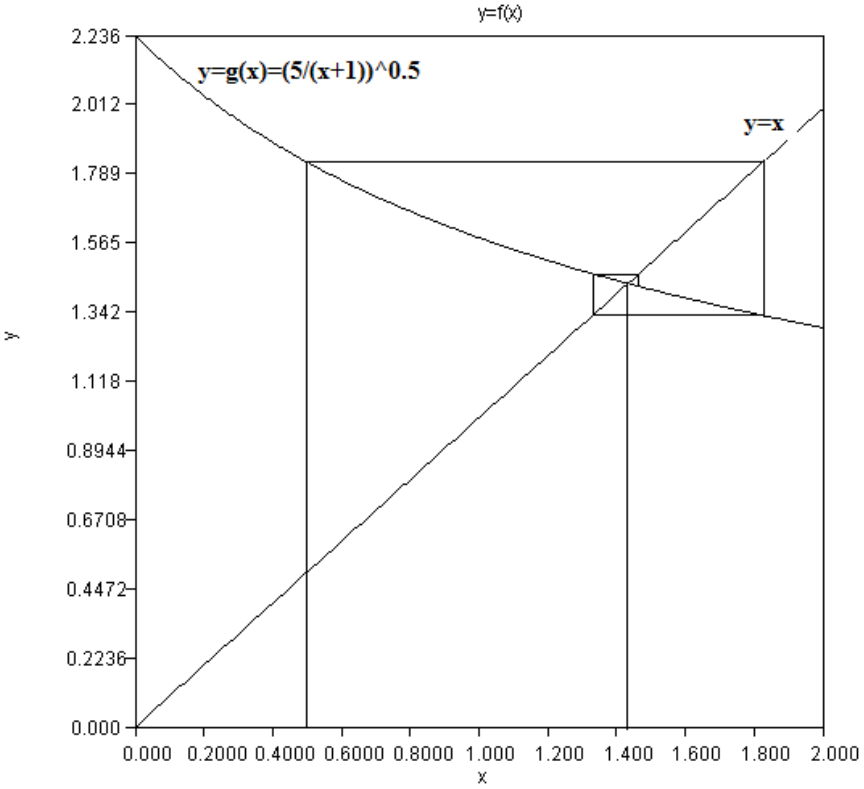
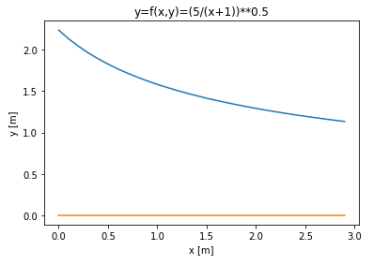


Figure 7.3.2 Convergence of the Fixed position Method

Program 7.3-2 : Fixed point iteration (Stephensen method) in Python code

```
 import numpy as np
import matplotlib.pyplot as plt
from math import *

def f1(x):
  return (5.0/(x+1.0))**0.5

def stephensen(x0):
  # Fixed point iteration with Aitken extrapolation
  # Stephensen Method
  maxit=100;
  iter=0;
  es=0.000001;
  ea=1.1*es;
  x=0;
  while iter<maxit and ea>es:
    x1=f1(x0);
    x2=f1(x1);
    x=x0-(x1-x0)*(x1-x0)/(x2-2*x1+x0);
    ea=abs(x-x0)*100.0;
    x0=x;
    iter=iter+1;
    if iter>=maxit:  print("Maximum number of iteration is exceeded  result might not be valid")
  return x

x = np.arange(0, 3, 0.1);
y = f1(x)
y1 = 0.0*x
plt.title('y=f(x,y)=(5/(x+1))**0.5')
plt.xlabel('x [m]')
plt.ylabel('y [m]')
plt.plot(x, y,x,y1)
print(" Stephensen method: ")
a = float(input("enter initial value : "))
rr=stephensen(a)
print(" ROOT  : ",rr,"\nFUNCTION VALUE : ",f1(rr))
```

```
In [21]: runfile('D:/okul/SC01/stephensen.py', wdir='D:/okul/SC01')
 Stephensen method:

enter initial value : 1
 ROOT  :  1.4334276638638197
FUNCTION VALUE :  1.4334276638638197
```




```
In [22]: |
```

7.4 NEWTON- RAPHSON METHOD

7.4.1 Isaac Newton

7.4.2 Joseph Raphson

Taylor Formula can be written as

$$f(x_{n+1}) = f(x_n) + \frac{f'(x_n)}{1!}(x_{n+1} - x_n) + \frac{f''(x_n)}{2!}(x_{n+1} - x_n)^2 + \frac{f^{(3)}(x_n)}{3!}(x_{n+1} - x_n)^3 + \cdots \quad (7.4.1)$$

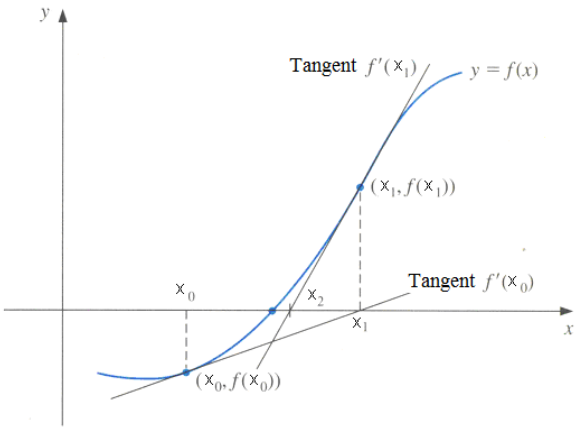
We would like to find $f(x_{n+1}) = 0$. If We substitude it into the Taylor Formula, and also cancel out terms above the first degree , The equation becomes

$$0 = f(x_n) + f'(x_n)(x_{n+1} - x_n) \quad (7.4.2)$$

If x_{n+1} taken to left hand side Newton-Raphson Formula is obtained

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \quad (7.4.3)$$

We can find the root iteratively starting from a first guess. Newton Formula aproached to the root quickly if the first guess is close to the actual root. But if the first gues is far away from the root, it might take longer or it might fail to reach to the root. Another difficulty in Newton-Raphson Formula is the requirement of knowing the actual derivative of the function



PROGRAM 7.4-1 Newton-Raphson method in python

```
# -*- coding: utf-8 -*-
"""
Created on Tue Apr 28 10:33:06 2020

@author: TURHAN
"""

# Newtons's method
def f(x):
    y=x*x-2
    return y
def df(x):
    y=2.0*x
    return y
def newton(f,df,x):
    eps=1e-10
    for i in range(100):
        if abs(f(x))<eps:return x
        else: x-=f(x)/df(x)
    return x
x=float(input("x="))
x=newton(f,df,x)
print("x=",x)
```

```
---------- Capture Output ----------
> "D:\co\python\pythonw.exe" NA5.py
enter initial guess : 1
fx -1.0 dfx 2.0 x= 1.5
fx 0.25 dfx 3.0 x= 1.4166666666666667
fx 0.006944444444444642 dfx 2.8333333333333335 x= 1.4142156862745099
fx 6.007304882871267e-06 dfx 2.8284313725490198 x= 1.4142135623746899
fx 4.510614104447086e-12 dfx 2.8284271247493797 x= 1.4142135623730951
 ROOT  :  1.4142135623730951
FUNCTION VALUE :  4.440892098500626e-16

> Terminated with exit code 0.
```

PROGRAM 7.4-2 Newton-Raphson method in excel

Newton-Raphson root finding $f(x)=x^2-a$

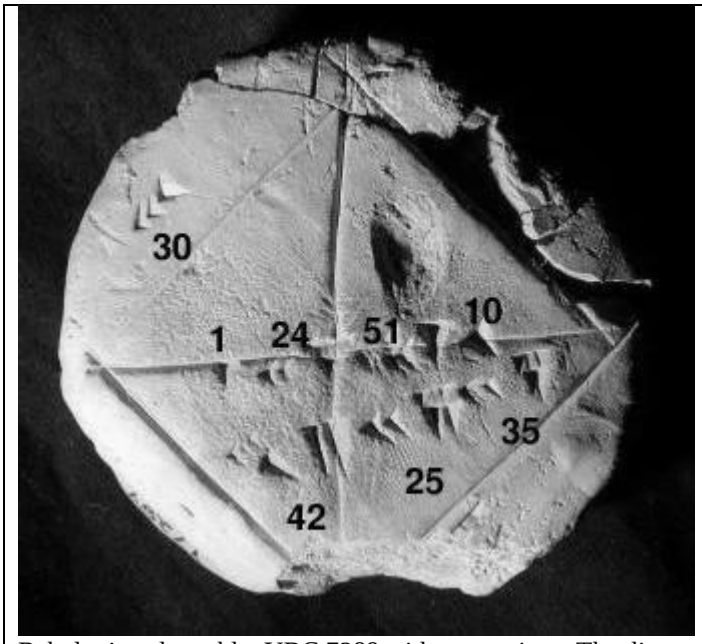
a	2	
x	f(x)=x*x-a	fx/dx=2*x
1.00000000000000000	-1	2
1.50000000000000000	0.25	3
1.41666666666667000	0.006944444	2.833333333
1.41421568627451000	6.0073E-06	2.828431373
1.41421356237469000	4.51061E-12	2.828427125
1.41421356237310000	0	2.828427125
1.41421356237310000	0	2.828427125
1.41421356237310000	0	2.828427125

Another program for Python version of the Newton-Raphson method. Please refer to **Program 2.1-1e Abstract class f_x in python language.** In this program function defined as a class function which is derived from the abstract class f_x

```
 from math import *
from f_x import *;

class f1(f_x):
  def func(self,x):
      y=x*x-2;
      return y;

class df1(f_x):
  def func(self,x):
      y=2.0*x;
      return y;

def newton(f,df,x):
   # Newton-Raphson root finding method
  nmax=100
  tolerance=1.0e-10
```

```
  for i in range(0,nmax):
      fx= f.func(x);
      dfx=df.func(x);
      x=x-fx/dfx
      print("fx",fx,"dfx",dfx,"x=",x)
      if abs(fx)<tolerance: return x
  return x


f=f1();
df=df1();
a=float(input("enter initial guess : "))
r=newton(f,df,a);
print(" ROOT  : ",r,"\nFUNCTION VALUE : ",f.func(r));
```

As it is seen from the results, convergence is quite fast for this problem.



Babylonian clay tablet YBC 7289 with annotations. The diagonal displays an approximation of the square root of 2 in four sexagesimal figures, 1 24 51 10, which is good to about six decimal digits.
$1 + 24/60 + 51/60^2 + 10/60^3 = 1.41421296...$
Babylonian algorithm is actually newton-raphson method.

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$
$$f(x_n) = x^2 - 2$$
$$f'(x_n) = 2x$$
$$x = x - \frac{x^2 - 2}{2x} = \frac{x}{2} + \frac{1}{x}$$

Babylonian algorithm in java format:

```
 #Babylonian algorithm
def  sqrt(xi):
  x = xi
  y =x*x-2
  e = 1.0e-10;
  while abs(y) > e:
      x =x /2.0+ 1.0/x
      y =x*x-2.0
  return x
print(sqrt(2.0))
```

```
runfile('E:/okul/SCO1/sqrt.py', wdir='E:/okul/SCO1')
```

1.4142135623746899

7.5 SECANT METHOD

Secant method: Instead of derivative, we can us difference formulas in Newton's method

One of the difficulty of the Newton-Raphson method is the requirement to find the derivative. If difference equation (Numerical derivative) is used instead of derivative (tangent to the function), a secant to the function is drawn to the function, so the method is called secant method. If derivative is approximated with a first degree difference equation

$$f'(x_n) \cong \frac{f(x_n)-f(x_{n-1})}{(x_n-x_{n-1})} \quad (7.5.1)$$

Newton –Raphson Formula is converted to

$$x_{n+1} = x_n - \frac{f(x_n)(x_n-x_{n-1})}{f(x_n)-f(x_{n-1})} \quad (7.5.2)$$

Two first estimation is needed to start iteration by using this formula $f(x_0)$ and $f(x_{-1})$

If we would like to establish a one estimation secant Formula, the previous difference equation can be written for a small Δx as (central difference Formula)

$$f'(x_n) \cong \frac{f(x_n+\Delta x)-f(x_n-\Delta x)}{2\Delta x} \quad (7.5.3)$$

Second equation becomes:

$$x_{n+1} = x_n - \frac{2f(x_n)\Delta x}{f(x_n+\Delta x)-f(x_n-\Delta x)} \quad (7.5.4)$$

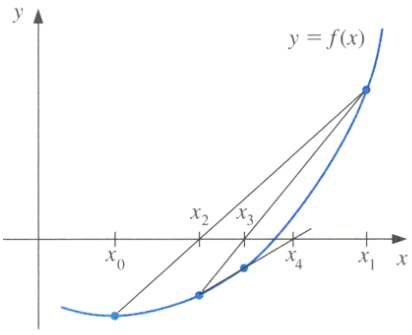
In order to minimize error Δx should be relatively small, to have a further approximation a second or higher derivative (difference) formulas can also be used

$$f'(x_n) \cong \frac{-f(x_n+2\Delta x)+8f(x_n+\Delta x)-8f(x_n-\Delta x)+f(x_n-2\Delta x)}{12\Delta x} \quad (7.5.5)$$

$$f'(x_n) \cong \frac{f(x_n+3\Delta x)-9f(x_n+2\Delta x)+45f(x_n+\Delta x)-45f(x_n-\Delta x)+9f(x_n-2\Delta x)-f(x_n-3\Delta x)}{60\Delta x} \quad (7.5.6)$$

$$f'(x_n) \cong \frac{-3f(x_n+4\Delta x)+32f(x_n+3\Delta x)-168f(x_n+2\Delta x)+672f(x_n+\Delta x)-672f(x_n-\Delta x)+168f(x_n-2\Delta x)-32f(x_n-3\Delta x)+3f(x_n-4\Delta x)}{840\Delta x}$$

(7.5.7)




```
# -*- coding: utf-8 -*-
from math import *

#def df(f,x):
#  h=0.001
#   return (f(x+dx)-f(x-dx))/(2.0*h)
def df(f,x):
   h=0.001
   return (-f(x+2.0*h)+8.0*f(x+h)-8.0*f(x-h)+f(x-2.0*h))/(12.0*h);

def secant(f,x):
   # secant root finding method
  nmax=100
  tolerance=1.0e-10
  for i in range(0,nmax):
      fx= f(x);
      dfx=df(f,x)
      x=x-fx/dfx
```

```
    print("fx",fx,"dfx",dfx,"x=",x)
    if abs(fx)<tolerance: return x
  return x

f1 = lambda x: x*x+2
a=complex(1,1);
r=secant(f1,a)
print(" ROOT  : ",r,"\nFUNCTION VALUE : ",f1(r))
```

```
runfile('E:/okul/SCO1/secant.py', wdir='E:/okul/SCO1')
fx (2+2j) dfx (2.0000000000000018+1.9999999999998908j) x= (-2.6867397195928788e-14+0.9999999999999722j)
fx (1.0000000000000555-5.373479439185609e-14j) dfx (-1.4802973661668753e-13+1.9999999999999445j) x= (3.700743415417599e-14+1.5000000000000138j)
fx (-0.2500000000000413+1.11022302462529e-13j) dfx (-3.700743415417188e-14+3.0000000000000275j) x= (-1.0279842820604793e-15+1.4166666666666674j)
fx (-0.006944444444446418-2.912622132504693e-15j) dfx 2.833333333333335j x= 1.4142156862745099j
fx (-6.007304882871267e-06+0j) dfx 2.8284313725490198j x= 1.4142135623746899j
fx (-4.510614104447086e-12+0j) dfx 2.82842712474938j x= 1.4142135623730951j
 ROOT  :  1.4142135623730951j
FUNCTION VALUE :  (-4.440892098500626e-16+0j)
```

```
f1 = lambda x: x*x-2
a=1;
r=secant(f1,a)
print(" ROOT  : ",r,"\nFUNCTION VALUE : ",f1(r))
```

```
runfile('E:/okul/SCO1/secant.py', wdir='E:/okul/SCO1')
fx -1 dfx 1.9999999999998537 x= 1.5000000000000366
fx 0.25000000000011013 dfx 2.9999999999995217 x= 1.4166666666666532
fx 0.00694444444440645 dfx 2.8333333333329103 x= 1.4142156862745094
fx 6.0073048815389996e-06 dfx 2.828431372548642 x= 1.4142135623746899
fx 4.510614104447086e-12 dfx 2.828427124748826 x= 1.4142135623730951
 ROOT  :  1.4142135623730951
FUNCTION VALUE :  4.440892098500626e-16
```

Secant method

Defined by using abstract class if_x

```
from math import *
from if_x import *

def secant(f,x):
  # secant root finding method
  nmax=100
  tolerance=1.0e-10
  for i in range(0,nmax):
      fx= f.func(x);
      dfx=f.dfunc(x,1)
      x=x-fx/dfx
      print("fx",fx,"dfx",dfx,"x=",x)
      if abs(fx)<tolerance: return x
  return x

class f1(if_x):func=lambda self,x:x*x-2.0
f=f1()
a=1;
r=secant(f,a)
print(" ROOT  : ",r,"\nFUNCTION VALUE : ",f.func(r))
```

```
runfile('E:/okul/SCO1/secant.py', wdir='E:/okul/SCO1')
Reloaded modules: if_x
fx -1.0 dfx 1.999999999998458 x= 1.5000000000003855
fx 0.2500000000011564 dfx 3.000000000002318 x= 1.4166666666667311
fx 0.0069444444446271625 dfx 2.8333333333347133 x= 1.414215686274511
fx 6.007304885979892e-06 dfx 2.828431372547751 x= 1.4142135623746899
fx 4.510614104447086e-12 dfx 2.828427124749537 x= 1.4142135623730951
 ROOT  :  1.4142135623730951
FUNCTION VALUE :  4.440892098500626e-16
```

Let us apply secant methos now to find the complex roots

```
from math import *
from if_x import *

def secant(f,x):
  # secant root finding method
```

```
  nmax=100
  tolerance=1.0e-10
  for i in range(0,nmax):
      fx= f.func(x);
      dfx=f.dfunc(x,1)
      x=x-fx/dfx
      print("fx",fx,"dfx",dfx,"x=",x)
      if abs(fx)<tolerance: return x
  return x

class f1(if_x):func=lambda self,x:x*x+2.0
f=f1()
a=complex(0,1)
r=secant(f,a)
print(" ROOT  : ",r,"\nFUNCTION VALUE : ",f.func(r))
```

```
runfile('E:/okul/SCO1/secant.py', wdir='E:/okul/SCO1')
fx (1+0j) dfx (3.0663302584885276e-13+2j) x= (-7.665825646221319e-14+1.5j)
fx (-0.25-2.299747693866396e-13j) dfx (9.833403932394243e-13+3.000000000000001j) x= (2.7315010923317305e-14+1.4166666666666667j)
fx (-0.006944444444444642+7.739253094939903e-14j) dfx 2.8333333333333335j x= 1.4142156862745099j
fx (-6.007304882871267e-06+0j) dfx 2.8284313725490198j x= 1.4142135623746899j
fx (-4.510614104447086e-12+0j) dfx 2.8284271247493797j x= 1.4142135623730951j
 ROOT  :  1.4142135623730951j
FUNCTION VALUE :  (-4.440892098500626e-16+0j)
```

As you will see not much changed except using complex variable as the first estimate.

Secant method id a first order method. Compare to bisection method, it will take less number of iterations

```
# M. Turhan Ã‡OBAN
# 23.01.2022

# secant root finding methods


from math import *
from if_x import *
def root_secant(f,x):
  # secant root finding method
  nmax=100
  tolerance=1.0e-20
  i=0
  fx= f.func(x)
  while abs(fx)>tolerance and i<nmax:
      dfx=f.dfunc(x,1)
      x=x-fx/dfx
      print("i=",i,"x=",x,"fx=",fx)
      fx= f.func(x)
      i=i+1
  return x


class f1(if_x):func=lambda self,x:x*x-exp(x)-3.0*x+2.0
f=f1()
x=3.6
r1=root_secant(f,x)
print(" ROOT  secant : ",r1,"\nFUNCTION VALUE : ",f.func(r1))
```

```
runfile('D:/okul/SCO1/root_secant.py', wdir='D:/okul/SCO1')
Reloaded modules: if_x
i= 0 x= 2.5987653648220843 fx= -32.438234443677985
i= 1 x= 1.4885173577304935 fx= -12.489840160792701
i= 2 x= 0.43756787438872435 fx= -4.680389917024559
i= 3 x= 0.2551481822234749 fx= -0.6701734065807567
i= 4 x= 0.2575297538349904 fx= 0.009003189707679438
i= 5 x= 0.2575302854398343 fx= 2.0087596988016543e-06
i= 6 x= 0.25753028543986073 fx= 9.992007221626409e-14
 ROOT  secant :  0.25753028543986073
FUNCTION VALUE :  0.0
```

PROBLEM For Turbulent flow inside of a pipe (Re>2300) friction coefficient inside of a pipe can be calculated by using Colebrook equation.

$$\frac{\mathbf{1}}{\sqrt{\boldsymbol{f}}} = -2.0log_{10}\left[\frac{\varepsilon/D}{3.7} + \frac{2.51}{Re\sqrt{f}}\right]$$

Re is Reynolds number, ε is surface rougness. Write a program to calculate total pressure drop in the pipe for a given Re number, pipe diameter, pipe length and local pressure drop coefficient.
f : friction coefficient
D pipe diameter
Hint : Change the colebrook equation into the form

$$F(x) = X + 2.0log_{10}\left[\frac{\varepsilon/D}{3.7} + \frac{2.51X}{Re}\right] = 0$$

Then find the root of the equation where $X = \frac{1}{\sqrt{f}}$

Use Haaland equation to obtaing the first estimate of the solution which has the form of

$$\mathbf{X}=\frac{\mathbf{1}}{\sqrt{f}} = -1.8log_{10}\left[\frac{(\varepsilon/D)^{1.11}}{3.7} + \frac{6.9}{Re}\right]$$

```
 from math import *
from if_x import *

class f1(if_x):
    def __init__(self,eod,Re):
        self.name = 'mathd',
        self.eod=eod
        self.Re=Re
    def func(self,X):
        F=X+2.0*log10(self.eod/3.7+2.51*X/self.Re);
        return F
    def Haaland(self):
        X=-1.8*log10(pow(self.eod,1.11)/3.7+6.9/self.Re);
        return X

def secant(f,x):
  # secant root finding method
  nmax=100
  tolerance=1.0e-10
  for i in range(0,nmax):
      fx= f.func(x)
      dfx=f.dfunc(x,1)
      x=x-fx/dfx
      if abs(fx)<tolerance: return x
  return x


f=f1(0.01,2500);
a=f.Haaland()
print(" f Haaland  : ",1.0/(a*a));
r=secant(f,a);
f1=1.0/(r*r);
print(" f colebrook  : ",f1);
```

```
runfile('E:/okul/SCO1/secant1.py', wdir='E:/okul/SCO1')
Reloaded modules: if_x
 f Haaland  :  0.05552457875442584
 f colebrook  :  0.053919466488887825
```

7.6 MULLER'S METHOD

Figure 7.5.1 Werner Muller, German Mathematician

In false position method roots were found by using the root of the linear line passing through the two interval points. The same concept can be use to fit a quadratic curve and find the roots of the quadratic function. But quadratic function fitting is required 3 estimation point. The method consist of deriving the coefficients of a

quadratic curve(parabola) that goes through three points. If function evaluation of these 3 points are $f(x_0)$, $f(x_1)$ and $f(x_2)$, enough information will be evailable to make a quadratic curve root estimation. Assuming quadratic function defined b the following equation :

$f(x) = a(x - x_2)^2 + b(x - x_2) + c$ (7.6.1)

Evaluation of the function in 3 given points will be

$f(x_0) = a(x_0 - x_2)^2 + b(x_0 - x_2) + c$ (7.6.2)

$f(x_1) = a(x_1 - x_2)^2 + b(x_1 - x_2) + c$ (7.6.3)

$f(x_2) = a(x_2 - x_2)^2 + b(x_2 - x_2) + c = c$ (7.6.4)

Considering the difference of the functions:

$f(x_0) - f(x_2) = a(x_0 - x_2)^2 + b(x_0 - x_2)$ (7.6.5)

$f(x_1) - f(x_2) = a(x_1 - x_2)^2 + b(x_1 - x_2)$ (2.6.6)

$h_0 = (x_1 - x_0)$ (7.6.7)

$h_1 = (x_2 - x_0)$ (7.6.8)

$d_0 = \frac{f(x_1) - f(x_0)}{h_0}$ (7.6.9)

$d_1 = \frac{f(x_2) - f(x_0)}{h_1}$ (7.6.10)

When these are substituted back into the main equation

$(h_0 + h_1)b - (h_0 + h_1)^2 a = h_0 d_0 + h_1 d_1$ (7.6.11)

$h_1 b - (h_1)^2 a = h_1 d_1$ (7.6.12)

is obtained. a and b can be solved from here.

$a = \frac{d_1 - d_0}{h_1 - h_0}$ (7.6.13)

$b = a h_1 + d_1$ (7.6.14)

$c = f(x_{2)}$ (7.6.15)

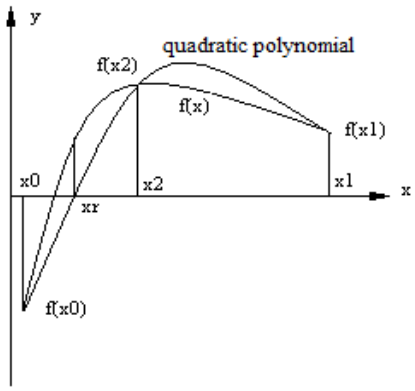


Figure 7.5.2 graphic representation of Muller root finding method.

to find root root :

$\Delta = \sqrt{b^2 - 4ac}$ (7.6.16)

if

$|b + \Delta| > |b + \Delta| \quad then\ e = b + \Delta$ (7.6.17)

Else

$e = b + \Delta$ (7.6.18)

$x_r = x_2 - \frac{2c}{e}$ (7.6.19)

$x_0 = x_1 \quad x_1 = x_2 \quad x_2 = x_r$ (2.6.20)

x_2 will be substituted with x_r and iteration continues. It should be noted that in a quadratic formula complex roots can be located as well as the real roots, therefore this method can be used to find complex roots as well.

Program 7.5-4 Muller method root finding with quadratic formula in Python code

```
# Muller method for root finding
# python version

from math import *
from f_x import *;

class f1(f_x):
```

```
  def func(self,x):
      y=x*x-2;
      return y;

def muller(f,x0,x1,x2):
  # Finding real roots by using Muller method
  iter=0;
  es1=0.001;
  es=es1;
  ea=1.1*es;
  maxit = 100;
  xr=x0;
  for iter in range(0,maxit):
    fx0=f.func(x0);
    fx1=f.func(x1);
    fx2=f.func(x2);
    h0=x1-x0;
    h1=x2-x1;
    d0=(fx1-fx0)/h0;
    d1=(fx2-fx1)/h1;
    a=(d1-d0)/(h1+h0);
    b=a*h1+d1;
    c=fx2;
    determinant=b*b-4.0*a*c;
    if determinant<=0:  determinant=0;
    else: determinant=sqrt(determinant);
    if abs(b+determinant)>  abs(b-determinant): den=b+determinant;
    else: den=b-determinant;
    dxr=-2*c/den;
    xr=x2+dxr;
    ea=abs((xr-x2)/xr)*100;
    x0=x1;
    x1=x2;
    x2=xr;
    if ea<es: return xr
  if iter>=maxit:  print('Warning : Maximum number of iteration is exceeded result may not be valis');
  return xr;


f=f1();
x0=float(input("enter first number : "));
x1=float(input("enter second number : "));
x2=float(input("enter third number : "));
r=muller(f,x0,x1,x2);
print("muller method root  : ",r);
```

```
---------- Capture Output ----------
> "D:\co\python\pythonw.exe" muller.py
enter first number : 0
enter second number : 1
enter third number : 3
muller method root  :  1.4142135623730951

> Terminated with exit code 0.
```

7.7 INVERSE QUADRATIC LAGRANGE INTERPOLATION

Lagrange interpolation will be investigated further in the interpolation section. In here, the Formula for the quadratic inverse lagrange polinomial will be introduced to utilise in finding roots of a function.

$$x = \frac{[y-f(x_1)][y-f(x_2)]x_3}{[f(x_3)-f(x_1)][f(x_3)-f(x_2)]} + \frac{[y-f(x_2)][y-f(x_3)]x_1}{[f(x_1)-f(x_2)][f(x_1)-f(x_3)]} + \frac{[y-f(x_3)][y-f(x_1)]x_2}{[f(x_2)-f(x_3)][f(x_2)-f(x_1)]} \quad (7.7.1)$$

In this equation if y=0 is taken, the value x will give us a quadratic approximation to the root of the function

$$x = \frac{f(x_1)f(x_2)x_3}{[f(x_3)-f(x_1)][f(x_3)-f(x_2)]} + \frac{f(x_2)f(x_3)x_1}{[f(x_1)-f(x_2)][f(x_1)-f(x_3)]} + \frac{f(x_3)f(x_1)x_2}{[f(x_2)-f(x_3)][f(x_2)-f(x_1)]} \quad (7.7.2)$$

In order to make the equation simpler further definations will apply as:

$R = \frac{f(x_2)}{f(x_3)}$ (2.8.4) $S = \frac{f(x_2)}{f(x_1)}$ (2.8.5) $T = \frac{f(x_1)}{f(x_3)}$ (7.7.3)

$$P = S[T(R-T)(x_3 - x_2) - (1-R)(x_2 - x_1)] \quad (7.7.4)$$

$$Q = (T-1)(R-1)(S-1) \quad (7.7.5)$$

Now the root can be written as:

$$x = x_2 + \frac{P}{Q} \quad (7.7.6)$$

When a new root is found the neigbour region of the root is left for the next iteration and the region further away to the root is eliminated.

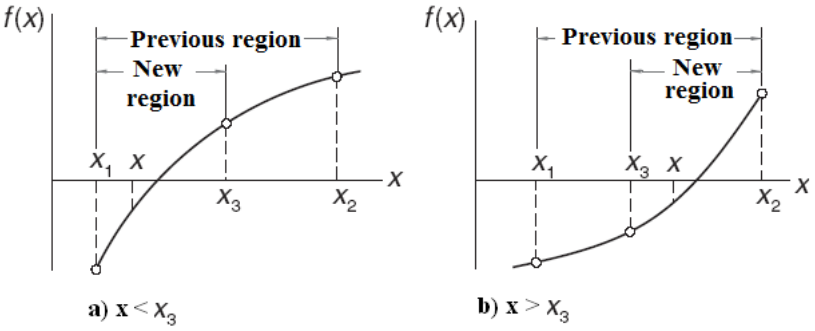


Figure 2.10 Quadratic inverse Lagrange interpolation region selection

Program 2.8-3 quadratic inverse lagrange interpolation formula method in Python language

```
 from math import *

from math import *
from f_x import *;

class f1(f_x):
  def func(self,x):
      y=x*x-2;
      return y;

def inverseinter2Droot(f,x1,x2,x3):
  #Inverse quadratic Lagrange erpolation
  p=1;
  maxit=100
  iter=0;
  tol=1.0e-10;
  es=0.00000001;
  ea=0.001;
  f1=f.func(x1);
  f2=f.func(x2);
  f3=f.func(x3);
  if f1==0: p=x1;
  elif f2==0: p=x2;
  elif f3==0: p=x3;
  if f1*f3>0: print("Root is not existed in the given region");
  while (ea>es) and (iter<maxit) and f1!=0 and f2!=0 and f3!=0:
    p=-(f2*f3*x1*(f2-f3)+f3*f1*x2*(f3-f1)+f1*f2*x3*(f1-f2))/((f1-f2)*(f2-f3)*(f3-f1));
    fp=f.func(p);
    ea=abs(x3-p);
    if abs(f3)<tol:  p=x3;
    if p>x3:  x1=x3;f1=f3;x3=p;f3=fp;
    elif p<x3: x2=x3;f2=f3;x3=p;f3=fp;
    iter=iter+1;
  if iter>=maxit:  print("Warning : Maximum number of iteration is exceeded");
  return p;

f=f1();
r=inverseinter2Droot(f,1.0,2.0,3.0);
print("r=",r);
```

```
---------- Capture Output ----------
```

```
> "D:\co\python\pythonw.exe" inverseinter2Droot.py
r= 1.4142135623730954
> Terminated with exit code 0.
```

7.8 BRENT METHOD (INVERSE QUADRATIC LAGRANGE INTERPOLATION – BISECTION AND SECANT COMBINED METHOD)

Figure 7.7.1 Richard P Brent

Being a quadratic method, Inverse quadratic Lagrange interpolation is an efficient method by combining it with bi-section and secant method one of the most used root finding method is created. Two versions of the program code and algorithm of the code is given below.

Program 7.8.1 brent.py : BRENT METHOD (FIRST VERSION) python code

```
from math import *

def brent(f,a,b):
    #brent root finding method
    maxit=500;
    iter=0;
    tol=1.0e-15;
    es=0.0000001;
    x1=a;f1=f(x1);
    x2=b;f2=f(x2);
    x3=(x1+x2)/2.0;f3=f(x3);
    if f1==0:    return x1;
    elif f2==0:  return x2;
    elif f3==0:  return x3;
    if f1*f2>0: print("No root is existed in the given region");
    p=-(f2*f3*x1*(f2-f3)+f3*f1*x2*(f3-f1)+f1*f2*x3*(f1-f2))/((f1-f2)*(f2-f3)*(f3-f1));
    fp=f(p);
    ea=abs(x3-p);
    while (ea>es) and (iter<maxit):
      if abs(f3)<tol: return x3;
      if (p<x1) and (p>x2):
        p=(x1+x2)/2.0;
        if p>x3:   x1=x3;f1=f3;x3=p;f3=fp;
        elif p<x3: x2=x3;f2=f3;x3=p;f3=fp;
      else:
        if p>x3: x1=x3;f1=f3;x3=p;f3=fp;
        elif p<x3: x2=x3;f2=f3;x3=p;f3=fp;
        p=-(f2*f3*x1*(f2-f3)+f3*f1*x2*(f3-f1)+f1*f2*x3*(f1-f2))/((f1-f2)*(f2-f3)*(f3-f1));
        fp=f(p);
        ea=abs(x3-p);
      iter=iter+1;
    if iter>=maxit: print("Warning : Maximum number of iteration is exceeded");
    return p;

f1 = lambda x: x*x-2
a=float(input("enter lower limit : "))
b=float(input("enter upper limit : "))
r=brent(f1,a,b);
print(" ROOT  : ",r,"\nFUNCTION VALUE : ",f1(r))
```

```
runfile('D:/okul/SCO1/brent.py', wdir='D:/okul/SCO1')
enter lower limit : 1
enter upper limit : 3
 ROOT  :  1.414213562373095
FUNCTION VALUE :  -4.440892098500626e-16
```

Algorithm
Input a,b and a function f
Calculate f(a) and f(b)
if(f(a)*f(b)<0 then give no root error
if |f(a)| < |f(b)| then swap(a,b)
c=a
repeat untill f(b)=0 or s=0 or |b-a| is small enough (convergence)
if(f(a)≠f(c) and f(b)≠f(c) then
$s = \frac{af(b)f(c)}{(f(a)-f(b))(f(a)-f(c))} + \frac{bf(a)f(c)}{(f(b)-f(a))(f(b)-f(c))} + \frac{cf(a)f(b)}{(f(c)-f(a))(f(c)-f(b))}$
(inverse quadratic interpolation)
else
$s = b - f(b)\frac{b-a}{(f(b)-f(a))}$ (secant rule)
end if
if *(condition 1) s* is not between (3*a* + *b*)/4 and *b*
or *(condition 2)* (mflag is set and |*s*−*b*| ≥ |*b*−*c*| / 2)
or *(condition 3)* (mflag is cleared and |*s*−*b*| ≥ |*c*−*d*| / 2)
or *(condition 4)* (mflag is set and |*b*−*c*| < |δ|)
or *(condition 5)* (mflag is cleared and |*c*−*d*| < |δ|)
then
$s := \frac{a+b}{2}$ *(bisection method)*
set mflag
else
clear mflag
end if
calculate *f*(*s*)
d := *c (d is assigned for the first time here; it won't be used above on the first iteration because mflag is set)*
c := *b*
if *f*(*a*) *f*(*s*) < 0 then *b* := *s* else *a* := *s* end if
if |*f*(*a*)| < |*f*(*b*)| then swap (*a*,*b*) end if
end repeat

- output *b or s (return the root*

7.9 HIGHER DERIVATIVE ORDER METHODS

Consider the Taylor Formula, but this time assume that two term is remaining

$$f(x_{n+1}) = 0 = f(x_n) + \frac{f'(x_n)}{1!}(x_{n+1} - x_n) + \frac{f''(x_n)}{2!}(x_{n+1} - x_n)^2 + \frac{f^{(3)}(x_n)}{3!}(x_{n+1} - x_n)^3 + \cdots \qquad (7.9.1)$$

When second term of the Taylor Formula is also taken into acount the Newton-Raphson Formula takes the form:

$$x_{n+1} = x_n - \frac{2f(x_n)f'(x_n)}{2f'(x_n)f'(x_n) - f(x_n)f''(x_n)} \qquad (7.9.2)$$

This method is called **Halley's method**.

Edmond Halley, British Mathematician and Astronomer

```
# M. Turhan COBAN
# 23.01.2022

# Halley root finding methods

from math import *
from if_x import *

def root_Halley(f,x):
   # alley root finding method
```

```
  nmax=100
  tolerance=1.0e-20
  i=0
  fx= f.func(x)
  alpha=0.32423
  beta=1.0-alpha
  while abs(fx)>tolerance and i<nmax:
     dfx=f.dfunc(x,1)
     d2fx=f.dfunc(x,2)
     f1=2.0*fx*dfx
     f2=2.0*dfx*dfx-fx*d2fx
     x=x-f1/f2
     print("i=",i,"x=",x,"fx=",fx)
     fx= f.func(x)
     i=i+1
  return x

class f1(if_x):func=lambda self,x:x*x-exp(x)-3.0*x+2.0
f=f1()
x=3.6
r5=root_Halley(f,x)
print(" ROOT  Halley : ",r5,"\nFUNCTION VALUE : ",f.func(r5))
```

```
runfile('D:/okul/SCO1/root_Halley1.py', wdir='D:/okul/SCO1')
Reloaded modules: if_x
i= 0 x= 1.4486037490878951 fx= -32.438234443677985
i= 1 x= 0.03827535612803068 fx= -4.5045247166036955
i= 2 x= 0.25788446666933107 fx= 0.8476216412313542
i= 3 x= 0.2575302854377124 fx= -0.0013382898462284132
i= 4 x= 0.25753028543986073 fx= 8.117950756059145e-12
 ROOT  Halley :  0.25753028543986073
FUNCTION VALUE :  0.0
```

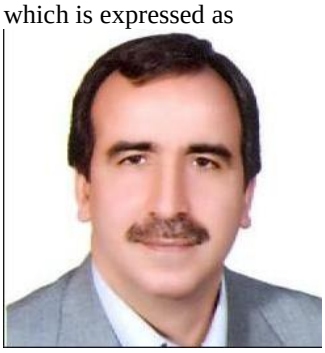
The iterative methods with higher-order convergence are presented in some literature . **Olver's method[97]** is also cubically convergent second order method. It is faster than the Newton-Raphson method.

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} - \frac{1}{2}\frac{f(x_n)f(x_n)f''(x_n)}{f'(x_n)f'(x_n)f'(x_n)} \qquad (7.9.3)$$

```
# M. Turhan Ã‡OBAN
# 25.01.2022

# Olver root finding method

from math import *
from if_x import *

def root_Olver(f,x):
   #Olver root finding method
   nmax=100
   tolerance=1.0e-20
   i=0
   fx= f.func(x)
   while abs(fx)>tolerance and i<nmax:
      dfx=f.dfunc(x,1)
      d2fx=f.dfunc(x,2)
      d3fx=f.dfunc(x,3)
      f1=0.5*fx*fx*d2fx
      f2=dfx*dfx*dfx
      if fx!=0:
         x=x-fx/dfx-f1/f2
      else:
         break
      print("i=",i,"x=",x,"fx=",fx)
      fx= f.func(x)
      i=i+1
   return x



class f1(if_x):func=lambda self,x:x*x-exp(x)-3.0*x+2.0
f=f1()
x=3.6
r4=root_Olver(f,x)
print(" ROOT  Olver : ",r4,"\nFUNCTION VALUE : ",f.func(r4))
```

```
runfile('D:/okul/SCO1/root_Olver1.py', wdir='D:/okul/SCO1')
i= 0 x= 2.0634936103802706 fx= -32.438234443677985
i= 1 x= 0.32370141086103177 fx= -7.805903460020479
i= 2 x= 0.2575168734077138 fx= -0.24855615446937662
i= 3 x= 0.2575302854398609 fx= 5.06797126789138e-05
i= 4 x= 0.2575302854398608 fx= -4.440892098500626e-16
 ROOT  Olver :  0.2575302854398608
FUNCTION VALUE :  0.0
```

In [48], Abbasbandy gives a third order iterative method, called **Abbasbandy's method**(AM) provisionally, which is expressed as



Daeid Abbasbandy, Iranian Mathematician

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} - \frac{1}{2}\frac{f(x_n)f(x_n)f''(x_n)}{f'(x_n)f'(x_n)f'(x_n)} - \frac{1}{6}\frac{f(x_n)f(x_n)f(x_n)f'''(x_n)}{f'(x_n)f'(x_n)f'(x_n)f'(x_n)} \qquad (7.9.4)$$

```
# M. Turhan Ã‡OBAN
# 25.01.2022

# Abbasbandy root finding methods

from math import *
from if_x import *

def root_Abbasbandy(f,x):
  # Abbasbandy root finding method
  nmax=100
  tolerance=1.0e-20
  i=0
  fx= f.func(x)
  while abs(fx)>tolerance and i<nmax:
     dfx=f.dfunc(x,1)
     d2fx=f.dfunc(x,2)
     d3fx=f.dfunc(x,3)
     f1=0.5*fx*fx*d2fx
     f2=dfx*dfx*dfx
     f3=0.166666667*fx*fx*fx*d3fx
     f4=dfx*dfx*dfx*dfx;
     if fx!=0:
        x=x-fx/dfx-f1/f2-f3/f4
     else:
        break
     print("i=",i,"x=",x,"fx=",fx)
     fx= f.func(x)
     i=i+1
  return x

class f1(if_x):func=lambda self,x:x*x-exp(x)-3.0*x+2.0
f=f1()
x=3.6
r3=root_Abbasbandy(f,x)
print(" ROOT  Abbasbandy : ",r3,"\nFUNCTION VALUE : ",f.func(r3))
```

```
runfile('D:/okul/SCO1/root_Abbasbandy1.py', wdir='D:/okul/SCO1')
Reloaded modules: if_x
i= 0 x= 1.8745227365029566 fx= -32.438234443677985
i= 1 x= -0.07673405242661357 fx= -6.627440433774902
i= 2 x= 0.25918682977164975 fx= 1.3099541475692424
i= 3 x= 0.25753028499909486 fx= -0.006258566997510151
i= 4 x= 0.2575302854398608 fx= 1.6655090639972059e-09
 ROOT  Abbasbandy :  0.2575302854398608
```

FUNCTION VALUE : 0.0

Li and Wang[49] suggested two fourth order methods and called them **Thiele Method 1** (TM1) and **Thiele Method 2** (TM2) due to fact that equations derived using Thiele's continued fraction method.

Thiele's interpolation is a formula that defines a rational function from a finite set of inputs. This interpolation formula is named after Thorvald N. Thiele.

Thorvald N. Thiele, Danish mathematician

It is expressed as a continued fraction

$$f(x) = f(x_1) + \cfrac{x - x_1}{\rho_1(x_1, x_2) + \cfrac{x - x_2}{\rho_2(x_1, x_2, x_3) - f(x_1) + \cfrac{x - x_3}{\rho_3(x_1, x_2, x_3, x_4) - \rho_1(x_1, x_2) + \cdots}}}$$

where ρ represents the reciprocal difference

$$\rho_1(x_1, x_2) = \frac{x_1 - x_2}{f(x_1) - f(x_2)}$$

$$\rho_2(x_1, x_2, x_3) = \frac{x_1 - x_3}{\rho_1(x_1, x_2) - \rho_1(x_1, x_3)} + f(x_1)$$

$$\rho_n(x_1, x_2, .., x_n) = \frac{x_1 - x_n}{\rho_{n-1}(x_1, x_2, .., x_{n-1}) - \rho_n(x_2, x_3, .., x_n)} + \rho_{n-2}(x_2, x_2, .., x_{n-1})$$

Equations has a general form of

$$x_{n+1} = x_n - \frac{P_1(x_n)}{Q_1(x_n)} \quad (7.9.5)$$

$$P_1(x) = 4f(x)f'(x)[3f'(x)f'(x)f"(x)\text{-}3f(x)f"(x)f"(x) + f(x)f'(x)f'''(x)] \quad (7.9.6)$$

$$Q_1(x) = 12f'(x)f'(x)f'(x)f'(x)f(x)\text{-}18f(x)f'(x)f'(x)f"(x)f"(x) + 3f(x)f(x)\, f"(x)f"(x)f"(x) + 4f(x)f'(x)f'(x)f'(x)f'''(x) \quad (7.9.7)$$

$$x_{n+1} = x_n - \frac{P_2(x_n)}{Q_2(x_n)} \quad (7.9.8)$$

$$P_2(x) = f(x)[6f'(x)f'(x)f''(x)\text{-}3f(x)f"(x)f"(x) + 2f(x)f'(x)f'''(x)] \quad (7.9.9)$$

$$Q_2(x) = 2f'(x)\left[3f'(x)f'(x)f"(x) - 3f(x)f"(x)f"(x) + f(x)f'(x)f'''(x)\right] \quad (7.9.10)$$

Program 2.11-7 Thiele1 method in python code

```
# -*- coding: utf-8 -*-
"""
Created on Tue Jan 25 06:22:21 2022

@author: M. Turhan COBAN

# Thiele 1 (Shenfeng Li & Rujing Wang) Root finding method
# Two fourth order Iterative Methods Based on Continued Fraction for
# Root-finding Problems, World Academy of Science, Engineering and Technology
# 60 2011&  root finding methods
"""
from math import *
from if_x import *

def root_Thiele1(f,x):
  # Abbasbandy root finding method
  nmax=100
  tolerance=1.0e-20
```

```
  i=0
  fx= f.func(x)
  while abs(fx)>tolerance and i<nmax:
     fx=f.func(x)
     dfx=f.dfunc(x,1)
     d2fx=f.dfunc(x,2)
     d3fx=f.dfunc(x,3)
     P1=4.0*fx*dfx*(3.0*dfx*dfx*d2fx-3.0*fx*d2fx*d2fx+fx*dfx*d3fx)
     Q1=12.0*dfx*dfx*dfx*dfx*d2fx-18.0*fx*dfx*dfx*d2fx*d2fx+3.0*fx*fx*d2fx*d2fx*d2fx+4.0*fx*dfx*dfx*dfx*d3fx
     if fx!=0:
        x-=P1/Q1
     else:
        break
     print("i=",i,"x=",x,"fx=",fx)
     fx= f.func(x)
     i=i+1
  return x

class f1(if_x):func=lambda self,x:x*x-exp(x)-3.0*x+2.0
f=f1()
x=3.6
r3=root_Thiele1(f,x)
print(" ROOT  Thiele 1 : ",r3,"\nFUNCTION VALUE : ",f.func(r3))
```

```
runfile('D:/okul/SCO1/root_Thiele1.py', wdir='D:/okul/SCO1')
Reloaded modules: if_x
i= 0 x= -4.516237397317507 fx= -32.438234443677985
i= 1 x= 0.16182552185254107 fx= 35.93418234896782
i= 2 x= 0.2575333614359719 fx= 0.365055836663706
i= 3 x= 0.2575302854398608 fx= -1.1623172178065744e-05
 ROOT  Thiele 1 :  0.2575302854398608
FUNCTION VALUE :  0.0
```

```
# -*- coding: utf-8 -*-
"""
Created on Tue Jan 25 06:22:21 2022

@author: M. Turhan COBAN

# Thiele 2 (Shenfeng Li & Rujing Wang) Root finding method
# Two fourth order Iterative Methods Based on Continued Fraction for
# Root-finding Problems, World Academy of Science, Engineering and Technology
# 60 2011&  root finding methods
"""
from math import *
from if_x import *

def root_Thiele2(f,x):
  # Abbasbandy root finding method
  nmax=100
  tolerance=1.0e-20
  i=0
  fx= f.func(x)
  while abs(fx)>tolerance and i<nmax:
     fx=f.func(x)
     dfx=f.dfunc(x,1)
     d2fx=f.dfunc(x,2)
     d3fx=f.dfunc(x,3)
     P2=fx*(6.0*dfx*dfx*d2fx-3.0*fx*d2fx*d2fx+2.0*fx*dfx*d3fx)
     Q2=2.0*dfx*(3.0*dfx*dfx*d2fx-3.0*fx*d2fx*d2fx+fx*dfx*d3fx)
     if fx!=0:
        x-=P2/Q2
     else:
        break
     print("i=",i,"x=",x,"fx=",fx)
     fx= f.func(x)
     i=i+1
  return x

class f1(if_x):func=lambda self,x:x*x-exp(x)-3.0*x+2.0
f=f1()
x=3.6
r3=root_Thiele2(f,x)
print(" ROOT  Thiele 2 : ",r3,"\nFUNCTION VALUE : ",f.func(r3))
```

```
runfile('D:/okul/SCO1/root_Thiele2.py', wdir='D:/okul/SCO1')
Reloaded modules: if_x
```

```
i= 0 x= 0.7127694878710749 fx= -32.438234443677985
i= 1 x= 0.2505859141492038 fx= -1.6699003011673375
i= 2 x= 0.25753028556180374 fx= 0.026257592132009844
i= 3 x= 0.25753028543986073 fx= -4.6078252324832647e-10
 ROOT  Thiele 2 :  0.25753028543986073
FUNCTION VALUE :  0.0
```

Shengfeng Li**[96]** Also suggested the following algorithm:

$$y_k = x_k - \frac{f(x_k)}{f'(x_k)}$$

$$x_{k+1} = x_k - \frac{(f(x_k) - f(y_k))f(x_k)}{(f(x_k) - 2f(y_k))f'(x_k)}$$

```
# M. Turhan COBAN
# 23.01.2022

# A fourth order root finding method by Shenfeng Li

# reference : Fourth-order iterative method without calculating
# the higher derivatives for nonlinear equations, Shenfeng Li
# Journal of Algorithms & computational Technology Vol 13:1-8
# DOI:10.1177/1748302619887686


from math import *
from if_x import *

def root_Shenfeng(f,x):
  # Shenfeng Li root finding method
  nmax=100
  tolerance=1.0e-20
  i=0
  fx= f.func(x)
  while abs(fx)>tolerance and i<nmax:
     dfx=f.dfunc(x,1)
     y=x-fx/dfx
     fy=f.func(y)
     f1=(fx-fy)*fx
     f2=(fx-2.0*fy)*dfx
     if f1!=0:
        x=x-f1/f2
     else:
        break
     print("i=",i,"x=",x,"fx=",fx)
     fx= f.func(x)
     i=i+1
  return x
```

```
runfile('D:/okul/SCO1/root_Shenfeng_Li1.py', wdir='D:/okul/SCO1')
Reloaded modules: if_x
i= 0 x= 0.9221320184499509 fx= -32.438234443677985
i= 1 x= 0.257443657345102 fx= -2.430714546291407
i= 2 x= 0.25753028543986073 fx= 0.00032734166948023
 ROOT  Shenfeng :  0.25753028543986073
FUNCTION VALUE :  0.0
```

A mixed Bisection-Shengeng Li Method

```
# M. Turhan COBAN
# 25.01.2022

# bisection root finding methods

# Hasan, Mohammed A., On the Derivation of Higher Order Root-finding Methods,
# Proceedings of the 2007 American Control Conference, ThA07.3, New York City,
# USA, July 11-13 2007

from math import *
from if_x import *

def root_bisection_Shenfeng_Li(f,a,b,eps):
  # bisection root finding method
```

```
  nmax=100
  tolerance=1.0e-20
  i=0
  fa= f.func(a)
  r=(a+b)/2.0
  fr=f.func(r)
  while abs(fr)>tolerance and i<nmax:
     if abs(fr)<eps:
        fr=f.func(r)
        dfr=f.dfunc(r,1)
        y=r-fr/dfr
        fy=f.func(y)
        f1=(fr-fy)*fr
        f2=(fr-2.0*fy)*dfr
        r=r-f1/f2
        print("Shenfeng Li i=",i,"fr=",fr)
     else:
        if fa*fr<0:
           b=r
        else:
           a=r
           fa=fr
        r=(a+b)/2.0
        print("bisecton i=",i,"fr=",fr)
     fr=f.func(r)

     i=i+1
  return r


class f1(if_x):func=lambda self,x:x*x-exp(x)-3.0*x+2.0
f=f1()
a=0
b=3.6
eps=0.5
r2=root_bisection_Shenfeng_Li(f,a,b,eps)
print(" ROOT bisection-Shenfeng Li : ",r2,"\nFUNCTION VALUE : ",f.func(r2),"eps = ",eps)
```

```
runfile('D:/okul/SCO1/root_bisection_Shenfeng_Li.py', wdir='D:/okul/SCO1')
Reloaded modules: if_x
bisecton i= 0 fr= -6.209647464412946
bisecton i= 1 fr= -2.3496031111569504
bisecton i= 2 fr= -0.7158121854901691
Shenfeng Li i= 3 fr= 0.12330228380813546
Shenfeng Li i= 4 fr= -1.910129654447701e-08
 ROOT bisection-Shenfeng Li :  0.2575302854398608
FUNCTION VALUE :  0.0 eps =  0.5
```

```
class f1(if_x):func=lambda self,x:x*x-exp(x)-3.0*x+2.0
f=f1()
a=0
b=3.6
eps=0.1
r2=root_bisection_Shenfeng_Li(f,a,b,eps)
print(" ROOT bisection-Shenfeng Li : ",r2,"\nFUNCTION VALUE : ",f.func(r2),"eps = ",eps)
```

```
runfile('D:/okul/SCO1/root_bisection_Shenfeng_Li.py', wdir='D:/okul/SCO1')
Reloaded modules: if_x
bisecton i= 0 fr= -6.209647464412946
bisecton i= 1 fr= -2.3496031111569504
bisecton i= 2 fr= -0.7158121854901691
bisecton i= 3 fr= 0.12330228380813546
bisecton i= 4 fr= -0.300033358391973
Shenfeng Li i= 5 fr= -0.0894331962288657
Shenfeng Li i= 6 fr= -5.392572610674051e-09
 ROOT bisection-Shenfeng Li :  0.25753028543986084
FUNCTION VALUE :  0.0 eps =  0.1
```

```
class f1(if_x):func=lambda self,x:x*x-exp(x)-3.0*x+2.0
f=f1()
a=0
b=3.6
eps=0.01
r2=root_bisection_Shenfeng_Li(f,a,b,eps)
print(" ROOT bisection-Shenfeng Li : ",r2,"\nFUNCTION VALUE : ",f.func(r2),"eps = ",eps)
```

```
runfile('D:/okul/SCO1/root_bisection_Shenfeng_Li.py', wdir='D:/okul/SCO1')
```

```
Reloaded modules: if_x
bisecton i= 0 fr= -6.209647464412946
bisecton i= 1 fr= -2.3496031111569504
bisecton i= 2 fr= -0.7158121854901691
bisecton i= 3 fr= 0.12330228380813546
bisecton i= 4 fr= -0.300033358391973
bisecton i= 5 fr= -0.0894331962288657
bisecton i= 6 fr= 0.016652993318758025
bisecton i= 7 fr= -0.036458691717268454
Shenfeng Li i= 8 fr= -0.009920223410241569
Shenfeng Li i= 9 fr= -8.113509863960644e-13
 ROOT bisection-Shenfeng Li :  0.2575302854398608
FUNCTION VALUE :  0.0 eps =  0.01
```

Hasan[50] proposed several higher order methods that is relatively more economical to use (relatively less function calculations) Some of them are listed here.

$$x_{n+1} = x_n - \frac{f(x_n)}{f'\left(x_n - \frac{f(x_n)}{2f'(x_n)}\right)} \qquad (7.9.11)$$

$$x_{n+1} = x_n - \frac{2f(x_n)}{f'(x_n) + f'\left(x_n - \frac{f(x_n)}{2f'(x_n)}\right)} \qquad (7.9.12)$$

$$x_{n+1} = x_n - \frac{2f(x_n)}{f'\left(x_n - \alpha\frac{f(x_n)}{f'(x_n)}\right) + f'\left(x_n - \beta\frac{f(x_n)}{f'(x_n)}\right)} \quad where\ \ \alpha + \beta = 1 \qquad (7.9.13)$$

```
#  M. Turhan COBAN
#  23.01.2022

# Hasan root finding methods

# Hasan, Mohammed A., On the Derivation of Higher Order Root-finding Methods,
# Proceedings of the 2007 American Control Conference, ThA07.3, New York City,
# USA, July 11-13 2007

from math import *
from if_x import *

def root_Hasan(f,x):
   # Hasan root finding method
  nmax=100
  tolerance=1.0e-20
  i=0
  fx= f.func(x)
  alpha=0.32423
  beta=1.0-alpha
  while abs(fx)>tolerance and i<nmax:
      dfx=f.dfunc(x,1)
      y1=x-alpha*fx/dfx
      y2=x-beta*fx/dfx
      dfy1=f.dfunc(y1,1)
      dfy2=f.dfunc(y2,1)
      x=x-2.0*fx/(dfy1+dfy2)
      print("i=",i,"x=",x,"fx=",fx)
      fx= f.func(x)
      i=i+1
  return x


class f1(if_x):func=lambda self,x:x*x-exp(x)-3.0*x+2.0
f=f1()
x=3.6
r2=root_Hasan(f,x)
print(" ROOT  Hasan : ",r2,"\nFUNCTION VALUE : ",f.func(r2))
```

```
runfile('D:/okul/SCO1/root_Hasan1.py', wdir='D:/okul/SCO1')
Reloaded modules: if_x
i= 0 x= 1.921863447355149 fx= -32.438234443677985
i= 1 x= 0.2885629148506894 fx= -6.90571204812608
i= 2 x= 0.2575302589901204 fx= -0.11692849599761512
i= 3 x= 0.2575302854398608 fx= 9.994485195008451e-08
 ROOT  Hasan :  0.2575302854398608
FUNCTION VALUE :  0.0
```

```
# M. Turhan Ã‡OBAN
# 23.01.2022

# Hasan root finding methods

# Hasan, Mohammed A., On the Derivation of Higher Order Root-finding Methods,
# Proceedings of the 2007 American Control Conference, ThA07.3, New York City,
# USA, July 11-13 2007

from math import *
from if_x import *

def root_Hasan(f,x):
  # Hasan root finding method
  nmax=100
  tolerance=1.0e-20
  i=0
  fx= f.func(x)
  r=3.0
  while abs(fx)>tolerance and i<nmax:
     dfx=f.dfunc(x,1)
     d2fx=f.dfunc(x,2)
     y1=x-fx/dfx
     dfy1=f.dfunc(y1,1)
     print("i=",i,"fx=",fx)
     if dfx!=0:
        x=x-2.0*fx/(dfx+dfy1)
     else:
        break
     fx= f.func(x)
     i=i+1
  return x


class f1(if_x):func=lambda self,x:x*x-exp(x)-3.0*x+2.0
f=f1()
x=3.6
r2=root_Hasan(f,x)
print(" ROOT  Hasan : ",r2,"\nFUNCTION VALUE : ",f.func(r2))
```

```
runfile('D:/okul/SCO1/root_Hasan2.py', wdir='D:/okul/SCO1')
Reloaded modules: if_x
i= 0 fx= -32.438234443677985
i= 1 fx= -8.151752380853384
i= 2 fx= -1.2468808618487799
i= 3 fx= -0.005780188046727197
i= 4 fx= -5.044245021679217e-10
 ROOT  Hasan :  0.25753028543986073
FUNCTION VALUE :  0.0
```

7.9 RIDDER METHOD

False position (Ragula Falsi) method was previously investigated. Ridder method is a modified form False position method. If a root existed between x1 and x2 Ridder method first evaluate the midpoint

$$x_3 = \frac{x_1+x_2}{2} \qquad (2.13.1)$$

and than solve a special conversion conversion factor e^Q. This convertion linearize the equation. It is basically find the root of the quadratic function

$$f(x_1) - 2\,f(x_3)e^Q + f(x_1) - 2\,f(x_2)e^Q = 0 \qquad (2.13.2)$$

The root of this function can be calculated as
of the quadratic function

$$e^Q = \frac{f(x_3)+sign[f(x_2)]\sqrt{f(x_3)^2-f(x_1)f(x_2)}}{f(x_2)} \qquad (2.13.3)$$

Function sign(x) will be given the sign of x. If x is smaller than 0 it will be -1, if x is bigger than 0 it will be +1. Now the false position method is applied, not to the values f(x1), f(x3),

f(x2), but to the values $f(x_1)$, $f(x_3)e^Q$, $f(x_2)e^{2Q}$, yielding a new guess for the root, x_4. The overall formula is:

$$x_4 = x_3 + (x_3 - x_1)\frac{f(x_3)+sign[f(x_1)-f(x_2)]f(x_3)}{\sqrt{f(x_3)^2-f(x_1)f(x_2)}} \qquad (2.13.4)$$

Because the equation is quadratic, it supplies a quadratic approximation. In addition the root x_4 should be in between x_1 and x_2.

Program 7.9.1 Root finding Ridder Method python version

```
from math import *

def SIGN(a,b):
    return abs(a)*b/abs(b);

def ridder(f,x1,x2):
    xacc=1.0e-9;
    MAXITER=200;ABC=-1.0e30;
    f_low=f(x1);
    f_high=f(x2);
    b1=f_low > 0.0 and f_high < 0.0;
    b2=f_low < 0.0 and f_high > 0.0;
    b3=b1 or b2;
    if b3 :
      x_low=x1;
      x_high=x2;
      answer=ABC;
      for j in range(0,MAXITER):
         x_mid=0.5*(x_low+x_high);
         f_mid=f(x_mid);
         s=sqrt(f_mid*f_mid-f_low*f_high);
         if s == 0.0: return answer;
         if f_low>=f_high: xx=1;
         else: xx=-1;
         xnew=x_mid+(x_mid-x_low)*xx*f_mid/s;
         if abs(xnew-answer) <= xacc: return answer;
         answer=xnew;
         fnew=f(answer);
         if fnew == 0.0: return answer;
         if SIGN(f_mid,fnew) != f_mid:x_low=x_mid;f_low=f_mid;x_high=answer;f_high=fnew;
         elif SIGN(f_low,fnew) != f_low:x_high=answer;f_high=fnew;
         elif SIGN(f_high,fnew) != f_high:x_low=answer;f_low=fnew;
         else: print("we should not reach to this point");
      if abs(x_high-x_low) <= xacc: return answer;
      print("Warning : Maximum number of iteration is exceeded \n");
    else:
       if f_low == 0.0: return x1;
       if f_high == 0.0: return x2;
       print("Warning : No roots existed between the given limits ");
    return 0.0;

f1 = lambda x: x*x-2
a=float(input("enter lower limit : "))
b=float(input("enter upper limit : "))
r=ridder(f1,a,b);
print(" ROOT  : ",r,"\nFUNCTION VALUE : ",f1(r))
```

```
runfile('D:/okul/SCO1/ridder2.py', wdir='D:/okul/SCO1')
Reloaded modules: Mathd

enter lower limit : 1

enter upper limit : 3
 ROOT  :  1.4142135621695027
FUNCTION VALUE :  -5.758462595650826e-10
```

7.10 ROOTS OF THE SECOND AND THIRD DEGREE POLYNOMIALS

The roots of quadratic polynomials are well known.

$$f(x) = ax^2 + bx + c = 0$$

$$\Delta = b^2 - 4ac$$

$x_0 = \frac{-b}{2a} + \frac{\sqrt{\Delta}}{2a}$
$x_1 = \frac{-b}{2a} - \frac{\sqrt{\Delta}}{2a}$
If $\Delta = b^2 - 4ac$ is negative, roots will be complex numbers.

Program 2.14-2 Root of quadratic polynomial (analytical formula) python version

```
from math import *

def square_roots(d):
   x= [0 for i in range(2)];
   a=d[2];
   b=d[1];
   c=d[0];
   delta=b*b-4*a*c;
   x[0]=(-b+sqrt(delta))/(2*a);
   x[1]=(-b-sqrt(delta))/(2*a);
   return x;

a=[1.0,-2.0,1.0]  # y(x)=x^2-2*x+1
print("Roots of the polynomial : ",square_roots(a))
```

```
---------- Capture Output ----------
> "D:\co\python\pythonw.exe" square_roots.py
Roots of the polynomial :  [1.0, 1.0]
```

Similar to quadratic polynomials, third degree polynomials can also be solved by using analytical formulations. The basic difference is that the roots should be calculated as complex variables. Assuming a third degree polynomial of

Figure NiccoloTartaglia

$f(x) = d_3x^3 + d_2x^2 + d_1x + d_0 = 0$

$a = \frac{d_2}{d_3} \quad b = \frac{d_1}{d_3} \quad c = \frac{d_0}{d_3}$

$f(x) = x^3 + ax^2 + bx + c = 0$

$Q = \frac{a^2-3b}{9}$ (2.14.4) $\quad y_1 = 2a^3$ (2.14.5) $y_2 = -9ab$ (2.14.6) $y_3 = 27c$

$y_4 = y_1 + y_2 + y_3$

$R = \frac{y_4}{54} = \frac{2a^3-9ab+27c}{54} \qquad \theta = cos^{-1}\left(\frac{R}{\sqrt{Q^3}}\right)$)

If $R^2 < Q^3$ three real roots are existed

$x_0 = \left(-2\sqrt{Q}\right)cos\left(\frac{\theta}{3}\right) - \frac{a}{3}$

$x_1 = \left(-2\sqrt{Q}\right)cos\left(\frac{\theta-2\pi}{3}\right) - \frac{a}{3}$

$x_2 = \left(-2\sqrt{Q}\right)cos\left(\frac{\theta+2\pi}{3}\right) - \frac{a}{3}$

(This equation first appears in Chapter VI of François Viete's treatise "De emendatione," published in 1615!, Original equation created by Niccolo Tartaglia in 1539 in poetry form)

If $R^2 \geq Q^3$

$A = (R + \sqrt{R^2 - Q^3})^{1/3}$ (2.14.13)

If A=0 $\Rightarrow$B=0 (2.14.14)

If A< >0 $\Longrightarrow B = \frac{Q}{A}$ (2.14.15)

$x_0 = (A+B) - \frac{a}{3}$ (2.14.16) real root

$x_1 = \left[-\frac{(A+B)}{2} - \frac{a}{3}\right] + \left[-\frac{\sqrt{3}(A-B)}{2}\right] i$ (2.14.17) complex root

$x_2 = \left[-\frac{(A+B)}{2} - \frac{a}{3}\right] - \left[-\frac{\sqrt{3}(A-B)}{2}\right] i$ (2.14.18) complex root

Program 2.14-4 Root of third degree polynomial (analytical formula) excel version

Root of third degree polynomial (analytical formula) Excel version

$f(x) = d_3x^3 + d_2x^2 + d_1x + d_0 = 0$ $f(x) = x^3 + ax^2 + bx + c = 0$

Enter poynomial coefficients

d3	d2	d1	d0
1	3	3	-1

	a	b	c
1	3	3	-1

Q	0
y1	54
y2	-81
y3	-27
y4	-54
R	-1
θ	#SAYI/0!

true/false
0

true if R^2<Q^3

x0	x1	x2
0	0	0

true if R^2>=Q^3

A	0	
B	0	

true/false
1

xo real	xo imaginary
-1	0

x1 real	x1 imaginary
-1	0

x2 real	x2 imaginary
-1	0

Python version of the program:

Program 2.14-5 Root of third degree polynomial (analytical formula) Python version

```
from math import *

def cubic_roots(d):
    x=[[0 for i in range(3)] for j in range(2)]
    a=d[2]/d[3]
    b=d[1]/d[3]
    c=d[0]/d[3]
    Q=(a*a-3.0*b)/9.0
    y1=2.0*a*a*a
    y2=-9*a*b
    y3=27*c
    y4=y1+y2+y3
    R=y4/54.0
    Q3=Q*Q*Q
    R2=R*R
    if R2<Q3:
        qq=-2.0*sqrt(Q)
        theta=acos(R/Math.sqrt(Q3))
        x[0][0]=qq*cos(theta/3.0)-a/3.0
        x[0][1]=qq*cos((theta-2.0*pi)/3.0)-a/3.0
        x[0][2]=qq*cos((theta+2.0*pi)/3.0)-a/3.0
    else:
        A=-pow((R+sqrt(R2-Q3)),(1.0/3.0))
        if A==0:  B=0
        else   :  B=Q/A
        x[0][0]=(A+B)-a/3.0
        x[0][1]=-0.5*(A+B)-a/3
        x[1][1]=sqrt(3.0)/2.0*(A-B)
        x[0][2]=-0.5*(A+B)-a/3
        x[1][2]=-sqrt(3.0)/2.0*(A-B)
```

```
    y=[complex(0,0) for i in range(3)]
    y[0]=complex(x[0][0],x[1][0])
    y[1]=complex(x[0][1],x[1][1])
    y[2]=complex(x[0][2],x[1][2])
    return y

a=[1.0,-3.0,3.0,-1.0]  # y(x)=x^3+3*x^2+3*x+1=(x+1)^3
print("Roots of the polynomial : ",cubic_roots(a))
```

```
runfile('E:/okul/SCO1/NA13D.py', wdir='E:/okul/SCO1')
Roots of the polynomial :  [(1+0j), (1-0j), (1+0j)]
```

Tartaglia's Poem

Italian:	**English Translation:**
Quando chel cubo con le cose appresso *Se agguaglia à qualche numero discreto* *Trouan dui altri differenti in esso.* *Dapoi terrai questo per consueto* *Che'llor produtto sempre sia eguale* *Alterzo cubo delle cose neto,* *El residuo poi suo generale* *Delli lor lati cubi ben sottratti* *Varra la tua cosa principale.* *In el secondo de cotestiatti* *Quando che'l cubo restasse lui solo* *Tu osseruarai quest'altri contratti,* *Del numer farai due tal part'à uolo* *Che l'una in l'altra si produca schietto* *El terzo cubo delle cose in stolo* *Delle qual poi, per communprecetto* *Torrai li lati cubi insieme gionti* *Et cotal somma sara il tuo concetto.* *El terzo poi de questi nostri conti* *Se solue col secondo se ben guardi* *Che per natura son quasi congionti.* *Questi trouai, & non con paβi tardi* *Nel mille cinquecentè, quatroe trenta* *Con fondamenti ben sald'è gagliardi* *Nella citta dal mar'intorno centa.*	01) When the cube with the cose beside it <x^3+px > 02) Equates itself to some other whole number, <=q> 03) Find two other, of which it is the difference. <u–v=q> 04) Hereafter you will consider this customarily 05) That their product always will be equal <uv=> 06) To the third of the cube of the cose net. 3/3, instead of $(p/3)^3$> 07) Its general remainder then 08) Of their cube sides , well subtracted, <u−−√3−v√3u3−v3> 09) Will be the value of your principal unknown. <=x> 10) In the second of these acts, 11) When the cube remains solo , <x^3=px+q> 12) You will observe these other arrangements: 13) Of the number <q> you will quickly make two such parts, <q=u+v> 14) That the one times the other will produce straightforward <uv=> 15) The third of the cube of the cose in a multitude, <p^3/3, instead of $(p/3)^3$> 16) Of which then, per common precept, 17) You will take the cube sides joined together. <u−−√3+v√3u3+v3 18) And this sum will be your concept. <=x> 19) The third then of these our calculations <x^3+q=px> 20) Solves itself with the second, if you look well after, 21) That by nature they are quasi conjoined. 22) I found these, & not with slow steps, 23) In thousand five hundred, four and thirty 24) With very firm and strong foundations 25) In the city girded around by the sea.

7.11 ROOTS OF THE POLYNOMIALS BY UTILISING SYNTHETIC DIVISION PROCESS

Horners method utilises synthetic division process to solve roots of polynomials. Synthetic division process use recursive process to obtain a division for a polynomial. If

$$P(x) = f_n(x) = a_0 + a_1x + a_1x^2 + \cdots + a_nx^n$$

and if this polynomial divides with $(x - x_0)$ to form polynomial

$$Q(x) = f_{n-1}(x) = b_1 + b_2x + b_3x^2 + \cdots + b_{n-1}x^{n-1}$$

Where $P(x) = f_n(x) = b_0 + (x - x_0)f_{n-1}(x)$ or $P(x) = b_0 + (x - x_0)Q(x)$

$R = b_0$ is the remainder of the synthetic division process. Of course if the remainder is equal to zero x_0 would be the root of this polynomial. Synthetic division can be calculated as following iterative equation:

$$b_n = a_n$$
$$b_i = a_i + b_{i+1}$$

Differentiating of $f_n(x)$with respect to x gives

$$f'_n(x) = f_{n-1}(x) + (x - x_0)f'_{n-1}(x)$$

and at $x=x_0$

$$f_n(x) = b_0$$
$$f'_n(x) = f_{n-1}(x) = Q(x)$$

Therefore Newton - Raphson iterative method can be applied to find the roots by using iterative process

$$x_n = x_{n-1} - \frac{f_n(x_{n-1})}{f_n'(x_{n-1})} = x_{n-1} - \frac{b_0}{f_n'(x_{n-1})}$$

$$x_n = x_{n-1} - \frac{b_0}{Q(x_{n-1})}$$

For example if a polynomial

$f_n(x) = 2x^4 - 3x^2 + 3x - 4$ is given, and x_0 is -2 {divide by (x+2)} synthetic division is applied as

x_0	a_4	a_3	a_2	a_1	a_0
-2	**2**	**0**	**-3**	**3**	**-4**
	b_4	b_3	b_2	b_1	b_0
	2	**0+-2*2=-4**	**5**	**-7**	**10**

$$Q(x) = f_{n-1}(x) = 2x^3 - 4x^2 + 5x - 7$$
$$P(x) = f_n(x) = 10 + (x + 2)Q(x)$$

Excel output for the process:

xo	a4	a3	a2	a1	a0
-2	2	0	-3	3	-4
	2	-4	5	-7	10

Q	-49
x1	-1.7959184

x1	a4	a3	a2	a1	a0
-1.7959184	2	0	-3	3	-4
	2	-3.5918367	3.4506456	-3.1970777	1.7416906

Q	-32.563821
x2	-1.7424329

x2	a4	a3	a2	a1	a0
-1.7424329	2	0	-3	3	-4
	2	-3.4848658	3.0721449	-2.3530065	0.0999559

Q	-28.866623
x3	-1.7389702

x3	a4	a3	a2	a1	a0
-1.7389702	2	0	-3	3	-4
	2	-3.4779405	3.048035	-2.3004421	0.0004003

Q	-28.63559

x4	-1.7389563

x4	a4	a3	a2	a1	a0
-1.7389563	2	0	-3	3	-4
	2	-3.4779125	3.0479377	-2.3002304	6.505E-09

Q	-28.634659
x5	-1.7389563

x5	a4	a3	a2	a1	a0
-1.7389563	2	0	-3	3	-4
	2	-3.4779125	3.0479377	-2.3002304	0

Q	-28.634659
x5	-1.7389563

Note that roots of a polynomial can be complex numbers, therefore it is better to calculate the roots as complex numbers. It should be also note that after reducing polynomial to a third, second or first degree remaining roots can be calculated by using standard root finding equations. An example program for Horner's method is listed below

Program 7.11.1 Complex Horner method example to find the roots of a polynomial (python language)

```
from math import *

def f_poly(c1,x):
    # this function calculates the value of
    # c least square curve fitting function
    n=len(c1);
    if n!=0.0:
      ff=c1[n-1];
      for i in range(n-2,-1,-1):
        ff=ff*x+c1[i];
    else:
      ff=0,0;
    return ff;

def horner(a):
  # roots of a polynomial by using synthetic division process
    n=len(a);
    x1=complex(1.56435,0.1);
    n1=n;
    nmax=150;
    tolerance=1.0e-10;
    n1=n+1;
    ref=3.1;
    y=[complex(0.0,0.0) for i in range(n1-2)];
    n2=n1-2;
    for k in range(0,n2):
          n=len(a);
          x=x1;
          xold=3.4878;
          ref=abs(xold-x);
          b=[complex(0.0,0) for i in range(n)];
          c=[complex(0.0,0.0) for i in range(n-1)];
          b[n-1]=a[n-1];
          ii=0;
          while ref>tolerance and ii<nmax:
            for i in range(n-2,-1,-1):
              b[i]=a[i]+b[i+1]*x;
            for i in range(n-2,-1,-1):  c[i]=b[i+1];
            Q=f_poly(c,x);
            P=b[0];
            x=x-P/Q;
            ref=abs(xold-x);
            ii=ii+1;
            xold=x;
          a=c;
          n=len(a);
          y[k]=x;
```

```
    return y;

a=[-4,3,-3,0,2];
print(horner(a));
```

```
runfile('D:/okul/SCO1/Mathd.py', wdir='D:/okul/SCO1')
[(1.254881884834291+1.34106353887572e-29j),
 (0.24203718580880437-0.9262454872675325j),
(0.24203718584251482+0.9262454872691118j),
 (-1.738956256536873-9.031220216115798e-12j)]
```

$$f(x) = x^5 - 3.5x^4 + 2.75x^3 + 2.125x^2 - 3.875x + 1.25$$

```
a=[1.25,-3.875,2.125,2.75,-3.5,1];
print(horner(a));
```

```
runfile('D:/okul/SCO1/horner1.py', wdir='D:/okul/SCO1')
[(1.0000000000000002-0.5j), (0.5000000000000011+1.0217019574503208e-15j), (1.9999999999895546-4.1185274253612687e-13j),
(1.0000000000176792+0.49999999997300376j), (-0.9999999999955536-3.5171310308612647e-12j)]
```

Now let us find roots of $f(x) = x^3 + 9813.18x^2 + 8571.08x + 0.781736 = 0$ by using horner method

```
a=[0.781736,8571.08,9813.18,1];
print(horner(a));
```

```
runfile('D:/okul/SCO1/horner1.py', wdir='D:/okul/SCO1')
[(-9.121576846347157e-05+4.9325894681659426e-26j), (-0.8734118615171431-4.424838606974589e-13j), (-9812.306496922753-
6.1173554997356664e-12j)]
```

Now let us find roots of $f(x) = 16x^4 - 40x^3 + 5x^2 + 20x + 6 = 0$ by using horner method

```
a=[6,20,5,-40,16]
print(Mathd.horner(a));
```

```
runfile('D:/okul/SCO1/Mathd.py', wdir='D:/okul/SCO1')
[(1.2416774447647838+0j), (1.9704460787298799-4.672866064934229e-20j), (-0.35606176190945776+0.16275838255939531j), (-
0.3560617615461311-0.16275838258481318j)]
```

Le Guerre method is also specificaly developed to find roots of a polynomial. Method formulations is as follows:

$$g(x) = [f'(x)]^2 - \frac{nf(x)f''(x)}{n-1} \qquad (2.15.10)$$

$$x_{k+1} = x_k - \frac{nf(x_k)}{f'(x_k) \pm (n-1)\sqrt{g(x_k)}} \quad (2.15.11)$$

In this definition, n is the degree of polynomial. The term $\sqrt{g(x_k)}$ in the equation can be complex (roots of polynomials can be complex numbers). Therefore the method will be written as complex variable method. By combining LeGuerre root finding method with synthetic division process, the remaining roots can also be found.

```
from math import *
from cmath import *

def f_poly(c1,x):
    # this function calculates the value of
    # c least square curve fitting function
    n=len(c1);
    if n!=0.0:
      ff=c1[n-1];
      for i in range(n-2,-1,-1):
        ff=ff*x+c1[i];
    else:
      ff=0,0;
    return ff;

def df_poly(c1,x):
    h=0.001
    return (-f_poly(c1,(x+2.0*h))+8.0*f_poly(c1,(x+h))-8.0*f_poly(c1,(x-h))+f_poly(c1,(x-2.0*h)))/(12.0*h);

def d2f_poly(c1,x):
```

```
    dx=0.001
    y1=-1.0/12.0*f_poly(c1,(x-2.0*dx))+4.0/3.0*f_poly(c1,(x-dx));
    y2=-5.0/2.0*f_poly(c1,x)+4.0/3.0*f_poly(c1,(x+dx))-1.0/12.0*f_poly(c1,(x+2.0*dx));
    return (y1+y2)/dx;

def LeGuerre(c1):
    # LeGuerre root finding method of polynomials
    maxit=500;
    iter=0;
    es=0.0000001;
    ea=1.1*es;
    tolerance=1e-15;
    n=len(c1)-1;
    x=complex(1.0,0.0);
    xold=x;
    while ea>es and iter<maxit:
      dfx=df_poly(c1,x);
      fx=f_poly(c1,x);
      d2fx=d2f_poly(c1,x);
      G=dfx/fx;
      H=G*G-d2fx/fx;
      y=sqrt((n*H-G*G)*(n-1.0));
      z1=G+y;
      z2=G-y;
      zbool=z1.real*z1.real+z1.imag*z1.imag> z2.real*z2.real+z2.imag*z2.imag;
      if zbool: z=z1;
      else: z=z2;
      aa=n/z;
      x=x-aa;
      iter=iter+1;
      #if x!=0: ea=abs((x-xold)/x)*100;
      if abs(fx)<tolerance: print("iter",iter);return x;
      xold=x;
    print("iter",iter);
    if iter>=maxit: print("Maximum number of iteration is exceeded \n result might not be valid");
    return x;

a=[2,0,1];
print(LeGuerre(a));
```

```
runfile('E:/okul/SCO1/LeGuuerre.py', wdir='E:/okul/SCO1')
iter 31
-1.414213562373095j
```

```
a=[6,20,5,-40,16]
print(LeGuerre(a));
```

```
runfile('E:/okul/SCO1/LeGuuerre.py', wdir='E:/okul/SCO1')
iter 5
(1.2416774447647838+0j)
```

7.12 COMPLEX ROOTS OF THE POLYNOMIALS BY BAIRSTOW'S METHOD

Bairstow's method is an iterative approach, like Horner's method it also uses synthetic division as basic tool. But instead of a single step division, a two step division is taken and utilised. The basic reason for that is to get a better utilisation of complex roots conjugate effect. Remember from the previous section that sythetic division recursion calculations has the form:

$f_n(x) = a_0 + a_1x + a_1x^2 + \cdots + a_nx^n$ (2.16.1)

and if this polynomial divides with $(x - x_0)$ to form polynomial

$f_{n-1}(x) = b_1 + b_2x + b_3x^2 + \cdots + b_{n-1}x^{n-1}$ (2.16.2)

Where $f_n(x) = b_0 + (x - x_0)f_{n-1}(x)$ (2.16.3) and $R = b_0$ is the remainder of the synthetic division process. Of course if the remainder is equal to zero x_0 would be the root of this polynomial. Synthetic division can be calculated as following iterative equation:

$b_n = a_n$ (2.16.4)

$b_i = a_i + b_{i-1}x_0$ (2.16.5)

Now if a second degree synthetic division is desired, the formulation will change a little. If second degree division equation is $x^2 - rx - s$ the result is a new polynomial in the form of

$f_{n-2}(x) = b_2 + b_3x + b_4x^2 + \cdots + b_nx^{n-2}$ (2.16.6) with the remainder

$R = b_1(x - r) + b_0$ (2.16.7) As with the normal synthetic division a simple recurance relation can be used as: $b_n = a_n$ (2.16.8)

$b_{n-1} = a_{n-1} + rb_n$ (2.16.9)

$b_i = a_i + rb_{i+1} + sb_{i+2}$ for i = n-2 to 0 (2.16.10)

As seen from the equation we have two roots (r and s) to solve simultaneously in one step calculation now. Consider that b coefficients are function of r and s, First degree Taylor series expansion of these coefficients can be written as

$b_1^n = b_1^{n-1} + \frac{\partial b_1^{n-1}}{\partial r}\Delta r + \frac{\partial b_1^{n-1}}{\partial s}\Delta s$ (2.16.11)

$b_0^n = b_0^{n-1} + \frac{\partial b_0^{n-1}}{\partial r}\Delta r + \frac{\partial b_0^{n-1}}{\partial s}\Delta s$ (2.16.12) Higher order terms of the Taylor series are ignored. The roots are defined as remainder terms of sysnthetic division as zero (b_0 and b_1 should be equal to zero) so if we assume the left hand side of the equation is zero, this equation becomes

$\frac{\partial b_1}{\partial r}\Delta r + \frac{\partial b_1}{\partial s}\Delta s = -b_1$ (2.16.13)

$\frac{\partial b_0}{\partial r}\Delta r + \frac{\partial b_0}{\partial s}\Delta s = -b_0$ (2.16.14) If partial derivatives of b is determined this set of equation can be easily solved. Bairstow showed that the partial derivatives can be obtained by synthetic division of b values. Then the partial derivatives can be substituted in above equation

$c_n = b_n$ (2.16.15)

$c_{n-1} = b_{n-1} + rc_n$ (2.16.16)

$c_i = b_i + rc_{i+1} + sc_{i+2}$ for i = n-2 to 1 (2.16.17)

So the equation becomes:

$c_2\Delta r + c_3\Delta s = -b_1$ (2.16.18)

$c_1\Delta r + c_2\Delta s = -b_0$ (2.16.19)

Solution of this equation by Gauss elimination process will give the following solution:

$W = \frac{c_1}{c_2}$ (2.16.20)

$\Delta s = \frac{b_0 - Wb_1}{c_2 - Wc_3}$ (2.16.21)

$\Delta r = \frac{-b_1 - c_3\Delta s}{c_2}$ (2.16.22)

$r^{n+1} = r^n + \Delta r^{n+1}$ (2.16.23)

$s^{n+1} = s^n + \Delta s^{n+1}$ (2.16.24)

This equation can be solved iteratively until the error term shrink to a minimum value

$|\varepsilon| = \left|\frac{\Delta r}{r}\right| + \left|\frac{\Delta s}{s}\right| < \varepsilon_s$ (2.16.25) when the error level drops under a specified stopping criterion, iteration stops, and roots can be calculated by using quadratic equation

$x = \frac{r \pm \sqrt{r^2 + 4s}}{2}$ (2.16.26)

Note that all these procedure can be calculated with complex numbers. It should be also note that after reducing polynomial to a third, second or first degree remaining roots can be calculated by using standard root finding equations. An example program is listed below:

$f(x) = x^5 - 3.5x^4 + 2.75x^3 + 2.125x^2 - 3.875x + 1.25$ with r=s=-1

7.13 ROOT FINDING BY USING SOFTWARE PACKAGES

In order to find roots, some built in functions already existed in package math programs. In Matlab-Octave function **fzero** is existed to find roots of a function and **roots** to find roots of a polynomial

```
>> format long
>> x=2;
>> y=@(x)x*x-2;
>> fzero(y,x)
ans =  1.41421356237310
>>
```

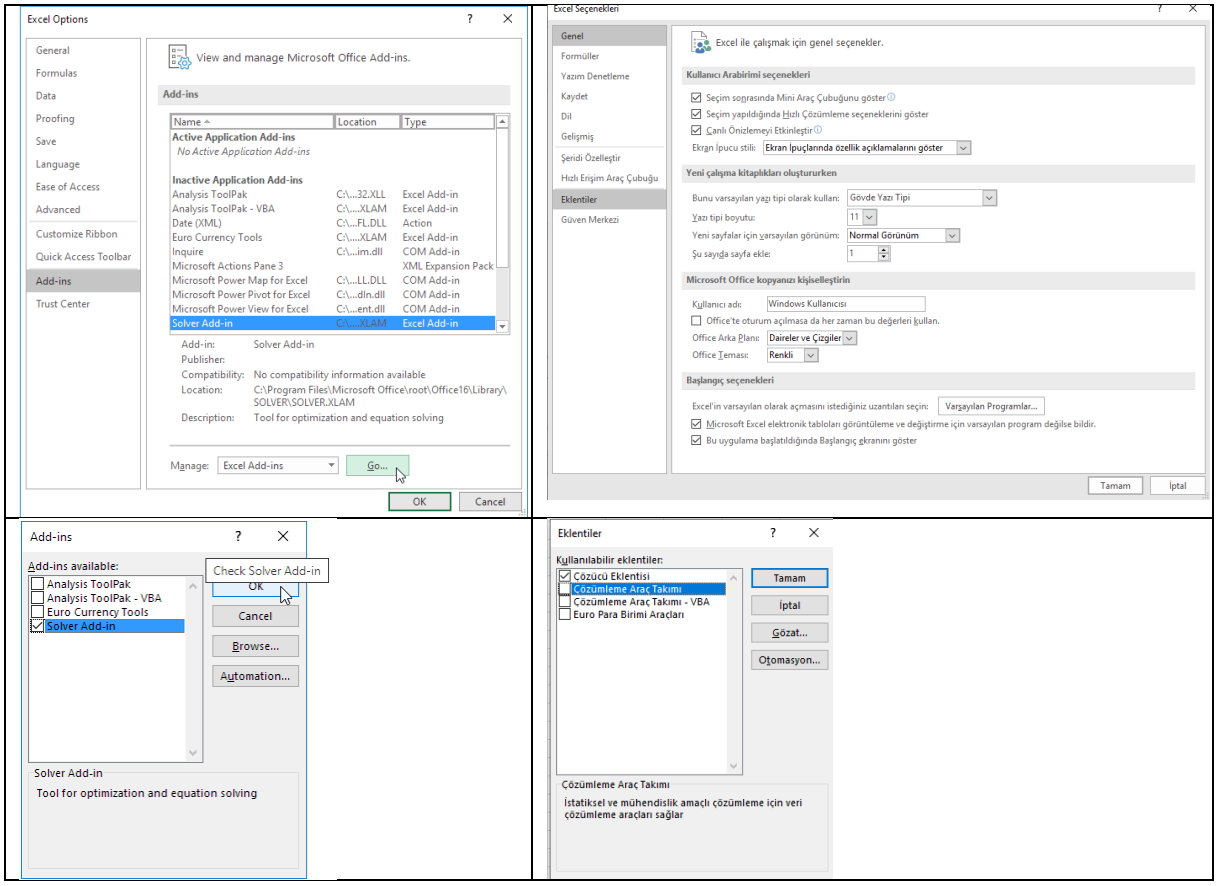
```
>> x=1;
>> y=@(x)x^5-3.5*x^4+2.75*x^3+2.125*x^2-3.875*x+1.25;
>> fzero(y,x)
```

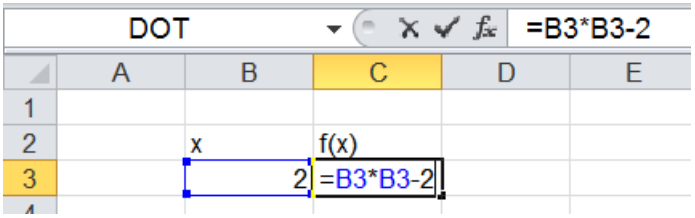
```
ans =  0.500000000000001
```

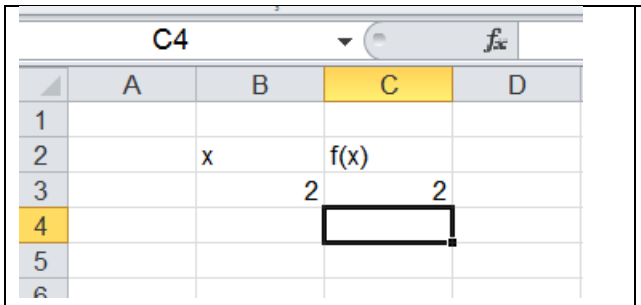
```
>> a=[1 -3.5 2.75 2.125 -3.875 1.25];
>> roots(a)
ans =
   2.000000000000003 + 0.000000000000000i
  -1.000000000000002 + 0.000000000000000i
   1.000000000000000 + 0.500000000000000i
   1.000000000000000 - 0.500000000000000i
   0.499999999999999 + 0.000000000000000i
```

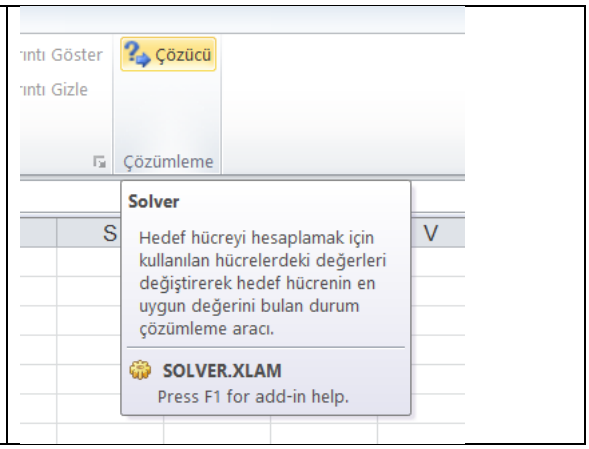
```
>> y=@(x)x*x+2;
>> x=1+1i;
>> fsolve(y,x)
ans =
 -0.000000000000016 + 1.414213562374690i
```

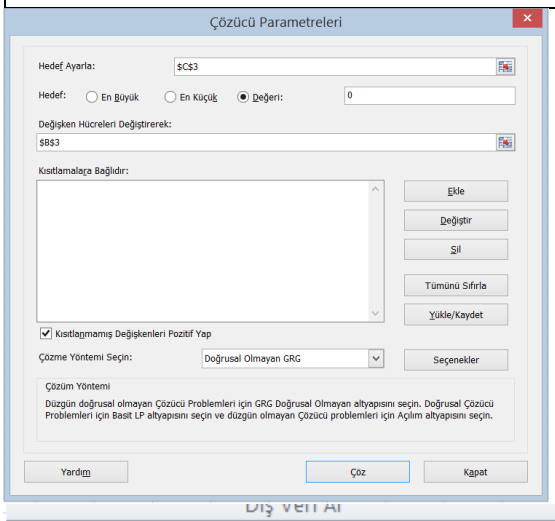
In excel, solver is existed to find real roots: We can use solver add in. In order to open solver:
On the File tab, click Options. (in turkish version dosya, seçenekler)
Under Add-ins(in turkish version eklentiler), select Solver Add-in (in turkish version Çözücü eklentisi), and click on the Go(Git..) button. You can find the Solver on the Data tab(Veri), in the Analyze group.

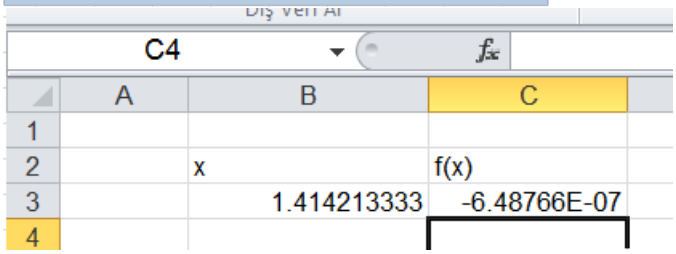












7.13 FINDING THE ROOTS LIMITS AND MULTIPLE ROOTS

In a one variable function, if a root to be find and if only one root existed in the given region boundaries of the function should carry inverse sign. To further process this, if given region is divided up to smaller region and function values at the limits are checked, the root area can be narrowed down. region can be enlarged to the side that it is estimated to have a root to ensure that at least one root is existed in a given region.

Algorithm for root searching:

```
Set search limit dx, search and region a,b  (a is starting point, b is end point of the search)
x1=a
x2=a+dx
while x2<b
  if f(x1)*f(x2) less than 0 then
      root existed between x1 and x2
      set root limists as x1 and x2
      x1=x2;
      x2=x1+dx
   if not
     x2=x2+dx
   end of if block
```

end of while block
return x1,x2 sets as root limits

```
# -*- coding: utf-8 -*-
"""
Created on Sat Dec 18 07:06:10 2021

@author: M. Turhan ÇOBAN
"""
# function f(x) should be defined by using class if_x

from math import *
from if_x import *

def root_limits(f,a,b):
    # find root limits of multiple roots
    # in between a and b
    dx1=1e-2
    x1=a
    x2=a+dx1
    f1=f.func(x1)
    f2=f.func(x2)
    c = [[0.0 for j in range(2)] for i in range(200)]
    n=0
    while x2<b:
        if f1*f2<=0:
            c[n][0]=x1
            c[n][1]=x2
            n=n+1
            x1=x2+dx1
            x2=x1+dx1
            f1=f.func(x1)
            f2=f.func(x2)
        else:
            x2=x2+dx1
            f2=f.func(x2)
    e=[[0 for x in range(2)] for y in range(n)]
    for i in range(0,n):
        e[i]=c[i]
    return e

def roots_bisection(f,a,b):
    c=root_limits(f,a,b)
    n=len(c)
    print("c=",c,"n=",n)
    x=[0.0 for x in range(n)]
    for i in range(n):
        x[i]=bisection(f,c[i][0],c[i][1])
    return x

def roots_secant(f,a,b):
    c=root_limits(f,a,b)
    n=len(c)
    x=[0.0 for x in range(n)]
    for i in range(n):
        x[i]=secant(f,c[i][1])
    return x

def bisection(f,x1,x2):
   # bisection root finding method
   f1=f.func(x1)
   r=(x1+x2)/2.0
   fr=f.func(r)
   eps=1.0e-6
   Maxiter=100
   i=0
   while abs(fr)>eps and i<Maxiter:
       if f1==0:
           return x1
       elif fr==0:
           return r
       elif f1*fr<0:
           x2=r
       else:
```

```
            x1=r
            f1=fr
        r=(x1+x2)/2.0
        fr=f.func(r)
        i=i+1
    return r


def secant(f,x):
    # secant root finding method
    nmax=100
    tolerance=1.0e-10
    for i in range(0,nmax):
        fx= f.func(x);
        dfx=f.dfunc(x,1)
        x=x-fx/dfx
        if abs(fx)<tolerance: return x
    return x



class f1(if_x):func=lambda self,x:sin(x)
f = f1()
a=float(input("enter lower limit : "))
b=float(input("enter upper limit : "))
x=roots_bisection(f,a,b);
print("Bisection x=",x)
x=roots_secant(f,a,b);
print("Secant x=",x)
```

```
runfile('D:/okul/SCO1/root_limits.py', wdir='D:/okul/SCO1')

enter lower limit : 1

enter upper limit : 8
c= [[1.0, 3.1499999999999764], [3.159999999999976, 6.289999999999909]] n= 2
Bisection x= [3.1415933609008553, 6.283185243606476]
Secant x= [3.141592653589793, 6.283185307179586]
```

$$\boldsymbol{f(x) = x * sin(x) - Bi * cos(x) = 0 \quad 0 \le x \le 50}$$

```
Bi=0.1
class f1(if_x):func=lambda self,x:x*sin(x)-Bi*cos(x)
f = f1()
a=0
b=50
x=roots_bisection(f,a,b);
print("Bisection x=",x)
x=roots_secant(f,a,b);
print("Secant x=",x)
```

```
runfile('D:/okul/SCO1/root_limits1.py', wdir='D:/okul/SCO1')
Reloaded modules: if_x
c= [[0, 0.3200000000000001], [0.3300000000000001, 3.179999999999976], [3.189999999999976, 6.29999999999991],
[6.30999999999991, 9.439999999999843], [9.449999999999843, 12.579999999999776], [12.589999999999776, 15.71999999999971],
[15.729999999999709, 18.86000000000015], [18.87000000000015, 22.00000000000064], [22.01000000000064, 25.14000000000113],
[25.150000000001132, 28.28000000000162], [28.290000000001623, 31.420000000002112], [31.430000000002114,
34.57000000000169], [34.58000000000169, 37.71000000000107], [37.720000000001065, 40.85000000000044], [40.86000000000044,
43.98999999999982], [43.999999999999815, 47.12999999999919]] n= 16
Bisection x= [0.31105224609375015, 3.1730970311164612, 6.299059430360703, 9.435376047491872, 12.574323088526501,
15.714326819777199, 18.854859549105317, 21.99569488287036, 25.136719484330357, 28.277870202066133, 31.419109303804362,
34.56041266113688, 37.70176424339519, 40.84315287500664, 43.98457065701467, 47.12601175933996]
Secant x= [0.31105284820029777, 3.1730971766928695, 6.299059359895646, 9.435375975760847, 12.574323161037869,
15.714326801776345, 18.854859544307704, 21.995694888011272, 25.136719451598626, 28.277870201784616, 31.419109301563672,
34.560412665381506, 37.701764232628236, 40.8431528825808, 43.98457067086168, 47.12601177101854]
```

$$\boldsymbol{f(r) = r * J_1(r) - Bi * J_0(r) = 0 \quad 0 \le r \le 50}$$

```
def J(v,x):
    JJ=1e-6
    a=1.0
    fact=1
    b=0.0
    if x <=20:
        x5=1
        lna=0.0;
```

```
        x1=v*log(0.5*x)
        lna1=log(0.25*x*x)
        for k in range(0,100):
            x2=lna
            x3=lgamma(k+1)
            x4=lgamma(v+k+1)
            b=x1+x2-x3-x4
            b=x5*exp(b)
            lna=lna+lna1
            x5=-1*x5
            JJ=JJ+b
    elif (v==0 and x==0):
        JJ=1
    else:
        JJ=sqrt(2.0/pi/x)*cos(x-0.5*v*pi-0.25*pi)
    return JJ


Bi=0.1
class f1(if_x):func=lambda self,r:r*J(1,r)-Bi*J(0,r)
f = f1()
a=0.0001
b=50
r1=roots_bisection(f,a,b);
print("Bisection r=",r1)
r2=roots_secant(f,a,b);
print("Secant r=",r2)
```

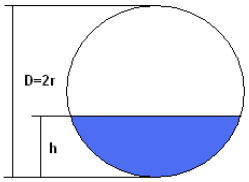
```
 runfile('D:/okul/SCO1/root_limits1.py', wdir='D:/okul/SCO1')
Reloaded modules: if_x
c= [[0.0001, 0.4501000000000003], [0.4601000000000003, 3.860099999999962], [3.8700999999999617, 7.030099999999894],
[7.040099999999894, 10.190099999999827], [10.200099999999827, 13.34009999999976], [13.35009999999976, 16.48009999999978],
[16.49009999999978, 19.630100000000272], [19.640100000000274, 22.790100000000766], [22.800100000000768,
25.930100000001257], [25.94010000000126, 29.070100000001748], [29.08010000000175, 32.210100000002164],
[32.22010000000216, 35.35010000000154], [35.36010000000154, 38.490100000000915], [38.50010000000091, 41.63010000000029],
[41.64010000000029, 44.770099999999665], [44.78009999999966, 47.92009999999904]] n= 16

Bisection r= [0.44168172607421896, 3.8577119041442485, 7.029821994399918, 10.18329652318937, 13.331190502738714,
16.476705308055656, 19.620950191116606, 22.78093644747811, 25.921997215844456, 29.063172932960356, 32.20442999134279,
35.345746376611345, 38.48710828313919, 41.62850489029914, 44.76992910871472, 47.9113750411024]

Secant r= [0.44168100975596203, 3.8577123228063437, 7.029821949601012, 10.183296530173594, 13.331190565014792,
16.476705333761902, 19.620950276518155, 22.78093634552872, 25.921997100437736, 29.06317281268636, 32.2044298521908,
35.34574654007115, 38.4871082732571, 41.62850485557948, 44.76992895208186, 47.911375151226714]
```

7.14 PROBLEMS

PROBLEM 1 Find the roots of $f(x) = x^2 - 0.9x - 8.5$ by using bisection method

PROBLEM 2 Calculate $\sqrt{7}$ by using one of the root finding method

PROBLEM 3



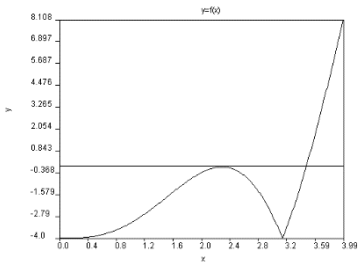
A spherical tank of radius r is filled with a liquid of height h. The Volume of liquid (V) in the tank is given by he following eqaution

$V = \pi h^2(3r - h)/3$

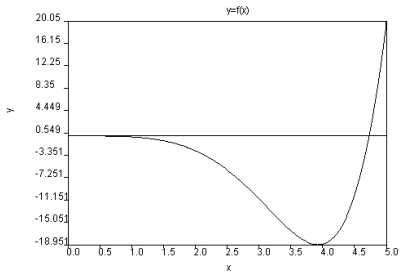
If r = 1 m and V= 0.5 m^3 find the liquid height

Note : due to geometry of a sphere h can not be less than 0 and can not be bigger than D=2r. The maximum amount of liquid ths spherical container can be take is 4.18879 m^3 (you do not need this data to solve problem)

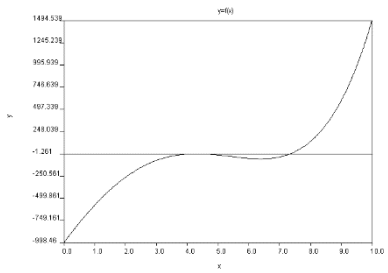
PROBLEM 4 Find the roots of $f(\boldsymbol{x}) = x^2|sin(x)| - 4$. by using one of the root finding method



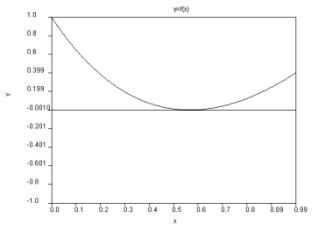
PROBLEM 5 Find the roots of $f(\boldsymbol{x}) = \cos(x)\cosh(x) - 1$ by using one of the root finding method



PROBLEM 6 Find the roots of $f(\boldsymbol{x}) = x^4 - 8.6x^3 - 35.5x^2 + 464.4x - 998.46$ by using one of the root finding method Try x_0=4 and $\mathbf{7 \leq x \leq 8}$

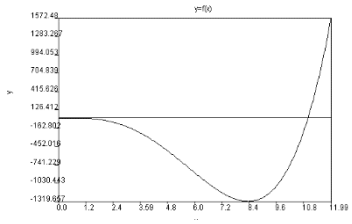


PROBLEM 7 Function $\boldsymbol{f(x)} = x^2 - 2e^{-x} + e^{-2x}$ has only one real root. Find the root bu using Newton-Raphson method. Use as starting value x_0=1

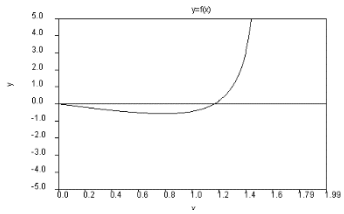


PROBLEM 8

Find all real roots of function $f(\boldsymbol{x}) = x^4 - 7.223x^3 + 13.4475x^2 - 0.672x - 10.223$



PROBLEM 9 Find roots of functionf $(\boldsymbol{x}) = \tan(\mathrm{x}) - 2x$.



PROBLEM 10 For Turbulent flow inside of a pipe (Re>2300) friction coefficient inside of a pipe can be calculated by using Colebrook equation.

$$\frac{\mathbf{1}}{\sqrt{\boldsymbol{f}}} = -2.0log_{10}\left[\frac{\varepsilon/D}{3.7} + \frac{2.51}{Re\sqrt{f}}\right]$$

Re is Reynolds number, ε is surface rougness. Write a program to calculate total pressure drop in the pipe for a given Re number, pipe diameter, pipe length and local pressure drop coefficient.
f : friction coefficient
D pipe diameter
Hint : Change the colebrook equation into the form

$$F(x) = X + 2.0log_{10}\left[\frac{\varepsilon/D}{3.7} + \frac{2.51X}{Re}\right] = 0$$

Then find the root of the equation where $X = \frac{1}{\sqrt{f}}$
Use Haaland equation to obtaing the first estimate of the solution which has the form of

$$\frac{\mathbf{1}}{\sqrt{\boldsymbol{f}}} = -1.8log_{10}\left[\frac{(\varepsilon/D)^{1.11}}{3.7} + \frac{6.9}{Re}\right]$$

PROBLEM 11
In round smooth tubes for turbulent flow region Fanning friction factor is given as:

$$\frac{\mathbf{1}}{\sqrt{\boldsymbol{f}}} = 1.737\ln\left[Re\sqrt{f}\right] - 0.4$$

This equation is valid for $4 * 10^3 < Re < 3 * 10^6$ By using one of the root finding methods, calculate friction factor for Re=5000. You can utilize any root finding method you learned so far.
Note : You can assume that f is changing between 0,001 and 0.01
PROBLEM 12
van der Waals equation of state for the gases :
$P = \frac{RT}{v-b} - \frac{a}{v^2}$ In this equation
$a = \frac{27}{64}\frac{R^2T_c^2}{P_c}$
$b = \frac{RT_c}{8P_c}$
Redlich ve Kwong equation of states
$P = \frac{RT}{v-b} - \frac{a}{v(v+b)T^{1/2}}$ In this equation
$a = 0.42748\frac{R^2T_c^{2.5}}{P_c}$
$b = 0.08664\frac{RT_c}{P_c}$
O2, for oxygen P_c=5.043 MPa T_c=154.58 K M=31.999 kg/kmol.
N2, for nitrogen P_c=3.39 MPa T_c=126.2 K M=28.013 kg/kmol.
R=8314.5 J/kmolK
Benedict -Webb-Rubin (BWR) equation of state

$$P = \frac{RT}{v} + \frac{RTB_0 - A_0 - C_0/T^2}{v^2} + \frac{RTb - a}{v^3} + \frac{a\alpha}{v^6} + \frac{c}{v^3}\left(1 + \frac{\gamma}{v^2}\right)e^{-\frac{\gamma}{v^2}}$$

Gas	A_0	B_0	$C_0.10^{-6}$	A	B	$c.10^{-6}$	$\alpha.10^3$	$\gamma.10^3$
O_2	1.49880	0.046524	0.0038617	-0.040507	-0.00027963	-0.00020376	0.008641	0.359
N_2	1.1925	0.0458	0.0058891	0.01490	0.00198154	0.000548064	0.291545	0.75

Develop a program to calculate v when T and P is given. By using
A) van der Waals equation of state : gas : oxygen and nitrogen
B) Redlich Kwong equation of state : gas : oxygen and nitrogen
C) Benedict-webb rubin equation of state : gas : oxygen
D) Benedict-webb equation of state : gas : nitrogen
E) Compare the result with ideal gas equation $P = \frac{RT}{v}$

PROBLEM 13
Find the root of function $f(x)=x\,e^{-x/2} + x^3 + 1$ with the limits of x_1=-1, x_2=0, x_3=1 ny using muller method.

PROBLEM 14

Find the root of function $f(x)=x^4-20x^3-25x^2+100x+130$ by using Leguerre method.

PROBLEM 15

Find the root of fuction $f(x)=x^2+1$.

PROBLEM 16

For a paralel flow heat exchanger Number of transfer unit(NTU) and effectiveness (ε) equation given as follows

$$\varepsilon = \frac{1 - exp[-NTU(1 + C^*)]}{(1 + C^*)}$$

After the experiments for a specific heat exchanger NTU=1.989 and ε = 0.574 is found from the experiment

a) Determine the range C^* . For this start C^*=0.1 and increase 0.1 in each step
b) Calculate C^* by using false position method.

PROBLEM 17

Two end of 5 cm long plate with crossection of 0.1cm x 20 cm plate is held at 80 °C ve 60 °C. The environmental temperature is T_∞ = 10 °C, convective heat transfer coefficient h=15 W/m^2 K. Bar cross sectional area A=2×10^{-4} m^2, and perimeter P=0.402 m.In the experiment mid point(x=0.025 m) temperature is found as T=67.5 °C. In this conditions, temperature distribution in the bar is given as

$$T - T_\infty = 68.7e^{-0.025m} + 11.3e^{0.025m}$$

In this equation m is defined as: $m = (\frac{hP}{kA})^{0.5}$

a) Find the range of m, stating from m=5, and increasing 5 at each step.
b) Solve m by using Newton-Raphson method
c) Determine thermal conductivity coefficient k from this value.

PROBLEM 18

The cost of the heat exchanger(M), is defined as a function of the length of heat excahger, L, diameter of the heat exchanger, D as**:** ***M=900 + 1100 D^{2,5} L + 320 D L*** In the optimization study the length of the heat exchanger is found as L=1.3 m and the cost as M=1900 TL. Find the diameter of the heat exchanger.

PROBLEM 19

An instrument 20000 TL worth is purcahased with no first payment and 4000 TL annual payment for 6 years. Calculate the interest rate applied by using secant method.

$$A = P\frac{i(i+1)^n}{(i+1)^n - 1}$$

In the equation P is the present worth, A is the annual payment, i is the interestrate, n is the number of payment years.

PROBLEM 20

In order to heat a substance to a required temperature the energy equation is given as

$Q = \int mC_p dT$ eşitliği ile bulunmaktadir.

A substance with an initial temperature of 30 °C, with a mass m=0.2 kg, is heated to some temperature and total energy spend is calculated as 3069.7 kJ . The specific heat of the substance is given with the equation C_p = 41.87+ 1.6748* T + 0.007537* T^2 (kJ/kg K). Find the final temperature by using bi-section method.

PROBLEM 21

Find the roots of $f(x) = x^3-6x^2+11x-6.1$ by using Legurre method

PROBLEM 22 Find the root(s) of the following equation in the range of [-3.5,3.5]

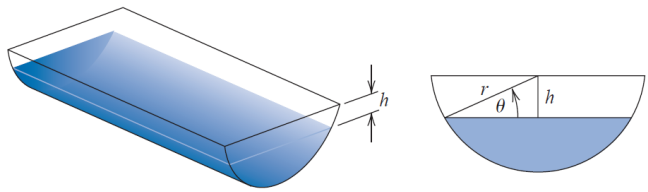
$$f(x) = \frac{1}{\sqrt{2\pi}}e^{-(\frac{1}{2})x^2} + \frac{1}{10}\sin(\pi x)$$

PROBLEM 23 Find the root(s) of the following equation

$$f(x) = \left[10x\left(1 - e^{-\frac{0.0004}{2000x}}\right) - 0.00001\right]$$

PROBLEM 24 In celestial mechanics, **Kepler's equation** is important. It reads $\boldsymbol{x = y - \varepsilon \sin y}$, in which x is a planet's mean anomaly, y its eccentric anomaly, and ε the eccentricity of its orbit. Taking $\varepsilon = 0.9$, construct a table of y for 30 equally spaced values of x in the interval 0 to π. Use Newton's method to obtain each value of y. The y corresponding to an x can be used as the starting point for the iteration when x is changed slightly.

PROBLEM 25 A trough of length L has a cross section in the shape of a semicircle with radius r. When filled with water to within a distance h of the top, the volume V of water is



$$V = L\left[0.5\pi r^2 - r^2 \arcsin\left(\frac{h}{r}\right) - h(r^2 - h^2)^{1/2}\right]$$

Suppose L=1 m, r=0.25 m and V=0.05 m^3 Find the depth of water**,** h

PROBLEM 26 Following functions and root values are given. Investigate number of iteration they are taken when Newton, Halley, Abbasbandy, Thiele 1 and Thiele 2 methods are used.

$f(x) = x^3 + 4x^2 - 25 \qquad x^* = 2.0352684811\ 82$

$f(x) = x^3 - e^x - 3x + 2 \qquad x^* = 0.2575302854\ 39$

$f(x) = x^3 - 10 \qquad x^* = 2.1544346900\ 32$

$f(x) = \cos(x) - x \qquad x^* = 0.7390851332\ 15$

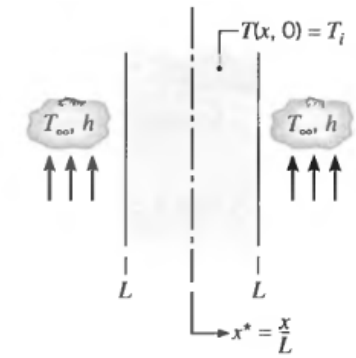
$f(x) = \sin^2(x) - x^2 + 1 \qquad x^* = 1.4044916482\ 15$

$f(x) = x^2 + \sin(x/5) - 1/4 \qquad x^* = 0.4099920179\ 89$

(b) Did you remember to plot the function before using bisection?

c) repeat using the root by Newton-Raphson, Illinois, Halley, Abbasbandy , Thiele1 and Thiele 2 methods. Find the number of iteration it takes in each case.

PROBLEM 28 In time dependent heat transfer, heat transfer of a Wall where both sides are exposed to convective heat transfer with convective heat transfer coefficient h and temperature T_∞ is shown in the figure (reference Fundamentals of heat and mass transfer Incropera, DeWitt, Bergman**)**



Time dependent temperature difference for such a wall is given by the equation

$$\theta_0^* = \frac{T_0 - T_\infty}{T_i - T_\infty} = \sum_{n=1}^{\infty} C_n \exp(-\xi_n^2 Fo)\cos(\xi_n x^*)$$

Where $Fo = \dfrac{\alpha t}{L^2}$ is Fourier number t is time(second), $\alpha = \dfrac{k}{\rho C_p}$ is thermal diffisuvity coefficient, L is the half length of the Wall. In order to solve coefficients ξ_n (eigenvalues) the following equation should be solved.

$\tan\xi_n - \frac{Bi}{\xi_n} = 0$ where $Bi = \frac{hL}{k}$ is Biot number, h is thermal covection coefficient and k is thermal conductivity of the Wall material. Assume Bi number is given and developed an algorithm to solve first 4 roots of the ξ_n. In order to help you The first 4 roots of the equation for various Bi number are given in the following table.

$Bi = \frac{hL}{k}$	ξ_1	ξ_2	ξ_3	ξ_4
0	0	3.1416	6.2832	9.4248
0.001	0.0316	3.1419	6.2833	9.4249
0.002	0.0447	3.1422	6.2835	9.4250
0.004	0.0632	3.1429	6.2838	9.4252
0.006	0.0774	3.1435	6.2841	9.4254
0.008	0.0893	3.1441	6.2845	9.4256
0.01	0.0998	3.1448	6.2848	9.4258
0.02	0.1410	3.1479	6.2864	9.4269
0.04	0.1987	3.1543	6.2895	9.4290
0.06	0.2425	3.1606	6.2927	9.4311
0.08	0.2791	3.1668	6.2959	9.4333
0.1	0.3111	3.1731	6.2991	9.4354
0.2	0.4328	3.2039	6.3148	9.4459
0.3	0.5218	3.2341	6.3305	9.4565
0.4	0.5932	3.2636	6.3461	9.4670
0.5	0.6533	3.2923	6.3616	9.4775
0.6	0.7051	3.3204	6.3770	9.4879
0.7	0.7506	3.3477	6.3923	9.4983
0.8	0.7910	3.3744	6.4074	9.5087
0.9	0.8274	3.4003	6.4224	9.5190
1.0	0.8603	3.4256	6.4373	9.5293
1.5	0.9882	3.5422	6.5097	9.5801
2.0	1.0769	3.6436	6.5783	9.6296
3.0	1.1925	3.8088	6.7040	9.7240
4.0	1.2646	3.9352	6.8140	9.8119
5.0	1.3138	4.0336	6.9096	9.8928

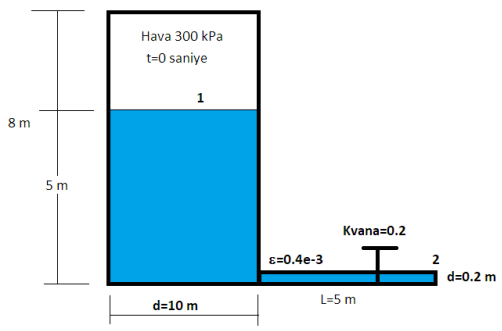
PROBLEM 29 Solve equation $3^x x^3 = 1$

PROBLEM 30 Calculate all roots of the following polynomials

$$f(x) = x^3 - x^2 + 2x - 2$$

$$f(x) = 3.704x^3 + 16.3x^2 - 21.97x + 9.34$$

PROBLEM 31A water container filled to H=5 m level. Total height of the container is HT=8 meters. At the top of water air with 300 kPa and 27 C existed. The top of the storage tank is closed. Diameter of the container is 10 m. Water is discharged through a pipe of d=0.2 m in diameter and L=5 m in length. There is a globe valve at this pipe (KL=0.2). Calculate velocity of the discharge water as a function of time

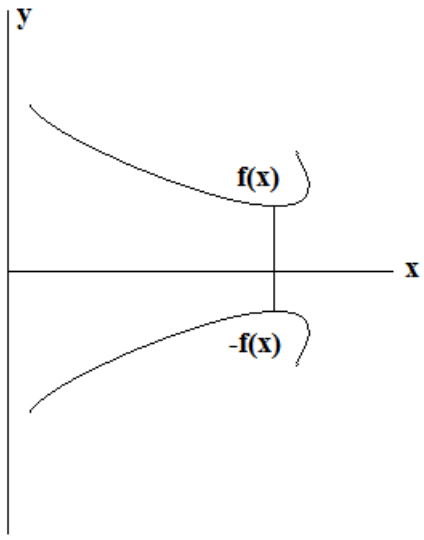


8. OPTIMISATION

Optimization is the name given to maximum or minimum finding process. When it is considered in mathematically, the root of the derivative of the function defines the optimum point.

$$\frac{df(x)}{dx} = 0$$

As the mathematical process, maximum and minimum is similar processes because
maximum (f(x)) = minimum(-f(x))



8.1 NEWTON'S METHOD

Newton- Raphson is investigated as one of the root finding methods. Due to the fact that roots of the derivatives of a function is also given the optimum point makes Newton method a cendidate to find optimum through the roots of the derivative of the function.
Minimum of f(x)
Roots of $g(x) = \frac{df(x)}{dx} = 0$
Newton raphson formula

$$x_{n+1} = x_n - \frac{g(x)}{\frac{dg(x)}{dx}} = x_n - \frac{\frac{df(x)}{dx}}{\frac{d^2f(x)}{dx^2}}$$

```
def f(x):
    y=x*x-2*x+5
    return y
def df(x):
    y=2.0*x-2
    return y
def d2f(x):
    y=2.0
    return y

def opt_newton(df,d2f,x):
    eps=1e-8;
    for i in range(100):
        xold=x
        x-=df(x)/d2f(x)
        dx=abs(x-xold)
        if dx<eps: return x
    return x

x=float(input("x="))
x=opt_newton(df,d2f,x)
print("x=",x)
```

```
runfile('E:/okul/SCO1/opt_newton.py', wdir='E:/okul/SCO1')

x=0.5
x= 1.0
```

Secant method: Secant method is the same as newton's method except derivatives calculated numerically

```
def f(x):
    y=x*x-2*x+5
    return y
def df(f,x):
    dx=0.01
    y=(f(x+dx)-f(x-dx))/(2.0*dx)
    return y
def d2f(f,x):
    dx=0.01
    y=(f(x+dx)-2.0*f(x)+f(x-dx))/(dx*dx)
    return y

def opt_secant(f,x):
    eps=1e-8;
    for i in range(100):
        xold=x
        x-=df(f,x)/d2f(f,x)
        dx=abs(x-xold)
        if dx<eps: return x
    return x

x=float(input("x="))
x=opt_secant(f,x)
print("x=",x)
```

```
runfile('E:/okul/SCO1/opt_secant.py', wdir='E:/okul/SCO1')

x=0.5
x= 1.0
```

Secant method in excel

Secant optimsation f(x)=1/3*x*x*x-2*x+5

a	2						
Δx	0.01						
x	f(x)	x+Δx	x-Δx	f(x+Δx)	f(x-Δx)	f'(x)	f''(x)
1	3.333333333	1.01	0.99	3.323434	3.343433	-0.99997	2
1.499983333	3.124995834	1.509983	1.48998	3.127646	3.122646	0.249983	2.999967
1.41665463	3.114390349	1.426655	1.40665	3.114601	3.114463	0.006944	2.833309
1.414203901	3.114381917	1.424204	1.4042	3.114523	3.114523	6.01E-06	2.828408
1.414201777	3.114381917	1.424202	1.4042	3.114523	3.114523	4.49E-12	2.828404
1.414201777	3.114381917	1.424202	1.4042	3.114523	3.114523	0	2.828404
1.414201777	3.114381917	1.424202	1.4042	3.114523	3.114523	0	2.828404
1.414201777	3.114381917	1.424202	1.4042	3.114523	3.114523	0	2.828404
1.414201777	3.114381917	1.424202	1.4042	3.114523	3.114523	0	2.828404
1.414201777	3.114381917	1.424202	1.4042	3.114523	3.114523	0	2.828404
1.414201777	3.114381917	1.424202	1.4042	3.114523	3.114523	0	2.828404
1.414201777	3.114381917	1.424202	1.4042	3.114523	3.114523	0	2.828404
1.414201777	3.114381917	1.424202	1.4042	3.114523	3.114523	0	2.828404
1.414201777	3.114381917	1.424202	1.4042	3.114523	3.114523	0	2.828404
1.414201777	3.114381917	1.424202	1.4042	3.114523	3.114523	0	2.828404

1.414201777	3.114381917	1.424202	1.4042	3.114523	3.114523	0	2.828404
1.414201777	3.114381917	1.424202	1.4042	3.114523	3.114523	0	2.828404
1.414201777	3.114381917	1.424202	1.4042	3.114523	3.114523	0	2.828404
1.414201777	3.114381917	1.424202	1.4042	3.114523	3.114523	0	2.828404
1.414201777	3.114381917	1.424202	1.4042	3.114523	3.114523	0	2.828404
1.414201777	3.114381917	1.424202	1.4042	3.114523	3.114523	0	2.828404
1.414201777	3.114381917	1.424202	1.4042	3.114523	3.114523	0	2.828404
1.414201777	3.114381917	1.424202	1.4042	3.114523	3.114523	0	2.828404
1.414201777	3.114381917	1.424202	1.4042	3.114523	3.114523	0	2.828404
1.414201777	3.114381917	1.424202	1.4042	3.114523	3.114523	0	2.828404
1.414201777	3.114381917	1.424202	1.4042	3.114523	3.114523	0	2.828404
1.414201777	3.114381917	1.424202	1.4042	3.114523	3.114523	0	2.828404
1.414201777	3.114381917	1.424202	1.4042	3.114523	3.114523	0	2.828404
1.414201777	3.114381917	1.424202	1.4042	3.114523	3.114523	0	2.828404

Newton method does not give if the found roots of first derivative is a maximum or minimum, but if we apply a second derivative test(third derivative of original function) it can be found if the optimum point is maximum or minimum

Root of function $g(x) = \frac{df(x)}{dx} = f'(x) = 0$

First derivative $g'^{(x)} = \frac{d^2 f(x)}{dx^2} = f''(x)$

Second derivative $g''(x) = \frac{d^3 f(x)}{dx^3} = f'''(x)$

If $g''^{(x)} < 0 \quad Maximum\ point$

Else if $g''^{(x)} > 0 \quad Minimum\ point$

Else if $g''^{(x)} = 0 \quad Extremum\ \ point$

```
from math import *
from if_x import *

def opt_newton(f,x):
    eps=1e-8;
    for i in range(100):
        xold=x
        x-=f.dfunc(x,1)/f.dfunc(x,2)
        dx=abs(x-xold)
        if dx<eps: return x
    return x


def optimum_test(f,x):
   # optimum test through second derivatine
   # x is maximum-minimum-extremum point
   y=f.dfunc(x,3)
   s=""
   if y==0.0 :  s="extremum"
   elif y>0.0:  s="minimum"
   else :       s="maximum"
   return s

class f1(if_x):func=lambda self,x:(x*x-x+5)
f=f1()
x=1.0
r=opt_newton(f,x)
```

```
print(" optimum point  : ",r,"\nFUNCTION VALUE : ",f.func(r),optimum_test(f,r));
```

```
runfile('D:/okul/SCO1/optimum_test.py', wdir='D:/okul/SCO1')
Reloaded modules: if_x
 optimum point  :  0.4999999999981659
FUNCTION VALUE :  4.75 maximum
```

8.2 ONE DIMENSIONAL NON-LINEAR FIBONNACHI SEARCH (GOLDEN SEARCH) METHOD

Series 0,1,1,2,3,5,8,13,21,34,… is called fibonnachi series. General definition is in the form of F(n)=F(n-1)+F(n-2) F(0)=0, F(1)=1

For the big fibonnaci terms the ration of two following fibonnachi number becomes constant. This constant is known as golden ratio,

$$R = \lim_{n\to\infty} \frac{F(n-1)}{F(n)} = \frac{\sqrt{5}-1}{2} = 0.618033989$$

Leonardo Pissano Fibbonachi

If a minimum being searched in a given region (a,c). In order to determine if a minimum is existed, region should be divided up into three sub-regions. This requires avaluation of two intermediate data points. Neigbors of the minimum point remains and the third region is eliminated. In the new avaluation a new two points will be required. Golden ratio is given us a fine propery so that one of the selected points will always be one of the new search point. So that only one new point will be evaluated in each step of the evaluation.

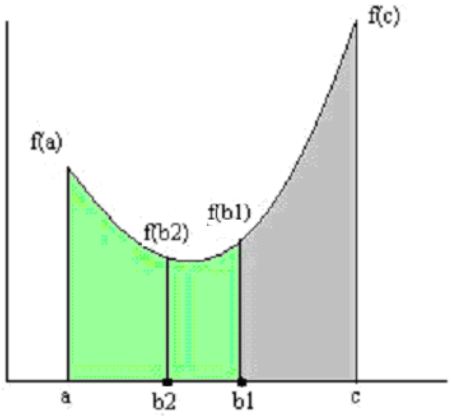


Figure Golden ratio search

If minimumin in region [a,c] is required

The length of the region will be d0=(c – a) .

Then b1 is taken as b1=a+d0*R

And b2 is taken as b2 = c – d0*R = a + d0*R^2

And if it is assumed that minimum is in point b2, region (b1-c) will be thrown. Our new region will be d1=d0*R olacağindan

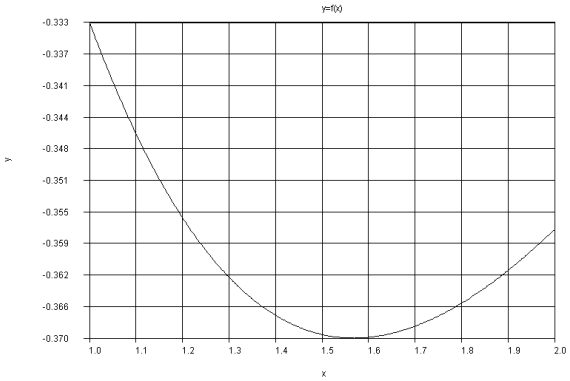
new b1 is taken as b1=a+d1*R=a+d0*R^2 which is the same as previous b2.

New b2 is taken n as b2 = c – d1*R = a + d1*R^2 = a + d0*R^3 .

PROGRAM 5.2-1A Golden Ratio or Fibonnachi search by excel

a	b	h	c	d	yc	yd
1	2	1	1.381966	1.618034	-0.366410476	-0.369427868
1.381966011	2	0.618034	1.618034	1.763932	-0.369427868	-0.366711114
1.381966011	1.763932	0.381966	1.527864	1.618034	-0.36949646	-0.369427868
1.381966011	1.618034	0.236068	1.472136	1.527864	-0.368819066	-0.36949646
1.472135955	1.618034	0.145898	1.527864	1.562306	-0.36949646	-0.369632208
1.527864045	1.618034	0.09017	1.562306	1.583592	-0.369632208	-0.369614176
1.527864045	1.583592	0.055728	1.54915	1.562306	-0.36960486	-0.369632208
1.549150281	1.583592	0.034442	1.562306	1.570437	-0.369632208	-0.369634304
1.562305899	1.583592	0.021286	1.570437	1.575462	-0.369634304	-0.369630023
1.562305899	1.575462	0.013156	1.567331	1.570437	-0.369634826	-0.369634304
1.562305899	1.570437	0.008131	1.565412	1.567331	-0.369634332	-0.369634826
1.565411519	1.570437	0.005025	1.567331	1.568517	-0.369634826	-0.369634819
1.565411519	1.568517	0.003106	1.566598	1.567331	-0.369634711	-0.369634826
1.56659776	1.568517	0.001919	1.567331	1.567784	-0.369634826	-0.369634851
1.567330897	1.568517	0.001186	1.567784	1.568064	-0.369634851	-0.36963485
1.567330897	1.568064	0.000733	1.567611	1.567784	-0.369634846	-0.369634851

Graphic exit of the function



Since one of the point is the same, evaluation of only one new point is required for each iteration step. The Algorithm of Golden search is given in Program below

```
# Golden search optimisation method
# python version

from math import *

def f1(x):
  y=x*x-x+5;
  return y;

def golden(f,a,b):
  # golden search
  r1=(sqrt(5)-1)/2;
  r2=r1*r1;
  h=b-a;
  ya=f1(a);
  yb=f1(b);
  c=a+r2*h;
  d=a+r1*h;
  yc=f1(c);
  yd=f1(d);
  k=1;
  epsilon=1.0e-10;
  while abs(yb-ya) > epsilon:
    k=k+1;
    if yc<yd:
      b = d;
      yb = yd;
      d = c;
      yd = yc;
      h = b - a;
      c = a + r2 * h;
      yc = f1(c);
    else:
      a = c;
      ya = yc;
```

```
        c = d;
        yc = yd;
        h = b - a;
        d = a + r1 * h;
        yd = f1(d);
  p=a;
  yp=ya;
  if yb<ya:
     p=b;
     yp=yb;
  return p;

a=float(input("enter lower limit : "));
b=float(input("enter upper limit : "));
r=golden(f1,a,b)
print("golden search optimisation  : ",r)
```

```
runfile('E:/okul/SCO1/golden.py', wdir='E:/okul/SCO1')

enter lower limit : 1

enter upper limit : 5
golden search optimisation  :  1.0
```

8.3 ONE DIMENSIONAL BISECTION METHOD

Optimum defined as root of so any root finding mathod can be employed to find optimum point by defining it as the root of the derivative of the function. Bisection method was specified above. The optimum point can be obtained by bisection method using derivative of the function

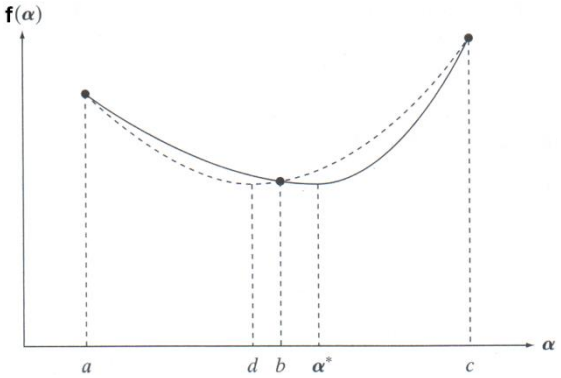
```
from math import *
from if_x import *;


def bisection_opt(f,xl,xu):
  # bisection root finding method
  maxit=100
  iter=0
  es=0.0000001
  ea=1.1*es
  s=""
  fxl= f.dfunc(xl,1)
  fxu= f.dfunc(xu,1)
  xr=0.0;
  while ea>es and iter<maxit:
     xold=xr
     xr=(xl+xu)/2.0;
     fxr= f.dfunc(xr,1);
     iter=iter+1;
     if xr!=0:
        ea=abs((xr-xold)/xr)*100
     test= fxl*fxr;
     if test==0.0 :      ea=0;
     elif test<0.0: xu=xr;fxu=fxr;
     elif test>0:    xl=xr;fxl=fxr;
     else :          ea=0;
     if iter>=maxit: print("Maximum number of iteration is exceeded \n result might not be valid")
  return xr

def optimum_test(f,x):
     # optimum test through second derivatine
     # x is maximum-minimum-extremum point
     y=f.dfunc(x,3)
     s=""
     if y==0.0 :  s="extremum"
     elif y>0.0:  s="minimum"
     else :       s="maximum"
     return s


class f1(if_x):func=lambda self,x:x*x-x+5
f=f1()
```

```
a=float(input("enter lower limit : "))
b=float(input("enter upper limit : "))
r=bisection_opt(f,a,b);
print(" optimum point  : ",r,"\nFUNCTION VALUE : ",f.func(r),optimum_test(f,r))
```

```
runfile('D:/okul/SCO1/bisection_opt.py', wdir='D:/okul/SCO1')
Reloaded modules: if_x

enter lower limit : 0.0

enter upper limit : 2.0
 optimum point  :  0.5000000004656613
FUNCTION VALUE :  4.75 maximum
```

8.4 ONE DIMENSIONAL NON-LINEAR QUADRATIC POLYNOMIAL METHOD

Golden search is a linear search method, it will always Works if a minimum(Maximum) is existed in the given region. But convergence speed is relatively slow. A quadratic search should theoretiacally be given values closer to the actual optimum value.



Minimum value of a quadratic equation is given with the Formula

$$d = b + 0.5\,\frac{(c-b)^2[f(a)-f(b)]-(a-b)^2[f(c)-f(a)]}{(c-b)[f(a)-f(b)]-(a-b)[f(c)-f(a)]}$$

After finding point d, point b is replaced with this point and iteration continues.

```
from math import *
from if_x import *

def opt_quadratic_poly(f,x0,x1,x2):
   maxit=100;
   epsilon=1.0e-10;
   delta=1.0e-5;
   f0 = f.func(x0)
   f1 = f.func(x1)
   f2 = f.func(x2)
   f3 = f2;
   x3 = 0;
   h = x1 - x0
   k=1
   dd=abs(f1-f0)
   while dd >epsilon or h>delta:
     k=k+1
     x3=(f0*(x1*x1-x2*x2)+f1*(x2*x2-x0*x0)+f2*(x0*x0-x1*x1))/(2*f0*(x1-x2)+2.0*f1*(x2-x0)+2.0*f2*(x0-x1))
     f3=f.func(x3);
     if x3 >= x0 and x3<x1:
         x2=x1
         f2=f1
         x1=x3
         f1=f3;
     elif x3 >= x1 and x3 <x2:
         x0=x1
         f0=f1
         x1=x3
         f1=f3
```

```
    elif x3 >x2 :
        x0=x1
        f0=f1
        x1=x2
        f1=f2
        x2=x3
        f2=f3
    elif x3 < x0 :
       x0=x3
       f0=f3
       x1=x0
       f1=f0
       x2=x1
       f2=f1
    dd=abs(f1-f0)
    h=abs(x1-x0)
    if k>maxit :
       break
   return x3;

class f1(if_x):func=lambda self,x:x*x-x+5
f=f1()
a=float(input("enter lower limit : "))
b=float(input("enter upper limit : "))
c=(a+b)/2.0;
r=opt_quadratic_poly(f,a,c,b);
print(" optimum point  : ",r,"\nFUNCTION VALUE : ",f.func(r))
```

```
runfile('D:/okul/SCO1/opt_quadratic_poly.py', wdir='D:/okul/SCO1')
Reloaded modules: if_x

enter lower limit : 0.0

enter upper limit : 2.0
 optimum point  :  0.5
FUNCTION VALUE :  4.75
```

8.5 HALLEY ONE DIMENSIONAL OPTIMISATION METHOD

Consider the Taylor Formula, but this time assume that two term is remaining

$$f(x_{n+1}) = 0 = f(x_n) + \frac{f'(x_n)}{1!}(x_{n+1} - x_n) + \frac{f''(x_n)}{2!}(x_{n+1} - x_n)^2 + \frac{f^{(3)}(x_n)}{3!}(x_{n+1} - x_n)^3 + \cdots \qquad (2.11.1)$$

When second term of the Taylor Formula is also taken into acount the Newton-Raphson Formula takes the form:

$$x_{n+1} = x_n - \frac{2f(x_n)f'(x_n)}{2f'(x_n)f'(x_n) - f(x_n)f''(x_n)}$$

This method is called **Halley's root finding method**.

In order to apply method to optimisation problems

Function will be replaced with its derivatives

$$x_{n+1} = x_n - \frac{2f'(x_n)f''(x_n)}{2f''(x_n)f''(x_n) - f'(x_n)f'''(x_n)}$$

```
from math import *
from if_x import *

def halley_opt(f,x):
    # halley optimisation method
    nmax=100
    tolerance=1.0e-10
    for i in range(0,nmax):
       fx=f.dfunc(x,1);
       dfx=f.dfunc(x,2)
       d2fx=f.dfunc(x,3)
       x=x-2.0*fx*dfx/(2.0*dfx*dfx-fx*d2fx)
       if abs(fx)<tolerance: return x
```

```
    return x

def optimum_test(f,x):
  # optimum test through second derivatine
  # x is maximum-minimum-extremum point
  y=f.dfunc(x,3)
  s=""
  if y==0.0 :  s="extremum"
  elif y>0.0:  s="minimum"
  else :      s="maximum"
  return s

class f1(if_x):func=lambda self,x:x*x-x+5
f=f1()
x=1.0
r=halley_opt(f,x)
print(" optimum point  : ",r,"\nFUNCTION VALUE : ",f.func(r),optimum_test(f,r));
```

```
runfile('D:/okul/SCO1/halley_opt.py', wdir='D:/okul/SCO1')
Reloaded modules: if_x
 optimum point  :  0.499999999998834
FUNCTION VALUE :  4.75 maximum
```

8.6 ONE DIMENSIONAL THIELE METHOD

Thiele Method was investigated in Root finding section 7.9. Due to fact that root of the derivatives is given optimisation, Thiele Method can be utilised as optimisation method as follows:

$$g(x) = \frac{df(x)}{dx} = f'(x)$$

$$x_{n+1} = x_n - \frac{P_1(x_n)}{Q_1(x_n)} \quad (7.9.5)$$

$$P_1(x) = 4g(x)g'(x)[3g'(x)g'(x)g"(x)\text{-}3g(x)g"(x)g"(x) + g(x)g'(x)g'''(x)] \quad (7.9.6)$$

$$Q_1(x) = 12g'(x)g'(x)g'(x)g'(x)g(x)\text{-}18g(x)g'(x)g'(x)g"(x)g"(x) + 3g(x)g(x)\ g"(x)g"(x)g"(x) + 4g(x)g'(x)g'(x)g'(x)g'''(x) \quad (7.9.7)$$

$$x_{n+1} = x_n - \frac{P_2(x_n)}{Q_2(x_n)} \quad (7.9.8)$$

$$P_2(x) = g(x)[6g'(x)g'(x)\text{-}3g(x)g"(x)g"(x) + 2g(x)g'(x)g'''(x)] \quad (7.9.9)$$

$$Q_2(x) = 2g'(x)\left[3g'(x)g'(x)g"(x) - 3g(x)g"(x)g"(x) + g(x)g'(x)g'''(x)\right] \quad (7.9.10)$$

```
# Thiele root finding method  for 1D optimisation

from math import *
from if_x import *

def Thiele_opt(f,x):
  #Thiele 1 method
  a=[0.0 for i in range(3)];
  nmax=500;
  tolerance=1.0e-10;
  gx=0;
  for i in range(0,nmax):
    gx=f.dfunc(x,1);
    dgx=f.dfunc(x,2);
    d2gx=f.dfunc(x,3);
    d3gx=f.dfunc(x,4);
    P1=4.0*gx*dgx*(3.0*dgx*dgx*d2gx-3.0*gx*d2gx*d2gx+gx*dgx*d3gx);
    Q1=12.0*dgx*dgx*dgx*dgx*d2gx-18.0*gx*dgx*dgx*d2gx*d2gx+3.0*gx*gx*d2gx*d2gx*d2gx+4.0*gx*dgx*dgx*dgx*d3gx;
    x-=P1/Q1;
    if abs(gx)<tolerance:
       return x;
  print("Maximum number of iteration is exceeded in Thiele 1\n");
  return x;


class f1(if_x):func=lambda self,x:x*x-x+5
f=f1()
x=1.0
```

```
r=Thiele_opt(f,x)
print(" optimum point  : ",r,"Function = ",f.func(r));
```

```
n [81]: runfile('D:/okul/SCO1/Thiele_opt.py', wdir='D:/okul/SCO1')
Reloaded modules: if_x
 optimum point  :  0.49999999999816586 Function =  4.75
```

8.7 ONE DIMENSIONAL HASAN METHODS

Hasan[50] proposed several higher order methods for root finding with relatively less function calculations. Hasan Method can be utilised as optimisation method as follows:

$$g(x) = \frac{df(x)}{dx} = f'(x)$$

$$x_{n+1} = x_n - \frac{g(x_n)}{g'\left(x_n - \frac{g(x_n)}{2g'(x_n)}\right)} \qquad (8.7.1)$$

$$x_{n+1} = x_n - \frac{2g(x_n)}{g'(x_n) + g'\left(x_n - \frac{g(x_n)}{2g'(x_n)}\right)} \qquad (8.7.2)$$

$$x_{n+1} = x_n - \frac{2f(x_n)}{f'\left(x_n - \alpha\frac{f(x_n)}{f'(x_n)}\right) + f'\left(x_n - \beta\frac{f(x_n)}{f'(x_n)}\right)} \quad where\ \ \alpha + \beta = 1 \qquad (8.7.3)$$

$$x_{n+1} = x_n - \frac{2g(x_n)}{g'(x_n) + \frac{1}{r}g'(x_n)log\left(\frac{g'(x_n)g'(x_n) - g(x_n)g''(x_n)}{g'(x_n)g'(x_n)}\right)} \qquad (8.7.4)$$

```
from math import *
from if_x import *

def hasan_opt(f,x):
    # finding real roots by using hasan method fourth order
    nmax=500;
    r=3;
    tolerance=1.0e-15;
    for iter in range(0,nmax):
      gx=f.dfunc(x,1);
      dgx=f.dfunc(x,2);
      d2gx=f.dfunc(x,3);
      x=x-gx/(dgx+1.0/r*dgx*log((dgx*dgx-gx*d2gx)/(dgx*dgx)));
      if abs(gx)<tolerance: return x;
    return x;


class f1(if_x):func=lambda self,x:x*x-x+5
f=f1()
x=1.0
r=hasan_opt(f,x)
print(" optimum point  : ",r,"\nFUNCTION VALUE : ",f.func(r));
```

```
runfile('D:/okul/SCO1/hasan_opt.py', wdir='D:/okul/SCO1')
Reloaded modules: if_x
 optimum point  :  0.5000000000009188
FUNCTION VALUE :  4.75
```

8.8 ONE DIMENSIONAL BRENT METHOD

It is seen that quadratic formula can used to find with formula

$$x = b + 0.5\frac{(b-a)^2[f(b) - f(a)] - (b-c)^2[f(b) - f(a)]}{(b-a)[f(b) - f(a)] - (b-c)[f(b) - f(c)]}$$

But this Formula can be reached to maximum instead of minimum as well. One more secure approximation could be to use Golden search to approach to the minimum point and then utilise the quadratic polynomial method. This combined method is called Brent method.

```
# Brent root finding method  for 1D optimisation

from math import *
from if_x import *

def brent_opt(f,a,b):
    #brent optimisation method
    maxit=500;
    iter=0;
    tol=1.0e-15;
    es=0.0000001;
    x1=a;f1= f.dfunc(x1,1)
    x2=b;f2= f.dfunc(x2,1)
    x3=(x1+x2)/2.0;f3= f.dfunc(x3,1)
    if f1==0:    return x1
    elif f2==0:  return x2
    elif f3==0:  return x3
    if f1*f2>0: print("No root is existed in the given region");
    p=-(f2*f3*x1*(f2-f3)+f3*f1*x2*(f3-f1)+f1*f2*x3*(f1-f2))/((f1-f2)*(f2-f3)*(f3-f1));
    fp= f.dfunc(p,1)
    ea=abs(x3-p);
    while (ea>es) and (iter<maxit):
      if abs(f3)<tol: return x3;
      if (p<x1) and (p>x2):
        p=(x1+x2)/2.0;
        if p>x3:   x1=x3;f1=f3;x3=p;f3=fp;
        elif p<x3: x2=x3;f2=f3;x3=p;f3=fp;
      else:
        if p>x3: x1=x3;f1=f3;x3=p;f3=fp;
        elif p<x3: x2=x3;f2=f3;x3=p;f3=fp;
        p=-(f2*f3*x1*(f2-f3)+f3*f1*x2*(f3-f1)+f1*f2*x3*(f1-f2))/((f1-f2)*(f2-f3)*(f3-f1));
        fp= f.dfunc(p,1)
        ea=abs(x3-p);
      iter=iter+1;
    if iter>=maxit: print("Warning : Maximum number of iteration is exceeded");
    return p;


class f1(if_x):func=lambda self,x:x*x-x+5
f=f1()
x1=1.0
x2=3.0
r=brent_opt(f,x1,x2)
print(" optimum point  : ",r,"Function = ",f.func(r));
```

```
runfile('D:/okul/SCO1/brent_opt.py', wdir='D:/okul/SCO1')
Reloaded modules: if_x
No root is existed in the given region
 optimum point  :  0.4999999999997318 Function =  4.75
```

8.9 ONE DIMENSIONAL RIDDER METHOD

Ridder method root finding is a modified form False position method. If a root existed between x1 and x2 Ridder method first evaluate the midpoint

$$x_3 = \frac{x_1+x_2}{2} \qquad (8.9.1)$$

and than solve a special conversion conversion factor e^Q. This convertion linearize the equation. It is basically find the root of the quadratic function

$$f(x_1) - 2\,f(x_3)e^Q + f(x_1) - 2\,f(x_2)e^Q = 0 \qquad (8.9.2)$$

The root of this function can be calculated as
of the quadratic function

$$e^{Q} = \frac{f(x_3)+sign[f(x_2)]\sqrt{f(x_3)^2-f(x_1)f(x_2)}}{f(x_2)} \qquad (8.9.3)$$

Function sign(x) will be given the sign of x. If x is smaller than 0 it will be -1, if x is bigger than 0 it will be +1. Now the false position method is applied, not to the values f(x1), f(x3), f(x2), but to the values $f(x_1)$, $f(x_3)e^{Q}$, $f(x_2)e^{2Q}$, yielding a new guess for the root, x_4. The overall formula is:

$$x_4 = x_3 + (x_3 - x_1)\frac{f(x_3)+sign[f(x_1)-f(x_2)]f(x_3)}{\sqrt{f(x_3)^2-f(x_1)f(x_2)}} \qquad (8.9.4)$$

Because the equation is quadratic, it supplies a quadratic approximation. In addition the root x_4 should be in between x_1 and x_2.

In order to transfer root finding method as optimisation method, derivaative of function is used instead of the function.

$$g(x) = \frac{df(x)}{dx} = f'(x) \quad (8.9.5)$$

$$x_4 = x_3 + (x_3 - x_1)\frac{g(x_3)+sign[g(x_1)-g(x_2)]g(x_3)}{\sqrt{g(x_3)^2-g(x_1)g(x_2)}} \quad (8.9.6)$$

```
from math import *
from if_x import *
def SIGN(a,b):
    return abs(a)*b/abs(b)

def ridder_opt(f,x1,x2):
    # finding real roots by using Ridder method fourth order
    xacc=1.0e-9;
    MAXITER=200
    ABC=-1.0e30
    f_low=f.dfunc(x1,1);
    f_high=f.dfunc(x2,1);
    print("f_low",f_low)
    b1=f_low > 0.0 and f_high < 0.0;
    b2=f_low < 0.0 and f_high > 0.0;
    b3=b1 or b2;
    if b3 :
      x_low=x1;
      x_high=x2;
      answer=ABC;
      for j in range(0,MAXITER):
         x_mid=0.5*(x_low+x_high);
         f_mid=f.dfunc(x_mid,1);
         s=sqrt(f_mid*f_mid-f_low*f_high);
         if s == 0.0: return answer;
         if f_low>=f_high: xx=1;
         else: xx=-1;
         xnew=x_mid+(x_mid-x_low)*xx*f_mid/s;
         if abs(xnew-answer) <= xacc: return answer;
         answer=xnew;
         fnew=f.dfunc(answer,1);
         if fnew == 0.0: return answer;
         if SIGN(f_mid,fnew) != f_mid:x_low=x_mid;f_low=f_mid;x_high=answer;f_high=fnew;
         elif SIGN(f_low,fnew) != f_low:x_high=answer;f_high=fnew;
         elif SIGN(f_high,fnew) != f_high:x_low=answer;f_low=fnew;
         else: print("we should not reach to this point");
      if abs(x_high-x_low) <= xacc: return answer;
      print("Warning : Maximum number of iteration is exceeded \n");
    else:
       if f_low == 0.0: return x1;
       if f_high == 0.0: return x2;
       print("Warning : No roots existed between the given limits ");
    return 0.0
```

```
class f1(if_x):func=lambda self,x:x*x-x+5
f=f1()
x1=0.0
x2=3.0
r=ridder_opt(f,x1,x2)
print(" optimum point  : ",r,"\nFUNCTION VALUE : ",f.func(r));
```

```
runfile('D:/okul/SCO1/ridder_opt.py', wdir='D:/okul/SCO1')
Reloaded modules: if_x
f_low -0.9999999999956392
 optimum point  :  0.5000000000001316
FUNCTION VALUE :  4.75
```

8.10 ONE DIMENSIONAL OPTIMISATION MULLER

Werner Muller, German Mathematician

In false position method roots were found by using the root of the linear line passing through the two interval points. The same concept can be use to fit a quadratic curve and find the roots of the quadratic function. But quadratic function fitting is required 3 estimation point. The method consist of deriving the coefficients of a quadratic curve(parabola) that goes through three points. If function evaluation of these 3 points are $f(x_0)$, $f(x_1)$ and $f(x_2)$, enough information will be evailable to make a quadratic curve root estimation. Assuming quadratic function defined b the following equation :

$f(x) = a(x - x_2)^2 + b(x - x_2) + c$ (2.6.1)

Evaluation of the function in 3 given points will be

$f(x_0) = a(x_0 - x_2)^2 + b(x_0 - x_2) + c$ (2.6.2)

$f(x_1) = a(x_1 - x_2)^2 + b(x_1 - x_2) + c$ (2.6.3)

$f(x_2) = a(x_2 - x_2)^2 + b(x_2 - x_2) + c = c$ (2.6.4)

Considering the difference of the functions:

$f(x_0) - f(x_2) = a(x_0 - x_2)^2 + b(x_0 - x_2)$ (2.6.5)

$f(x_1) - f(x_2) = a(x_1 - x_2)^2 + b(x_1 - x_2)$ (2.6.6)

$h_0 = (x_1 - x_0)$ (2.6.7)

$h_1 = (x_2 - x_0)$ (2.6.8)

$d_0 = \frac{f(x_1) - f(x_0)}{h_0}$ (2.6.9)

$d_1 = \frac{f(x_2) - f(x_0)}{h_1}$ (2.6.10)

When these are substituted back into the main equation

$(h_0 + h_1)b - (h_0 + h_1)^2 a = h_0 d_0 + h_1 d_1$ (2.6.11)

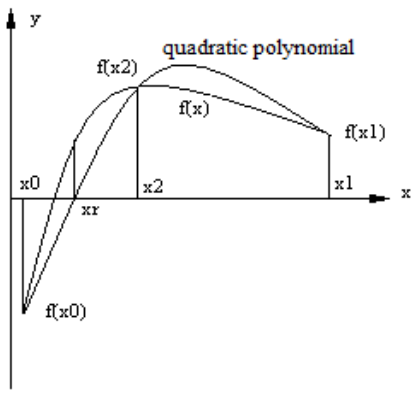
$h_1 b - (h_1)^2 a = h_1 d_1$ (2.6.12)

is obtained. a and b can be solved from here.

$a = \frac{d_1 - d_0}{h_1 - h_0}$ (2.6.13)

$b = a h_1 + d_1$ (2.6.14)

$c = f(x_{2)}$ (2.6.16)



graphic representation of Muller root finding method.

to find root root :

$\Delta = \sqrt{b^2 - 4ac}$ (2.6.17)

if

$|b + \Delta| > |b + \Delta| \quad then\ e = b + \Delta$ (2.6.18)

Else

$e = b + \Delta$ (2.6.19)

$x_r = x_2 - \frac{2c}{e}$ (2.6.20)

$x_0 = x_1 \quad x_1 = x_2 \quad x_2 = x_r$ (2.6.21)

x_2 will be substituted with x_r and iteration continues. It should be noted that in a quadratic formula complex roots can be located as well as the real roots, therefore this method can be used to find complex roots as well. In order to transfer root finding method as optimisation method, derivaative of function is used instead of the function.

$$g(x) = \frac{df(x)}{dx} = f'(x)$$

```
#muller optimisation method
from math import *
from if_x import  *
def muller_opt(f,x0,x1,x2):
    # Finding real roots by using Muller method
    iter=0;
    es1=0.001;
    es=es1;
    ea=1.1*es;
    maxit = 100;
    xr=x0;
    for iter in range(0,maxit):
      fx0=f.dfunc(x0,1);
      fx1=f.dfunc(x1,1);
      fx2=f.dfunc(x2,1);
      h0=x1-x0;
      h1=x2-x1;
      d0=(fx1-fx0)/h0;
      d1=(fx2-fx1)/h1;
      a=(d1-d0)/(h1+h0);
      b=a*h1+d1;
      c=fx2;
      determinant=b*b-4.0*a*c;
      if determinant<=0:  determinant=0;
      else: determinant=sqrt(determinant);
      if abs(b+determinant)>  abs(b-determinant): den=b+determinant;
      else: den=b-determinant;
      hr=-2*c/den;
      xr=x2+hr;
      ea=abs((xr-x2)/xr)*100;
      x0=x1;
      x1=x2;
```

```
      x2=xr;
      if ea<es: return xr
    if iter>=maxit:  print('Warning : Maximum number of iteration is exceeded result may not be valis');
    return xr

class f1(if_x):func=lambda self,x:x*x-x+5
f=f1()
x0=0.0
x1=1.0
x2=3.0
r=muller_opt(f,x0,x1,x2)
print(" optimum point  : ",r,"\nFUNCTION VALUE : ",f.func(r));
```

```
runfile('D:/okul/SCO1/muller_opt.py', wdir='D:/okul/SCO1')
Reloaded modules: if_x
 optimum point  :  0.49999999999799116
FUNCTION VALUE :  4.75
```

8.11 ONE DIMENSIONAL ABBASBANDY OPTIMISATION METHOD

In [48], Abbasbandy gives a third order iterative methodfor root finding , called **Abbasbandy's method**(AM) provisionally, which is expressed as

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} - \frac{1}{2}\frac{f(x_n)f(x_n)f''(x_n)}{f'(x_n)f'(x_n)f'(x_n)} - \frac{1}{6}\frac{f(x_n)f(x_n)f(x_n)f'''(x_n)}{f'(x_n)f'(x_n)f'(x_n)f'(x_n)} \qquad (8.11.1)$$

. In order to transfer root finding method as optimisation method, derivaative of function is used instead of the function.

$$g(x) = \frac{df(x)}{dx} = f'(x) \quad (8.11.2)$$

$$x_{n+1} = x_n - \frac{g(x_n)}{g'(x_n)} - \frac{1}{2}\frac{g(x_n)g(x_n)g''(x_n)}{g'(x_n)g'(x_n)g'(x_n)} - \frac{1}{6}\frac{g(x_n)g(x_n)g(x_n)g'''(x_n)}{g'(x_n)g'(x_n)g'(x_n)g'(x_n)} \quad (8.11.3)$$

```
# Abbasbandy root finding method  for 1D optimisation

from math import *
from if_x import *

def Abbasbandy_opt(f,x):
   #Thiele 1 method
   a=[0.0 for i in range(3)];
   nmax=500;
   tolerance=1.0e-10;
   gx=0;
   for i in range(0,nmax):
     gx=f.dfunc(x,1);
     dgx=f.dfunc(x,2);
     d2gx=f.dfunc(x,3);
     d3gx=f.dfunc(x,4);
     P1=0.5*gx*gx*d2gx
     Q1=dgx*dgx*dgx
     P2=1.0/6.0*gx*gx*gx*d3gx
     Q2=dgx*dgx*dgx*dgx
     x-=gx/dgx;
     x-=P1/Q1
     x-=P2/Q2
     if abs(gx)<tolerance:
        return x;
   print("Maximum number of iteration is exceeded in Abbasbandy \n");
   return x;


class f1(if_x):func=lambda self,x:x*x-x+5
f=f1()
x=1.2
r=Abbasbandy_opt(f,x)
print(" optimum point  : ",r,"Function = ",f.func(r));
```

```
runfile('D:/okul/SCO1/Abbasbandy_opt.py', wdir='D:/okul/SCO1')
Reloaded modules: if_x
```

```
optimum point  :  0.5000000000012949 Function =  4.75
```

8.12 FINDING MORE THAN ONE ONE DIMESIONAL OPTIMUM POINT

When more than one optimum point is searched, one of the method is to look at derivative of roots, as carried out in root seach method.

Algorithm for root searching:

```
Set search limit dx, search and region a,b  (a is starting point, b is end point of the search)
x1=a
x2=a+dx
while x2<b
  if df(x1)/dx*df(x2)/dx less than 0 then
      root(optimum point) existed between x1 and x2
      set root(optimum point)  limists as x1 and x2
      x1=x2;
      x2=x1+dx
   if not
     x2=x2+dx
   end of if block
end of while block
return x1,x2 sets as root limits
```

After achieving root limits, and finding optimum points by some method such as bisection or secant, the values can be evaluated by checking second derivative to see if it is a minimum, maximum or extremum.

```
# -*- coding: utf-8 -*-
"""
Created on Sat Dec 18 07:06:10 2021

@author: M. Turhan ÇOBAN
"""
# function f(x) should be defined by using class if_x

from math import *
from if_x import *
import numpy as np
import matplotlib.pyplot as plt

def opt_limits(f,a,b):
    # find root limits of multiple roots
    # in between a and b
    dx1=1e-2
    x1=a
    x2=a+dx1
    f1=f.dfunc(x1,1)
    f2=f.dfunc(x2,1)
    c = [[0.0 for j in range(2)] for i in range(200)]
    n=0
    while x2<b:
        if f1*f2<=0:
            c[n][0]=x1
            c[n][1]=x2
            n=n+1
            x1=x2+dx1
            x2=x1+dx1
            f1=f.dfunc(x1,1)
            f2=f.dfunc(x2,1)
        else:
            x2=x2+dx1
            f2=f.dfunc(x2,1)
    e=[[0 for x in range(2)] for y in range(n)]
```

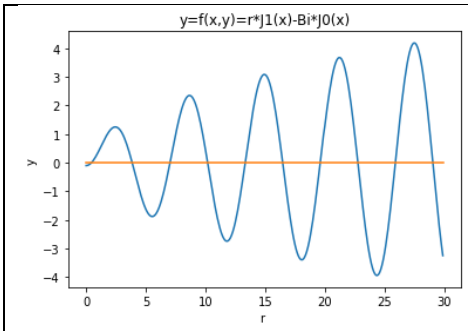
```
    for i in range(0,n):
        e[i]=c[i]
    return e

def minimum_bisection(f,a,b):
    c=opt_limits(f,a,b)
    n=len(c)
    print("c=",c,"n=",n)
    x=[0.0 for x in range(n)]
    y=[0.0 for x in range(n)]
    n1=0
    for i in range(n):
      x[i]=bisection(f,c[i][0],c[i][1])
      if f.dfunc(x[i],2)>0:
          y[n1]=x[i]
          n1=n1+1
    z=[0.0 for x in range(n1)]
    for i in range(n1):
        z[i]=y[i];
    return z

def optimum_bisection(f,a,b):
    c=opt_limits(f,a,b)
    n=len(c)
    x=[0.0 for x in range(n)]
    for i in range(n):
      x[i]=bisection(f,c[i][0],c[i][1])
    return x

def optimum_secant(f,a,b):
    c=opt_limits(f,a,b)
    n=len(c)
    x=[0.0 for x in range(n)]
    for i in range(n):
      x[i]=secant(f,c[i][1])
    return x

def minimum_secant(f,a,b):
    c=opt_limits(f,a,b)
    n=len(c)
    x=[0.0 for x in range(n)]
    y=[0.0 for x in range(n)]
    n1=0
    for i in range(n):
      x[i]=secant(f,c[i][1])
      ff1=f.dfunc(x[i],2)
      cf=f.dfunc(x[i],2)>0
      if cf:
          y[n1]=x[i]
          n1=n1+1
    z=[0.0 for x in range(n1)]
    for i in range(n1):
        z[i]=y[i];
    return z

def bisection(f,x1,x2):
   # bisection optimisation  method
   f1=f.dfunc(x1,1)
   r=(x1+x2)/2.0
   fr=f.dfunc(r,1)
   eps=1.0e-6
   Maxiter=100
   i=0
   while abs(fr)>eps and i<Maxiter:
      if f1==0:
          return x1
      elif fr==0:
          return r
      elif f1*fr<0:
          x2=r
      else:
          x1=r
          f1=fr
      r=(x1+x2)/2.0
      fr=f.dfunc(r,1)
      i=i+1
```

```
    return r


def secant(f,x):
  # secant optimisation method
  nmax=100
  tolerance=1.0e-10
  for i in range(0,nmax):
      fx= f.dfunc(x,1);
      dfx=f.dfunc(x,2)
      x=x-fx/dfx
      if abs(fx)<tolerance: return x
  return x



def J(v,x):
    JJ=1e-6
    a=1.0
    fact=1
    b=0.0
    if x <=20:
        x5=1
        lna=0.0;
        x1=v*log(0.5*x)
        lna1=log(0.25*x*x)
        for k in range(0,100):
            x2=lna
            x3=lgamma(k+1)
            x4=lgamma(v+k+1)
            b=x1+x2-x3-x4
            b=x5*exp(b)
            lna=lna+lna1
            x5=-1*x5
            JJ=JJ+b
    elif (v==0 and x==0):
        JJ=1
    else:
        JJ=sqrt(2.0/pi/x)*cos(x-0.5*v*pi-0.25*pi)
    return JJ


Bi=0.1
class f1(if_x):func=lambda self,r:r*J(1,r)-Bi*J(0,r)
f = f1()
a=0.001
b=30
r1=optimum_bisection(f,a,b);
print("optimum Bisection r=",r1)
r2=optimum_secant(f,a,b);
print("Secant r=",r2)
r3=minimum_bisection(f,a,b);
print("minimum Bisection r=",r3)
r4=minimum_secant(f,a,b);
print("minimum secant r=",r4)
x = np.arange(a, b, 0.1);
y1=0.0*x
n=len(x);
y=[0 for i in range(n)]
for i in range(n):
  y[i] = f.func(x[i])
plt.title('y=f(x,y)=r*J1(x)-Bi*J0(x)')
plt.xlabel('r ')
plt.ylabel('y ')
plt.plot(x, y,x,y1)
```

```
from abc import ABC, abstractmethod
from math import *

class if_x(ABC):

    @abstractmethod
    def func(self,x):
        pass;

    def dfunc(self,x,n):
```

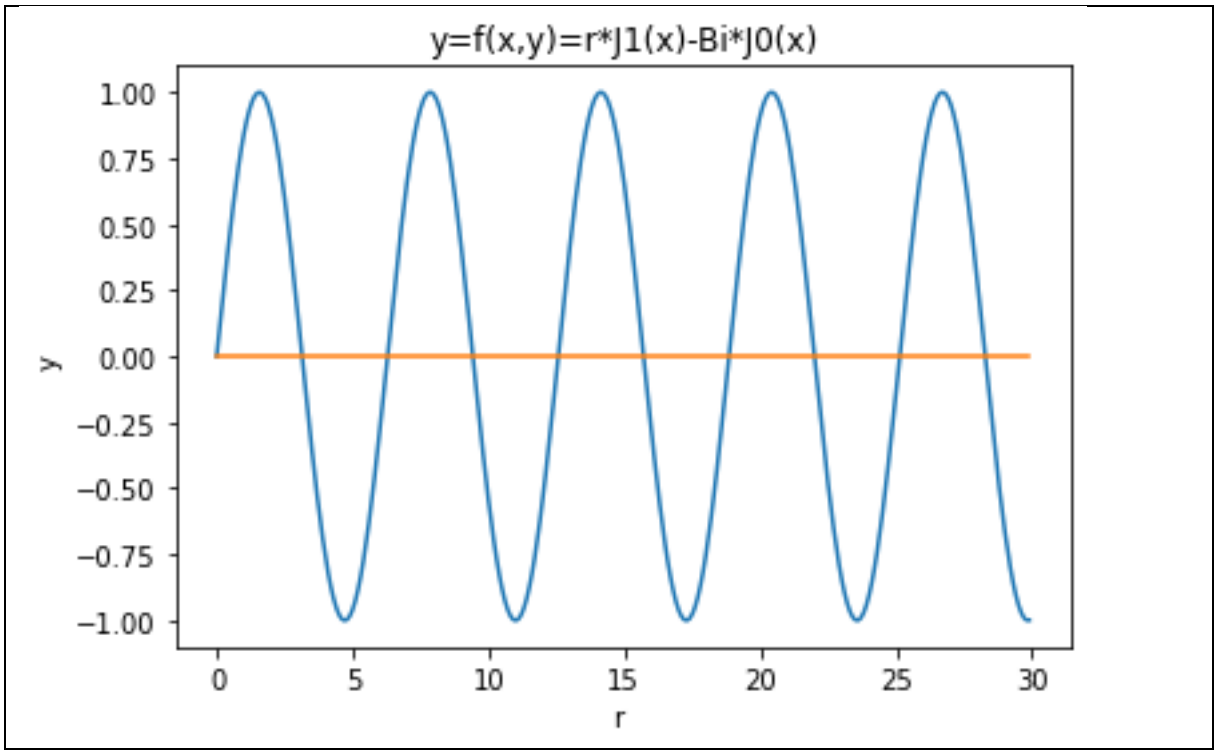
```
        dx=0.0001;
        if n==0: df=self.func(x);
        elif n==1: df=(3.0*self.func(x-4.0*dx)-32.0*self.func(x-3.0*dx)+168.0*self.func(x-2.0*dx)-672.0*self.func(x-
dx)+672.0*self.func(x+dx)-168.0*self.func(x+2.0*dx)+32.0*self.func(x+3.0*dx)-3.0*self.func(x+4.0*dx))/840.0/dx;
        elif n==2: dx=0.001;df=(-14350.0*self.func(x)-9.0*self.func(x-4.0*dx)+128.0*self.func(x-3.0*dx)-1008.0*self.func(x-
2.0*dx)+8064.0*self.func(x-dx)+8064.0*self.func(x+dx)-1008.0*self.func(x+2.0*dx)+128.0*self.func(x+3.0*dx)-
9.0*self.func(x+4.0*dx))/5040.0/dx/dx;
        elif n==3: dx=0.001;df=(-7.0*self.func(x-4.0*dx)+72.0*self.func(x-3.0*dx)-338.0*self.func(x-2.0*dx)+488.0*self.func(x-dx)-
488.0*self.func(x+dx)+338.0*self.func(x+2.0*dx)-72.0*self.func(x+3.0*dx)+7.0*self.func(x+4.0*dx))/240.0/dx/dx/dx;
        elif n==4: dx=0.001;df=(2730.0*self.func(x)+7.0*self.func(x-4.0*dx)-96.0*self.func(x-3.0*dx)+676.0*self.func(x-2*dx)-
1952.0*self.func(x-dx)-1952.0*self.func(x+dx)+676.0*self.func(x+2.0*dx)-
96.0*self.func(x+3.0*dx)+7.0*self.func(x+4.0*dx))/240.0/dx/dx/dx/dx;
        elif n==5: dx=0.01;df=(self.func(x-4.0*dx)-9.0*self.func(x-3.0*dx)+26.0*self.func(x-2.0*dx)-29.0*self.func(x-
dx)+29.0*self.func(x+dx)-26.0*self.func(x+2.0*dx)+9.0*self.func(x+3.0*dx)-self.func(x+4.0*dx))/6.0/dx/dx/dx/dx/dx;
        elif n==6: dx=0.01;df=(-150.0*self.func(x)-self.func(x-4.0*dx)+12.0*self.func(x-3.0*dx)-52.0*self.func(x-
2.0*dx)+116.0*self.func(x-dx)+116.0*self.func(x+dx)-52.0*self.func(x+2.0*dx)+12.0*self.func(x+3.0*dx)-
self.func(x+4.0*dx))/4.0/dx/dx/dx/dx/dx/dx;
        elif n==7: dx=0.01;df=(-self.func(x-4.0*dx)+6.0*self.func(x-3.0*dx)-14.0*self.func(x-2.0*dx)+14.0*self.func(x-dx)-
14.0*self.func(x+dx)+14.0*self.func(x+2.0*dx)-6.0*self.func(x+3.0*dx)+self.func(x+4.0*dx))/2.0/dx/dx/dx/dx/dx/dx/dx;
        elif n==8: dx=0.1;df=(70.0*self.func(x)+self.func(x-4.0*dx)-8.0*self.func(x-3.0*dx)+28.0*self.func(x-2.0*dx)-56.0*self.func(x-
dx)-56.0*self.func(x+dx)+28.0*self.func(x+2.0*dx)-8.0*self.func(x+3.0*dx)+self.func(x+4.0*dx))/dx/dx/dx/dx/dx/dx/dx/dx;
        else: df=0;
        return df;

    def gauss_legendre_coefficients(self,x1,x2,n):
        #calculates legendre gauss-coefficients as coefficients of the integral
        #for n terms
        eps=3e-15
        m=int((n+1.0)/2.0)
        xm=0.5*(x2+x1)
        xl=0.5*(x2-x1)
        nmax=100
        a=[[0.0 for i in range(n)] for j in range(2)]
        for i in range(1,m+1):
            z=cos(pi*((i-0.25)/(n+0.5)))
            for ii in range(nmax):
                #while abs(z-z1) > eps:
                p1=1.0;
                p2=0.0;
                for j in range(1,n+1):
                    p3=p2
                    p2=p1
                    p1=(float)(((2.0*j-1.0)*z*p2-(j-1.0)*p3)/j)
                pp=(float)(n*(z*p1-p2)/(z*z-1.0))
                z1=z;
                z=z1-p1/pp
                if abs(z-z1) < eps: break
            a[0][i-1]=xm-xl*z
            a[0][n-i]=xm+xl*z
            a[1][i-1]=2.0*xl/((1.0-z*z)*pp*pp)
            a[1][n-i]=a[1][i-1]
        return a

    def integral(self,x1,x2,n):
        #integral func(x)dx
        #integral of a function by using gauss-legendre quadrature
        #between x1 and x2
        a=self.gauss_legendre_coefficients(x1,x2,n)
        z=0
        for i in range(n):
            z=z+a[1][i]*self.func(a[0][i])
        return z
```

```
runfile('D:/okul/SCO1/opt_limits.py', wdir='D:/okul/SCO1')
Reloaded modules: if_x
optimum Bisection r= [2.445357347488395, 5.538103071212696, 8.665260330200052, 11.800005079269202, 14.937616037861858,
18.07652464455369, 21.233996376753375, 24.371954701901494, 27.51074039077909]
Secant r= [2.4453579083832677, 5.538102946958073, 8.665260475482734, 11.800005693167916, 14.93758758667544,
18.076444632215296, 21.233996338889188, 24.371954483328352, 27.510740485604636]
c= [[0.001, 2.4509999999999916], [2.4609999999999914, 5.540999999999926], [5.5509999999999255, 8.670999999999859],
[8.680999999999859, 11.800999999999792], [11.810999999999792, 14.940999999999725], [14.950999999999725,
18.081000000000028], [18.09100000000003, 21.241000000000522], [21.251000000000523, 24.381000000001013],
[24.391000000001014, 27.511000000001502]] n= 9
minimum Bisection r= [5.538103071212696, 11.800005079269202, 18.07652464455369, 24.371954701901494]
minimum secant r= [5.538102946958073, 11.800005693167916, 18.076444632215296, 24.371954483328352]
```

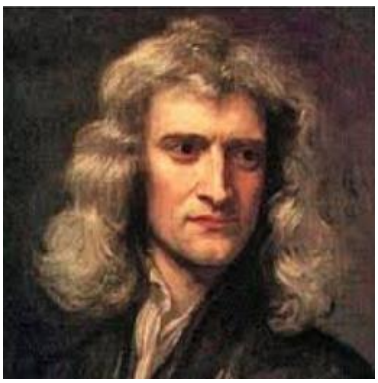



```
Bi=0.1
class f1(if_x):func=lambda self,x:sin(x)
f = f1()
a=0.0
b=30
r1=optimum_bisection(f,a,b);
print("optimum Bisection r=",r1)
r2=optimum_secant(f,a,b);
print("Secant r=",r2)
r3=minimum_bisection(f,a,b);
print("minimum Bisection r=",r3)
r4=minimum_secant(f,a,b);
print("minimum secant r=",r4)
x = np.arange(a, b, 0.1);
y1=0.0*x
n=len(x);
y=[0 for i in range(n)]
for i in range(n):
  y[i] = f.func(x[i])
plt.title('y=f(x,y)=r*J1(x)-Bi*J0(x)')
plt.xlabel('r ')
plt.ylabel('y ')
plt.plot(x, y,x,y1)
```

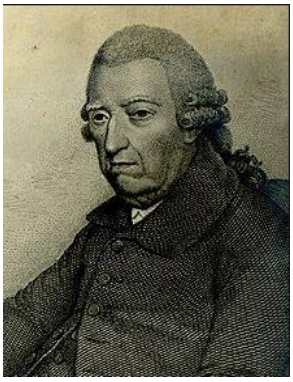
```
runfile('D:/okul/SCO1/opt_limits1.py', wdir='D:/okul/SCO1')
Reloaded modules: if_x
optimum Bisection r= [1.5707964324951182, 4.712388248443547, 7.853982238769408, 10.995574736594964, 14.13716723442052,
17.2787597322463, 20.420351600647365, 23.56194497585385, 26.703537473679965, 29.845129971506083]
Secant r= [1.5707963267944889, 4.712388980384815, 7.853981633976002, 10.995574287565736, 14.137166941153724,
17.278759594744823, 20.420352248333145, 23.561944901923574, 26.70353755551332, 29.845130209103147]
c= [[0.0, 1.5800000000000012], [1.5900000000000012, 4.719999999999944], [4.729999999999944, 7.859999999999877],
[7.869999999999877, 10.99999999999981], [11.00999999999981, 14.139999999999743], [14.149999999999743, 17.2799999999999],
[17.289999999999903, 20.430000000000394], [20.440000000000396, 23.570000000000885], [23.580000000000886,
26.710000000001376], [26.720000000001377, 29.850000000001867]] n= 10
minimum Bisection r= [4.712388248443547, 10.995574736594964, 17.2787597322463, 23.56194497585385, 29.845129971506083]
minimum secant r= [4.712388980384815, 10.995574287565736, 17.278759594744823, 23.561944901923574, 29.845130209103147]
```



8.13 MULTIDIMENSIONAL NEWTON-RAPHSON OPTIMISATION METHOD



Sir Isaac Newton



Joseph Raphson

Root finding of multivariable functions by using Newton-Raphson method was prevously investigated. By remembering that optimum point is the root of the derivative of the function, the method can be utilise also to find optimum point.

If function $f(x_1,x_2,x_3,\ldots,x_n)$ First and second derivative and independent difference vector of the function can be written as:

$$\nabla f = \begin{Bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \frac{\partial f}{\partial x_3} \\ \cdots\cdots \\ \frac{\partial f}{\partial x_n} \end{Bmatrix} \text{ and } \delta^{m+1} = \begin{Bmatrix} (x_1^{m+1} - x_1^m) \\ (x_2^{m+1} - x_2^m) \\ (x_3^{m+1} - x_3^m) \\ \cdots\cdots \\ (x_n^{m+1} - x_n^m) \end{Bmatrix} \text{ and } [H] = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \frac{\partial^2 f}{\partial x_1 \partial x_3} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \frac{\partial^2 f}{\partial x_2 \partial x_3} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \frac{\partial^2 f}{\partial x_3 \partial x_1} & \frac{\partial^2 f}{\partial x_3 \partial x_2} & \frac{\partial^2 f}{\partial x_3^2} & \cdots & \frac{\partial^2 f}{\partial x_3 \partial x_n} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \frac{\partial^2 f}{\partial x_1 \partial x_n} & \frac{\partial^2 f}{\partial x_2 \partial x_n} & \frac{\partial^2 f}{\partial x_3 \partial x_n} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

Newton-Raphson iteration for (m+1) th iteration

$[H^m]\{\delta^{m+1}\} = -\{\nabla f^m\}$

As it is seen from the equation a linear system of equation is formed. The equation can be solved by using methods such as gauss elimination. An initial estimate for all the x values are required to start iterative solution. Second derivartive matrix is also called a Hessian matrix. First and second derivatives can also be calculated numerically, but that will increase the errors orthe number of iteration steps. The system of equation can also be written as:

$\{\delta^{m+1}\} = [H^m]^{-1}[-\{\nabla f^m\}]$

In this format$[H^m]^{-1}$ is the inverse Hessian matrix. If vector δ is written in open form, it becomes

$\{x^{m+1}\} = \{x^m\} + [H^m]^{-1}[-\{\nabla f^m\}]$

As a more general definition of this equation a step function α can be added into the term. It becomes

$\{x^{m+1}\} = \{x^m\} + \alpha^m[H^m]^{-1}[-\{\nabla f^m\}]$

In Newton-Raphson method step function α is equal to unity. In some other methods different utilisation of this general description will be invetigated.

Program Abstract class f_xj for function y=f(x0,x1,x2,..) (Python version)

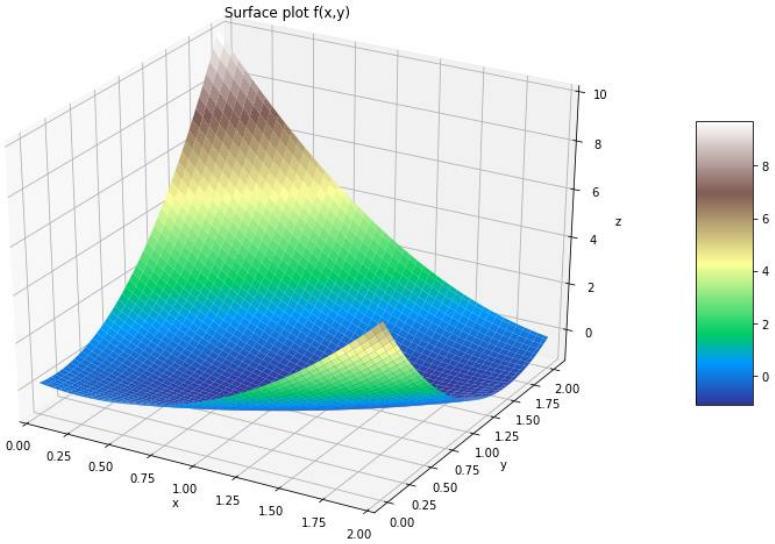
```
from abc import ABC, abstractmethod
from math import *

class f_xj(ABC):
  # multifunction multi independent variable
  # vector of dependent variables are returned
  # example f[0]=x[0]+sin(x[1])
  #         f[1]=x[0]*x[1]-x[1]
  # func(x) returns the value of f[0] and f[1]
  # as a two dimensional vector

  @abstractmethod
  def func(self,x):
    pass;

  def funcxy(self,x1,y1):
    n=2;
    x=[0 for i in range(n)];
    x[0]=x1;
    x[1]=y1;
    return self.func(x);

  def dfunc_(self,x,x_ref):
     #derivative of the function with respect to x_ref
     h0=0.015675863;
     m=7;
     n=len(x);
     x1=[0 for i in range(n)];
     x2=[0 for i in range(n)];
     for i in range(n):
        x1[i]=x[i];
        x2[i]=x[i];
     #derivative of a simple function
     T=[[0 for i2 in range(m)] for j2 in range(m)];
     h=[0 for i in range(m)];
     for i in range(m):
        h[i]=0;
        for j in range(m):
           T[i][j]=0;
```

```
    h[0]=h0;
    r=0.5;
    for i in range(m):
        h[i]=h0*pow(r,i);
    for i in range(m):
        x1[x_ref]+=h[i];
        x2[x_ref]-=h[i];
        f1=self.func(x1);
        f2=self.func(x2);
        T[i][0]=( f1 - f2)/(2.0*h[i]);
        x1[x_ref]=x[x_ref];
        x2[x_ref]=x[x_ref];
    for k in range(1,m):
         for i in range(0,m-k):
             T[i][k]=(h[i]*h[i]*T[i+1][k-1] - h[i+k]*h[i+k]*T[i][k-1])/(h[i]*h[i]
             - h[i+k]*h[i+k]);
    xx=T[0][n-1];
    return xx;

def d2func_(self,x,x_ref):
    #derivative of the function with respect to x_ref
    h0=0.015675863;
    m=7;
    n=len(x);
    x1=[0 for i in range(n)];
    x2=[0 for i in range(n)];
    for i in range(n):
        x1[i]=x[i];
        x2[i]=x[i];
    #derivative of a simple function
    T=[[0 for i2 in range(m)] for j2 in range(m)];
    h=[0 for i in range(m)];
    for i in range(m):
        h[i]=0;
        for j in range(m):
            T[i][j]=0;
    h[0]=h0;
    r=0.5;
    for i in range(m):
        h[i]=h0*pow(r,i);
    for i in range(m):
        x1[x_ref[1]]+=h[i];
        x2[x_ref[1]]-=h[i];
        f1=self.dfunc_(x1,x_ref[0]);
        f2=self.dfunc_(x2,x_ref[0]);
        T[i][0]=( f1 - f2)/(2.0*h[i]);
        x1[x_ref[1]]=x[x_ref[1]];
        x2[x_ref[1]]=x[x_ref[1]];
    for k in range(1,m):
         for i in range(0,m-k):
             T[i][k]=(h[i]*h[i]*T[i+1][k-1] - h[i+k]*h[i+k]*T[i][k-1])/(h[i]*h[i]
             - h[i+k]*h[i+k]);
    xx=T[0][n-1];
    return xx;

def dfunc(self,x):
    n=len(x);
    c=[0 for i in range(n)];
    for i in range(n):
        c[i]=self.dfunc_(x,i);
    return c;

def d2func(self,x):
    n=len(x);
    x_ref=[0 for i2 in range(2)];
    c=[[0 for i2 in range(n)] for j2 in range(n)];
    for i in range(n):
        for j in range(n):
            x_ref[0]=i;
            x_ref[1]=j;
            c[i][j]=self.d2func_(x,x_ref);
    return c;
```

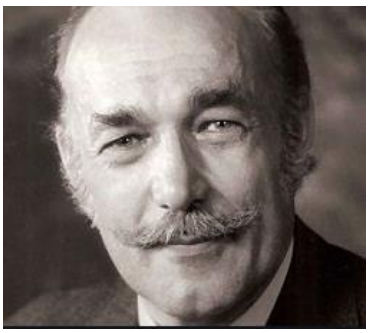
Program 5.8-4 Multivariable non-linear Newton-Raphson method (Python version)

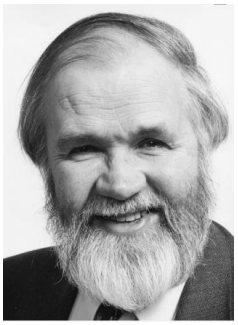
```
 from math import *
from f_xj import *;
from gauss import *;

class f1(f_xj):
  def func(self,x):
      y=3.0*x[0]*x[0]-4.0*x[0]*x[1]+2.0*x[1]*x[1]-x[0]-x[1];
      return y;

def newton(f,x):
   # Newton-Raphson root finding method
   k=len(x);
   nmax=100;
   tolerance=1.0e-10
   for i in range(nmax):
      fx= f.dfunc(x);
      dfx=f.d2func(x);
      dx=gauss(dfx,fx);
      for j in range(k):
          x[j]-=dx[j];
      total=0;
      for j in range(k):
          total+=fx[j];
      if abs(total)<tolerance: return x
   return x;

x=[1,1];
f=f1();
r=newton(f,x);
print("r=",r);
```

```
runfile('D:/okul/SCO1/newtond_opt.py', wdir='D:/okul/SCO1')
Reloaded modules: f_xj, gauss
r= [0.9999999999999799, 1.2499999999999776]
```



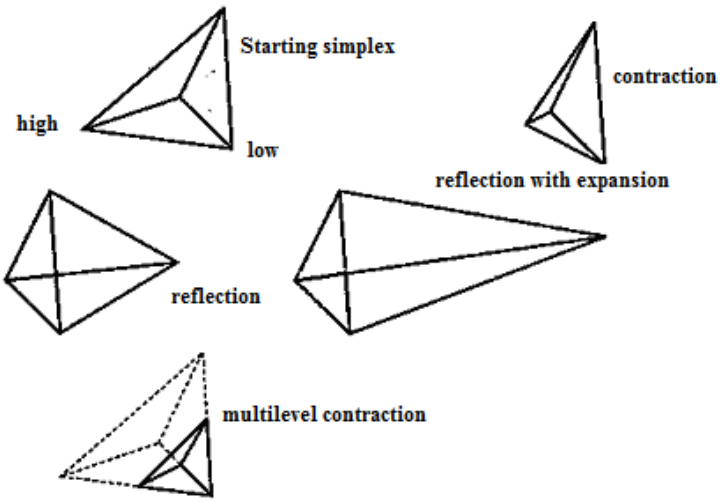
8.13 NELDER – MEAD SIMPLEX NONLINEAR MULTIVARIABLE OPTIMIZATION METHOD





John Nelder Roger Mead

The simplex method developed by Nelder and Mead method is a multidimensional search method . Simplex is an n dimensional geometric entity. It contains (n+1) points in n dimensional space. The method uses the concept of a simplex, which is a special polytop of $N + 1$ vertices in N dimensions. Examples of simplices include a line segment on a line, a triangle on a plane, a tetrahedron in three-dimensional space and so forth. Required processes for the simplex amobea's movements are shown in the figure



Process can be given as follows

- **1. Order** according to the values of the vertices:

  $\boldsymbol{f(x_1) \leq f(x_2) \leq \cdots \leq f(x_{n+1})}$

- **2. Calculate** x_0, the center of gravity off all points except x_{n+1}

- **3. Reflection:** Compute reflected point $x_r = x_0 + \alpha(x_0 - x_{n+1})$

If the reflected point is better than the second worst, but not better than the best, i.e.:
$\boldsymbol{f(x_1) \leq f(x_2) \leq f(x_n)}$
then obtain a new simplex by replacing the worst point x_{n+1} with the reflected point x_r, and go to step1

- **4. Expansion:**

If the reflected point is the best point so far, $\boldsymbol{f(x_r) < f(x_1)}$ **Then** compute the expanded point
$x_e = x_0 + \gamma(x_0 - x_{n+1})$
If the expanded point is better than the reflected point, $\boldsymbol{f(x_e) < f(x_r)}$ **then** obtain a new simplex by replacing the worst point x_{n+1} with the expanded point x_e, and go to step 1.
Else obtain a new simplex by replacing the worst point x_{n+1} with the reflected point x_r, and go to step 1.
Else (i.e. reflected point is worse than second worst) continue at step 5.

- **5. Contraction:**

Here, it is certain that , $\boldsymbol{f(x_r) < f(x_n)}$
Compute contracted point $x_c = x_{n+1} + \rho(x_0 - x_{n+1})$
If the contracted point is better than the worst point, i.e. $\boldsymbol{f(x_c) < f(x_{n+1})}$
then obtain a new simplex by replacing the worst point x_{n+1} with the contracted point x_c, and go to step 1.
Else go to step 6.

- **6. Reduction:**

For all but the best point with

$x_i = x_1 + \sigma(x_i - x_1)$ for all $i \in \{2, \dots, n+1\}$ goto step 1

Note: $\boldsymbol{\alpha}, \boldsymbol{\gamma}, \boldsymbol{\rho}$ and σ are respectively the reflection, the expansion, the contraction and the shrink coefficient. Standard values are $\boldsymbol{\alpha = 1, \gamma = 2, \rho = 1/2}$ and $\boldsymbol{\sigma = 1/2}$

For the **reflection**, since x_{n+1} is the vertex with the higher associated value among the vertices, we can expect to find a lower value at the reflection of x_{n+1} in the opposite face formed by all vertices point x_i except x_{n+1}.

For the **expansion**, if the reflection point x_r is the new minimum along the vertices we can expect to find interesting values along the direction from x_o to x_r.

Concerning the **contraction**: If $f(x_r) > f(x_n)$ we can expect that a better value will be inside the simplex formed by all the vertices x_i.

The initial simplex is important, indeed, a too small initial simplex can lead to a local search, consequently the NM can get more easily stuck. So this simplex should depend on the nature of the problem. Instead of given n+1 point same process can be done by giving one point and a change vector by defining

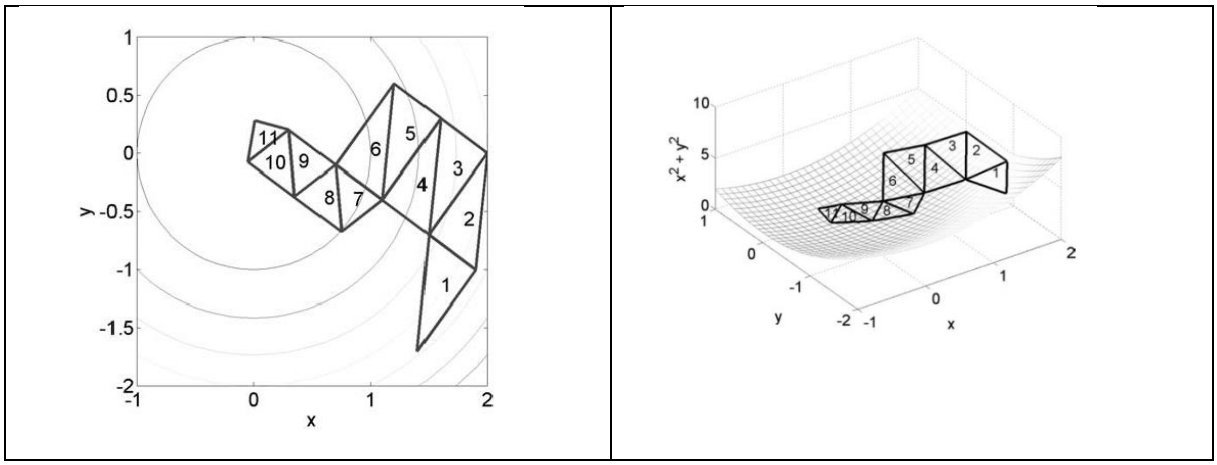
$x_{i-1} = P_i + \lambda\, dx_j$ where if i=j λ=1 else if i ≠ j λ=0 .

For example:

if $P = \begin{Bmatrix} a \\ b \\ c \end{Bmatrix}$ and $dx = \begin{Bmatrix} da \\ db \\ dc \end{Bmatrix}$

$$x_0 = \begin{Bmatrix} a \\ b \\ c \end{Bmatrix} \quad x_1 = \begin{Bmatrix} a + da \\ b \\ c \end{Bmatrix} \quad x_2 = \begin{Bmatrix} a \\ b + db \\ c \end{Bmatrix} \quad x_3 = \begin{Bmatrix} a \\ b \\ c + dc \end{Bmatrix}$$

Figure 6.2-1 Movements of the nelder-Mead simplex element to find the optimum point




```
from gauss import *;
from math import *;
from f_xj import *;
import numpy as np;
import matplotlib.pyplot as plt;
from mpl_toolkits.mplot3d import Axes3D
from random import *;

class f1(f_xj):
  def func(self,x):
      x1=x[0]-2.0
      x2=x[1]-1.0
      x3=x[2]-1.0
      y=x1*x1*x1*x1+x2*x2+3.0*x3*x3*x3*x3*x3*x3
      return y;
```

```
def  nelder(fnelder,a,da,maxiteration,tolerance,printlist):
    #print("a",a)
    #print("da",da)
    n=len(a)
    m=n+1
    x=[[0.0 for j in range(n)] for i in range(n+1)]
    p=[[0.0 for j in range(m)] for i in range(m)]
    s=""
    NDIMS = len(x)-1
    NPTS = len(x)
    FUNC = NDIMS
    ncalls = 0;
    for i in range(m):
        for j in range(n):
            if i==j:
                x[i][j]=a[j]+da[j]
                p[i][j]=x[i][j]
            else:
                x[i][j]=a[j]
                p[i][j]=x[i][j]
        p[i][FUNC] = fnelder.func(p[i])
        #print("x[",i,"]=",x[i])
        #print("p[",i,"]=",p[i])
    #print("p",p)
    #Inlet variable definitions
    #fnelder : abstract multivariable function f(x)
    #x : independent variable set of n+1 simplex elements
    #maxiteration : maximum iteration number
    #tolerance :

    #///// construct the starting simplex ////////////////////
    z=[0.0 for i in range(NDIMS)]
    best = 1.0E99;
    #/////////////// calculate the first function values for the simplex ////////////////
    iter=0;
    for iter in range(1,maxiteration):
        #////////  define lo, nhi, hi (low high next_to_high //////////////
        ilo=0
        ihi=0
        inhi = -1
        # -1 means missing
        flo = p[0][FUNC];
        fhi = flo
        #print("flo=",flo,"fhi=",fhi)
        #double pavg,sterr;
        for i in range(1,NPTS):
            if p[i][FUNC] < flo:
                flo=p[i][FUNC]
                ilo=i
            if  p[i][FUNC] > fhi:
                fhi=p[i][FUNC]
                ihi=i
            #print("i=",i,"flo=",flo,"fhi=",fhi,"ilo=",ilo,"ihi=",ihi,"pifunc=",p[i][FUNC])
        fnhi = flo
        inhi = ilo
        #print("fnhi=",fnhi,"inhi=",inhi)
        for i in range(NPTS):
            if (i != ihi) and (p[i][FUNC] > fnhi):
                fnhi=p[i][FUNC]
                inhi=i
        #///////// exit criteria //////////////
        if (iter % 4*NDIMS) == 0:
            #calculate the avarage (including maximum value)
            pavg=0
            for i in range(NPTS):
                 pavg=pavg+p[i][FUNC]
            pavg=pavg/NPTS
            tot=0.0
            if printlist!=0:
                print(iter)
                for j in range(NDIMS):
                    print(p[ilo][j]," ")
            for i in range(NPTS):
                tot=(p[i][FUNC]-pavg)*(p[i][FUNC]-pavg)
            sterr=sqrt(tot/NPTS)
            for j in range(NDIMS):
```

```
            z[j]=p[ilo][j]
        best = p[ilo][FUNC];
    # end of if iter%4*NDMS)==0
    #//// calculate avarage without maximum value //////
    ave=[0.0 for i in range(NDIMS)]
    for j in range(NDIMS):
        ave[j] = 0
    for i in range(NPTS):
        if i != ihi:
            for j in range(NDIMS):
                ave[j] = ave[j]+p[i][j];
    for j in range(NDIMS):
        ave[j]=ave[j]/(NPTS-1)
    #print("ave[",j,"]=",ave[j])
    #//////// reflect ///////////////
    #print("reflect")
    r=[0.0 for i in range(NDIMS)]
    for j in range(NDIMS):
        r[j] = 2.0*ave[j] - p[ihi][j]
    fr = fnelder.func(r)
    if (flo <= fr) and (fr < fnhi): #in zone: accept
        for j in range(NDIMS):
            p[ihi][j] = r[j]
        p[ihi][FUNC] = fr
        continue
    #///(//////////// expand
    if fr < flo:
        #print("expand")
        e=[0.0 for i in range(NDIMS)]
        for j in range(NDIMS):
            e[j] = 3.0*ave[j] - 2.0*p[ihi][j];
        fe = fnelder.func(e);
        if fe < fr:
            for j in range(NDIMS):
                p[ihi][j] = e[j]
            p[ihi][FUNC] = fe
            continue
        else:
            for j in range(NDIMS):
                p[ihi][j] = r[j]
            p[ihi][FUNC] = fr
            continue
    #//////////// shrink:
    if fr < fhi:
        #print("srink")
        c=[0.0 for i in range(NDIMS)]
        for j in range(NDIMS):
            c[j] = 1.5*ave[j] - 0.5*p[ihi][j]
        fc = fnelder.func(c);
        if fc <= fr:
            for j in range(NDIMS):
                p[ihi][j] = c[j];
            p[ihi][FUNC] = fc
            continue
        else:   #////// shrink
            for i in range(NPTS):
                if i != ilo:
                    for j in range(NDIMS):
                        p[i][j] = 0.5*p[ilo][j] + 0.5*p[i][j]
                    p[i][FUNC] = fnelder.func(p[i])
            continue
    #/////////////// contract
    if fr >= fhi:
        #print("contract")
        cc=[0.0 for i in range(NDIMS)]
        for j in range(NDIMS):
            cc[j] = 0.5*ave[j] + 0.5*p[ihi][j];
        fcc = fnelder.func(cc)
        if fcc < fhi:
            for j in range(NDIMS):
                p[ihi][j] = cc[j];
            p[ihi][FUNC] = fcc
            continue
        else:
            for i in range(NPTS):
                if i != ilo:
```

```
                    for j in range(NDIMS):
                        p[i][j] = 0.5*p[ilo][j] + 0.5*p[i][j]
                    p[i][FUNC] = fnelder.func(p[i]);
    return z

a=[0.78,0.7464,1.234]
da=[0.5122,0.643,0.98]
f=f1()
r=nelder(f,a,da,100,1.0e-15,0)
print("r = ",r)
```

```
runfile('E:/okul/SCO1/nelder.py', wdir='E:/okul/SCO1')
Reloaded modules: gauss, f_xj
r =  [1.9999060985742285, 1.0000000151179926, 0.9975092633746454]
```

EXAMPLE PROBLEM

Find minimum value of function $f(x_0, x_1) = 3x_0^2 - 4x_0x_1 + 2x_1^2 - x_0 - x_1$with the initial value of $\{x^0\} = \begin{Bmatrix} x_0^0 \\ x_1^0 \end{Bmatrix} = \begin{Bmatrix} -2 \\ 2 \end{Bmatrix}$ by using Nelder-Mead method

```
class f2(f_xj):
  def func(self,x):
      y=3.0*x[0]*x[0]-4.0*x[0]*x[1]+2.0*x[1]*x[1]-x[0]-x[1]
      return y;
```

```
a=[-2.0,2.0]
da=[0.5122,0.643]
f=f2()
r=nelder(f,a,da,100,1.0e-15,0)
print("r = ",r)
```

```
runfile('E:/okul/SCO1/nelder.py', wdir='E:/okul/SCO1')
Reloaded modules: gauss, f_xj
r =  [1.0000000029030756, 1.2500000011024857]
```

8.14 NON-LINEAR MULTIDIMENSIONAL STEEPEST DESCENT METHOD

For the n dimensional non lineear $f(x_0,x_1,x_2,\ldots,x_n)$ function , we would like to find minimum value of the function. The General definition of the Newton –Raphson approach can be written as:

$$\{x^{m+1}\} = \{x^m\} + \alpha^m[H^m]^{-1}[-\{\nabla f^m\}]$$

where

$$\nabla f(x_1, x_2, x_3, .., x_n) = \begin{Bmatrix} \frac{\partial f(x_1,x_2,x_3,..,x_n)}{\partial x_1} \\ \frac{\partial f(x_1,x_2,x_3,..,x_n)}{\partial x_2} \\ \frac{\partial f(x_1,x_2,x_3,..,x_n)}{\partial x_3} \\ \cdots\cdots \\ \frac{\partial f(x_1,x_2,x_3,..,x_n)}{\partial x_n} \end{Bmatrix} \text{ and } [H] = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \frac{\partial^2 f}{\partial x_1 \partial x_3} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \frac{\partial^2 f}{\partial x_2 \partial x_3} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \frac{\partial^2 f}{\partial x_3 \partial x_1} & \frac{\partial^2 f}{\partial x_3 \partial x_2} & \frac{\partial^2 f}{\partial x_3^2} & \cdots & \frac{\partial^2 f}{\partial x_3 \partial x_n} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \frac{\partial^2 f}{\partial x_1 \partial x_n} & \frac{\partial^2 f}{\partial x_2 \partial x_n} & \frac{\partial^2 f}{\partial x_3 \partial x_n} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

is Hessian Matrix, α is a step function. In steepest descent method the value of inverse hessian, $[H]^{-1}$ is taken as unit vector [I] Then the definition of the equation becomes:

$$\{x^{m+1}\} = \{x^m\} + \alpha^m[-\{\nabla f^m\}]$$

In this method, As a first action a starting point for the iteration will be selected. If the starting function is P_0

$P^0 = f(x_1^0, x_2^0, x_3^0, .., x_n^0)$

In each iteration step values of the function and derivatives are calculated.

$d^k = -\nabla f(x_1^k, x_2^k, x_3^k, .., x_n^k)$ k=0..n

Then for each x value is increased by d^k with an unknown multiplier α. In the new equation as only unkown remains is unknown multiplier α (so equation converted to a one dimensional equation as a function of α)

$$f(\alpha^k) = f(x^k + \alpha^k d^k) = f\left(\left[x_1^k - \alpha^k \frac{\partial f(x_1^k, x_2^k, x_3^k, .., x_n^k)}{\partial x_1}\right], \left[x_2^k - \alpha^k \frac{\partial f(x_1^k, x_2^k, x_3^k, .., x_n^k)}{\partial x_2}\right], \dots, \left[x_n^k - \alpha^k \frac{\partial f(x_1^k, x_2^k, x_3^k, .., x_n^k)}{\partial x_n}\right]\right)$$

α^k value minimizing function $f(\alpha^k)$ is found by using one of the one dimensional minimisation methods. The new x^{k+1} value calculated by using

$$x^{k+1} = x^k + \alpha^k d^k$$

Process continues until the error is small.

EXAMPLE PROBLEM

Find minimum value of function $f(x_0, x_1) = 3x_0^2 - 4x_0x_1 + 2x_1^2 - x_0 - x_1$ with the initial value of $\{x^0\} = \begin{Bmatrix} x_0^0 \\ x_1^0 \end{Bmatrix} = \begin{Bmatrix} -2 \\ 2 \end{Bmatrix}$ by using steepest descent method.

$$\nabla f(x_0, x_1) = \begin{Bmatrix} 6x_0^0 - 4x_1^0 - 1 \\ -4x_0^0 + 4x_1^0 - 1 \end{Bmatrix} = \begin{Bmatrix} -21 \\ 15 \end{Bmatrix}$$

$$\begin{Bmatrix} x_0^1 \\ x_1^1 \end{Bmatrix} = \begin{Bmatrix} x_0^0 \\ x_1^0 \end{Bmatrix} + \alpha^0 \begin{Bmatrix} 6x_0^0 - 4x_1^0 - 1 \\ -4x_0^0 + 4x_1^0 - 1 \end{Bmatrix} = \begin{Bmatrix} -2 \\ 2 \end{Bmatrix} + \alpha^0 \begin{Bmatrix} -21 \\ 15 \end{Bmatrix} = \begin{Bmatrix} -2 - 21\alpha^0 \\ 2 + 15\alpha^0 \end{Bmatrix}$$

If these values are substituted into the function

$f(x_0, x_1) = 3x_0^2 - 4x_0x_1 + 2x_1^2 - x_0 - x_1$ it becomes

$$f(\alpha^0) = 3(-2 - 21\alpha^0)^2 - 4(-2 - 21\alpha^0)(2 + 15\alpha^0) + 2(-2 - 21\alpha^0)^2 - (-2 - 21\alpha^0) - (2 + 15\alpha^0)$$

$$f(\alpha^0) = 36 + 666\alpha^0 + 3033\alpha_0^2$$

In order to find minimum of this function, analitical solution will be used in this example

$$f'^{(\alpha^0)} = 666 + 6066\alpha^0 = 0 \quad \alpha^0 = -0.109792284$$

If this value is substituted into x_0^1 and x_1^1

For α^0=-0.109792284 $\begin{Bmatrix} x_0^1 \\ x_1^1 \end{Bmatrix} = \begin{Bmatrix} 0.3056379 \\ 0.35311572 \end{Bmatrix}$

For α^1=-1.121212 $\begin{Bmatrix} x_0^2 \\ x_1^2 \end{Bmatrix} = \begin{Bmatrix} 0.9544105 \\ 1.2613973 \end{Bmatrix}$

For α^2=-0.109792284 $\begin{Bmatrix} x_0^3 \\ x_1^3 \end{Bmatrix} = \begin{Bmatrix} 0.9894 \\ 1.2363 \end{Bmatrix}$

For α^3=-1.121212 $\begin{Bmatrix} x_0^4 \\ x_1^4 \end{Bmatrix} = \begin{Bmatrix} 0.9993 \\ 1.250173 \end{Bmatrix}$

The same problem is also solved in the algorithm

Program 5.10-2 steepest descent algorithm (Python version)

```
from math import *
from f_xj import *;
from f_x import *;
from gauss import *;

class f1(f_xj):
  def func(self,x):
      y=(1.0-x[0])*(1.0-x[0])+100.0*(x[1]-x[0]*x[0])*(x[1]-x[0]*x[0]);
      return y;

class f1dim(f_x):
    def __init__(self,ff,pi,xi):
      self.name = 'f1dim';
      self.ff1=ff;
      self.pc=pi;
      self.xc=xi;
```

```
    def func(self,x):
        n=len(self.pc);
        xt = [0.0 for ii in range(n)];
        for j in range(n):
            xt[j]=self.pc[j]+x*self.xc[j];
        return self.ff1.func(xt);

def linmin(f,p,xi):
    f1=f1dim(f,p,xi);
    xi=1.235436154162;
    xmin=newton(f1,xi);
    return xmin;

def multiply_c_v(left,right):
    #multiplying a vector with a constant
    n=len(right);
    b = [0.0 for i in range(n)];
    for i in range(n):
        b[i]=left*right[i];
    return b;

def add(left,right):
    #addition of two vectors
    n1=len(left);
    n2=len(right);
    if n1>=n2: nMax=n1;
    else     : nMax=n2;
    b = [0.0 for i in range(nMax)];
    for i in range(n1):
        b[i]=b[i]+left[i];
    for i in range(n2):
        b[i]=b[i]+right[i];
    return b;

def steepestdescent(f,x):
    # steepestdescent method
    ti=1.0;
    nmax=100;
    tolerance=1.0e-10;
    n=len(x);
    b = [0.0 for ii in range(n)];
    dy = [0.0 for ii in range(n)];
    i=0;
    total=0;
    for i in range(n):
        b[i]=1.0;
        total+=abs(b[i]);
    i=0;
    while i<nmax and total>tolerance:
        dy=f.dfunc(x);
        lm=linmin(f,x,dy);
        b= multiply_c_v(lm,dy);
        x=add(x,b);
        total=0;
        for j in range(n):
            total+=abs(b[j]);
        i=i+1;
    return x;

def newton(f,x):
   # Newton-Raphson optimisation method
   # single variable
  nmax=100
  tolerance=1.0e-10
  for i in range(0,nmax):
      fx= f.dfunc(x);
      dfx=f.d2func(x);
      x=x-fx/dfx
      if abs(fx)<tolerance: return x
  return x;

x=[1,1];
f=f1();
r=steepestdescent(f,x);
print("r=",r);
```

```
runfile('D:/okul/SCO1/steepest_descent_opt.py', wdir='D:/okul/SCO1')
Reloaded modules: f_xj, f_x, gauss
r= [1.0000095779092373, 1.2500087681731902]
```

8.15 HOMOTOPHY(CONTINUATION) METHOD

Homotophy or continuation method is explain in detail at chapter 9.2 as nonlinear root finding method. Nonlinear root finding algorithms can be converted by using derivatives instead of function and second derivatives (hessian matrix) for derivatives

$$\nabla f(x_1, x_2, x_3, .., x_n) = \begin{Bmatrix} \frac{\partial f(x_1,x_2,x_3,..,x_n)}{\partial x_1} \\ \frac{\partial f(x_1,x_2,x_3,..,x_n)}{\partial x_2} \\ \frac{\partial f(x_1,x_2,x_3,..,x_n)}{\partial x_3} \\ \cdots \\ \frac{\partial f(x_1,x_2,x_3,..,x_n)}{\partial x_n} \end{Bmatrix} \text{ and } [H] = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \frac{\partial^2 f}{\partial x_1 \partial x_3} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \frac{\partial^2 f}{\partial x_2 \partial x_3} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \frac{\partial^2 f}{\partial x_3 \partial x_1} & \frac{\partial^2 f}{\partial x_3 \partial x_2} & \frac{\partial^2 f}{\partial x_3^2} & \cdots & \frac{\partial^2 f}{\partial x_3 \partial x_n} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \frac{\partial^2 f}{\partial x_1 \partial x_n} & \frac{\partial^2 f}{\partial x_2 \partial x_n} & \frac{\partial^2 f}{\partial x_3 \partial x_n} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

```
#continuation_optimisation
# Calculates solution of nonlinear system of equation for derivative of optimisation function
from math import *
from fi_xi import *
from f_xj import *
from continuation_NLE import *

# create derivative of optimisation function
# as a new function (system of non-linear equations)
class f2(fi_xi):
  def __init__(self,f):
      self.ff1=f

  def func(self,x):
     ff=self.ff1.dfunc(x)
     return ff

# function to optimise
class f1(f_xj):
  def func(self,x):
     y=3.0*x[0]*x[0]-4.0*x[0]*x[1]+2.0*x[1]*x[1]-x[0]-x[1];
     return y;

fa=f1()
fb=f2(fa)
x0=[0.5,1.0]
a=continuationRK6(fb,x0,10)
print("a=\n",a)
```

```
#continuation_NLE solution of non-linear system of equation
from math import *
from fi_xi import *
from gauss import *

def newton(f,x):
   # Newton-Raphson root finding method
   k=len(x);
   nmax=100;
   tolerance=1.0e-10
   for i in range(nmax):
      fx= f.func(x);
```

```
      dfx=f.dfunc(x,1);
      dx=gauss(dfx,fx);
      for j in range(k):
         x[j]-=dx[j];
      total=0;
      for j in range(k):
         total+=dx[j];
      if abs(total)<tolerance: return x
   return x

def mult_c_v(left,right):
    #multiplying a vector with a constant
    i=0
    n=len(right)
    b=[0 for x in range(n)]
    for i in range(n):
       b[i]=left*right[i]
    return b;

def mult_m_m(left,right):
    #multiplying two matrisis
    n1=len(left);
    m1=len(left[0]);
    n2=len(right);
    m2=len(right[0]);
    b = [[0 for x in range(n1)] for y in range(m2)];
    if m1 != n2:
       print("warning : matrix sizes should be same");
       return b;
    for i in range(0,n1):
       for j in range(0,m2):
          for k in range(0,m1):
             b[i][j]+=left[i][k]*right[k][j];
    return b
def mult_m_v(left,right):
    #multiplication of one matrix with one vector
    #int ii,jj,i,j,k;
    m1=len(left[0])
    n1=len(left)
    m2=len(right)
    b = [0 for x in range(m2)]
    if n1 != m2:
        print("inner matrix dimensions must agree")
        for ii in range(n1):
           b[ii]=0
        return b;
    for i in range(m1):
        b[i]=0
        for k in range(n1):
           b[i]=b[i]+left[i][k]*right[k]
    return b;
#end of multiply of a matrix and a vector

def mult_v_m(left,right):
    #multiplication of one vector with one matrix
    #ii,jj,i,j,k;
    m2=len(right[0])
    n2=len(right)
    m1=len(left)
    b=[0.0 for i in range(m1)]
    if n2 != m1:
        print("inner matrix dimensions must agree")
        for ii in range(n2):
           b[ii]=0
           return b
        for i in range(m2):
```

```
            b[i]=0;
            for k in rage(m1):
                b[i]+=right[i][k]*left[k]
    return b

def mult_c_m(left,right):
    #multiplying a matrix with a constant
    n=len(right);
    m=len(right[0])
    b = [[0 for x in range(m)] for y in range(n)];
    for i in range(0,n):
      for j in range(0,m):
        b[i][j]=right[i][j]*left;
    return b
    #end of multiplying a matrix with a constant double

def add_m_m(left,right):
    #adding two matrisis
    n1=len(left);
    m1=len(left[0]);
    n2=len(right);
    m2=len(right[0]);
    b = [[0 for x in range(n1)] for y in range(m2)];
    if m1 != n2:
      print("warning : matrix sizes should be same");
      return b;
    for i in range(0,n1):
      for j in range(0,m2):
            b[i][j]=left[i][j]+right[i][j];
    return b

def add_v_v(left,right):
    #addition of two vectors
    n1=len(left)
    n2=len(right)
    nMax=0
    i=0
    if n1>=n2:
       nMax=n1
    else:
       nMax=n2
    b=[0 for x in range(nMax)]
    for i in range(n1):
       b[i]=b[i]+left[i]
    for i in range(n2):
       b[i]=b[i]+right[i]
    return b
#end of vector addition method

def continuationRK6(f,x,N):
    #=====================================================
    # Roots of nonlinear system of equations Homotophy RK6
    # yi+1 = yi + (1/90)*( 7k1 + 32k3 +12k4+32k5+7k6)h
    # k1=f(xi,yi)
    # k2=f(xi+0.25h , yi+0.25k1h)
    # k3=f(xi+0.25h , yi+0.125k1h+0.125k2h)
    # k4=f(xi+0.5h , yi - 0.5k2h+k3h)
    # k5=f(xi+0.75h , yi + (3/16)k1h+(9/16)k4h)
    # k6=f(xi+h , yi - (3/7)k1h+(2/7)k2h+(12/7)k3h - (12/7)k4h+(8/7)k5h)
    #====================================================
    #x vector of independent variables
    #y vector of dependent variables
    #dy derivative vector of dependent variables
    i=0
    nmax=400
    tolerance=1.0e-20
```

```
    n=len(x)
    h=1.0/float(N)
    b=[0.0 for i in range(n)]
    x1=[0.0 for i in range(n)]
    x0=[0.0 for i in range(n)]
    b0=[0.0 for i in range(n)]
    for i in range(n):
        x0[i]=x[i]
    k = [[0 for x in range(n)] for y in range(6)]
    b=mult_c_v(-h,f.func(x))
    for i in range(N):
        A=f.dfunc(x,1)
        k[0]=gauss(A,b)
        b0=mult_c_v(-h,f.func(x))
        b=b0
        x1=add_v_v(x,mult_c_v(0.25,k[0]));
        A=f.dfunc(x1,1)
        #k2=f(xi+0.25h , yi+0.25k1h)
        k[1]=gauss(A,b)
        b=mult_c_v(-h,f.func(x0))
        x1=add_v_v(x,add_v_v(mult_c_v(0.125,k[0]),mult_c_v(0.125,k[1])))
        A=f.dfunc(x1,1)
        #k3=f(xi+0.25h , yi+0.125k1h+0.125k2h)
        k[2]=gauss(A,b)
        b=mult_c_v(-h,f.func(x0))
        x1=add_v_v(x,add_v_v(mult_c_v(-0.5,k[1]),k[2]));
        A=f.dfunc(x1,1)
        #k4=f(xi+0.5h , yi - 0.5k2h+k3h)
        k[3]=gauss(A,b)
        b=mult_c_v(-h,f.func(x0))
        x1=add_v_v(x,add_v_v(mult_c_v((3.0/16.0),k[0]),mult_c_v((9.0/16.0),k[3])));
        A=f.dfunc(x1,1)
        #k5=f(xi+0.75h , yi + (3/16)k1h+(9/16)k4h)
        k[4]=gauss(A,b)
        b=mult_c_v(-h,f.func(x0))
        x1=add_v_v(x,add_v_v(mult_c_v((-
3.0/7.0),k[0]),add_v_v(mult_c_v((2.0/7.0),k[1]),add_v_v(mult_c_v((12.0/7.0),k[2]),add_v_v(mult_c_v((-
12.0/7.0),k[3]),mult_c_v((8.0/7.0),k[4])))));
        A=f.dfunc(x1,1)
        #k6=f(xi+h , yi - (3/7)k1h+(2/7)k2h+(12/7)k3h - (12/7)k4h+(8/7)k5h)
        k[5]=gauss(A,b)
        b=mult_c_v(-h,f.func(x0))
        #yi+1 = yi + (1/90)*( 7k1 + 32k3 +12k4+32k5+7k6)h
        for j in range(n):
            x[j]=x[j]+1.0/90.0*(7.0*k[0][j]+32.0*k[2][j]+12.0*k[3][j]+32.0*k[4][j]+7.0*k[5][j])
    return x
```

```
runfile('D:/okul/SCO1/continuation_opt.py', wdir='D:/okul/SCO1')
Reloaded modules: if_x
a=
 [1.0000000032646588, 1.250000003940646]
```

8.16 SCHOLASTIC METHODS : GENETIC ALGORITHMS

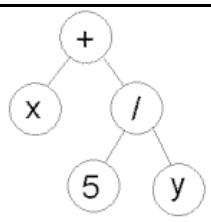
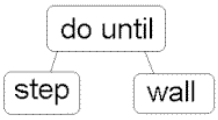
8.16.1 NATURAL SELECTION AND SYSTEM OF GENES IN NATURE

Genetic algorithms are a widely use modern programming method. This method based on natural selection mechanism in nature. Characteristics are carried in GENES, Father and Mother passes their genes to the child. Individuals carrying genes that given better characteristics to fit the nature has a better chance of survival and to give offsprings, therefore more of their gene characteristics will be carried into the next generation. Sometimes gene structure passing throgh from parents can be changed by a mechanism called mutation. It is usuall a very small change. The result of this change could be deadly(less survival potential in environment) or sometimes make the individual fit the envronment better. If that is the case, his chance of passing his genes including this mutated one will be improved.

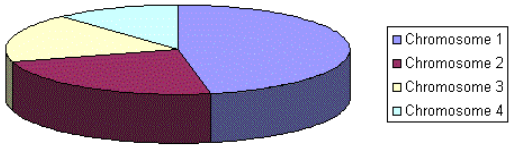
8.16.2 GENETIC ALGORITHMS, GENERAL DEFINITIONS

In general an algorithm of optimization by genetic algorithms can be summerized as:

[START] Set up a population of m member (one chromosome for each member, x gene-vector of independent variables)
[START] for each chromosome (vector of independent variables) calculate *f(x)* fitting value, repeat this for each individual in population
[NEW GENERATION] Generate new generations according to the following steps
[SELECTION] Select parents from population by considering their fitting values. Better fits should have a better chance to be selected but no one is eliminated completely from the selection process.
[PRODUCTION] Produce a new individual(s) from the parents. Replace parent generation with the new generation. Some of the successful parents can retain in the new generation
[MUTATION] mutate new generations with a ceratain (given) mutation rate
[ACCEPT] Add new individuals to the population.
[REPEAT] use the new generations to create new populations by using the same rule
[TEST] when the required conditions are occured or enough generations are generated stop the simulation
[LOOP] If the required conditions are not occured continue the process
Gens an choromosomes are the basic calculating units of the genetic calculations. Mathematically all the independent variales are an individual gene. . Chromsome is the vector of genes(independent varables). SAmple of some chromosome types utilised fort he computer simulation are given as the following table

chromosome A **chromosome B** **chromosome C**	**5.3243 0.4556 2.3293 2.4545** **ABDJEIFJDHDIERJFDLDFLFEGT** **(back), (foward), (right), (left), (turn)**
chromosome D	 (+ x (/ 5 y))
chromosome E	 (do_until step wall)

In the natural genes the fitting function is the survival in the nature, in mathmatical genes optimization (Maximisation) of the fitting function will be the base. Fitting functions usually added up and converted into the percantage form

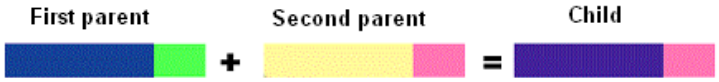


This percentage is called converted fitting function. Percentage in the total pipe is biger for the individuals with a beter fitting functions. Parents selected by using converted fitting functions and random number, so that individuals with a beter fitting has a beter chance to be selected. From the selected population of parent candidates actual parents are also selected by random selection. Then from parents gene the gene of th new population is created. Mathematically this process is called crossing. In crossing process genes of each parents are randomly selected to create the new child. For example if the gene is binary numbers and if single crossing is used, both parents gene is divided from a randomly selected point the first part of first parent is combined with the second part of second parent for the second part of the first parent combined with the first part of the second parent to crate a child.

For example for a binary group of 8 bit:

If 8*random(0-1) = 5 and if the first 5 bit is taken from the first parent and the remaining bits from the second parent : The new child:

11001011 + 11011111 = 11001111



Chromosome 1 (First Parent)	11011 | 00100110110
Chromosome 2 (Second parent)	11011 | 11000011110
1. Child	11011 | 11000011110
2. Child	11011 | 00100110110

If double crossing applied, two random numbers are generated and genes are divided into three parts. The first part is taken from the first parent, second part is taken from the second parent and the third part is taken from the first parent(or vice versa)

For example for a binary group of 8 bit i:

Random1=8*random(0-1) = 2

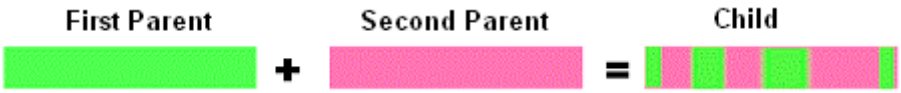
Random2=8*random(0-1) = 6

First 2 bit is taken from the first parent, from third to 6th bit is taken from the second parent and 7th and 8th bit is taken from the first parent.

11001011 + 11011111 = 11011111

As a third method every gene is selected by a random number decision process. If generated random number is smaller than 0.5 the first parent's binary digit is selected, if not second parents binary digit is selected.

11001011 + 11011111 = 11001011



8.16.3 GENETIC ALGORITHMS, OPTIMIZATION

In this section actual optimization process with genetic algorithms will be given in details. AS it is seen earlier in this chapter optimization process is minimisation or maximisation of a multivariable function

f(x1,x2,x3,…..xi)

It should again be rminded tt maximisation and minimisation is the same process with a sign change

max [f(x)] = min [- f(x)] = min [g(x)]
Another mathematical fact that will be used in genetic algorithms is that addition of a constant to the function will not change the optimum point
max [f(x)] = max [f(x) + C] = max [g()]
In genetic algorithm applications the nature of the algorithm is to find the maximum, in order to get correct percentages constant numbers are added into the function to ensure the resulting value is always positive.
In engineering optimization we usually face the real numbers. IF a real number is to be represented as a binary sequence, a conversion process between binary digits and real numbers are needed. If n+1 independent variable is existed as
(x0,x1,x2,..xi..xn)
In order to apply genetic algorithms lower and upper bounds of each variable should be known.
xo : xo_minimum xo_maximum
x1 : x1_minimum x1_maximum
….
xi : xi_minimum xi_maximum
…
xn : xn_minimum xn_maximum
definitins should be given. Ifbinary digit system is used number of representation digits also should be known. If the independent variable is defined as
xi_min <= xi <= xi_max N bit binary
Binary equivalent of N digits can be calculated as follows:
Binary group with n digit first converted into a 10 base integer number as

$$(b_N, b_{N-1}, b_{N-2}, \dots., b_1, b_0)_2 = \left(\sum_{i=0}^{N} b_i 2^i\right)_{10} = x'$$

Then the integer number is converted into a binary digit by using the following conversion process

$$x = xmin + x'\left(\frac{xmax - xmin}{2^N - 1}\right)$$

Inverse process can also be apply to convert digits back to the binary digits. Each independent variables can be given with one gene, and the set of independent variables can be given b a a vector of genes. Evaluation function is assumd to be in the following form

```
class f1(f_xj):func=lambda self,x:21.5+x[0]*sin(4.0*pi*x[0])+x[1]*sin(20.0*pi*x[1])
```

The value sets of independent variables as whole together with the evaluated function value will be called as genotype (as borrowed from nature's genetic system). Number of chromosomes in each individual can be one or more. Number of individual will be set by the user of the program. For each individual starting variable values set by using random selection. Function values are calculated for each genotype. It is the fitting value of the genome. All fitting values are adde and then normalise to get percantage base fitting values. Percantage base fitting values are adde by starting from zero to 1and created cumulative fitting scale for all population. Cumulative fitting values are based on the scale of the fitting function. If a random number between 0 to 1 is selected, the chance of this number is fall in a region with a bigger fitting function is grater than that of a small one. After an individual in the population is selected as a parent cendidate, it can be removed from the selection list or it can remain in the list(a second selection can be allowed). From the selected parent candidates two parents are selected randomly to create a new child.In the new child population a selected percentage can be allowed (selected again randomly) to mutate. Mutation in a binary code can be achived to flip an indivudual binary digit(from 0 to 1 or from 1 to 0). Process continues for a defined number of generations.

8.16.4 GENETIC ALGORITHMS, CLASS GENE

Genetic algorithms mimic the natural selection process of genes. The first structure used here to mimic natural genes are used binary numbers as defined in previous subchapter. Constructor of gene class can accept either double or binary numbers. Total bit length is a variable but it is limited to 64 bits (0—63). Gene class consist of gene crossing and mutation processes as well.

Gene class

```
 from abc import ABC, abstractmethod
from math import *
import random
from f_x import *


class Gene(ABC):

    def __init__(self, xming,xmaxg,N):
        self.xming=xming
        self.xmaxg=xmaxg
        self.N=N
        self.xN=2**N-1
        self.rand_float(N)


    #get a single bit of Gene
    def getBit(self,n):
        return self.xs[n]

    # creates a random string of p long 0 or 1 (binary)
    def rand_str(self,p):
        key1 = ""
        for i in range(p):
            key1=key1+str(random.randint(0, 1))
        return key1

    # set Gene from a string binary p long like "00101100.."
    def set_str(self,s):
        self.xs=s
        self.xp=int(s,2)
        self.x=self.xming+(self.xmaxg-self.xming)/self.xN*self.xp

    # create a random integer(10 base) xp based on p long binary
    def rand_int(self,p):
        key1 = ""
        for i in range(p):
            key1=key1+str(random.randint(0, 1))
        return int(key1,2)
    # create a random binary(2 base) xp based on p long binary
    def rand_bin(self,p):
        key1 = ""
        for i in range(p):
            key1=key1+str(random.randint(0, 1))
        x=int(key1,2)
        return bin(x)

    # create a random float xmin<=x<xmax
    # and set gene to this value
    def rand_float(self,p):
        key1 = ""
        for i in range(p):
            key1=key1+str(random.randint(0, 1))
        self.xs=key1
        self.xp=int(key1,2)
```

```
        self.x=self.xming+(self.xmaxg-self.xming)/self.xN*self.xp

    #print gene values
    def print(self):
        print("N=",self.N,"xs=",self.xs,"x=",self.x)

    #flip a single bit at position n if 0 to 1 if 1 to 0
    def flip(self,nn):
        if nn>self.N-1:
            nn=self.N-1
        else:
            nn=int(random.random()*self.N)
        if self.xs[nn]=='0': self.xs=self.xs[:nn]+'1'+self.xs[nn+1:]
        else: self.xs=self.xs[:nn]+'0'+self.xs[nn+1:]
        self.xp=int(self.xs,2)
        self.x=self.xming+(self.xmaxg-self.xming)/self.xN*self.xp

     #flip a single bit at a random position if 0 to 1 if 1 to 0
    def mutate(self):
        nn=int(random.random()*self.N)
        #print("nn=",nn)
        if self.xs[nn]=='0': self.xs=self.xs[:nn]+'1'+self.xs[nn+1:]
        else: self.xs=self.xs[:nn]+'0'+self.xs[nn+1:]
        self.xp=int(self.xs,2)
        self.x=self.xming+(self.xmaxg-self.xming)/self.xN*self.xp

    # split string binary gene into two pieces
    # seperated at point n
    def split(self,nn):
        #nn=int(random.random()*self.N)
        #print("nn=",nn)
        s1=self.xs[:nn]
        s2=self.xs[nn:]
        s=[s1,s2]
        return s

    # split string binary gene into three pieces
    # seperated at point n
    def split2(self,nn1,nn2):
        #nn=int(random.random()*self.N)
        #print("nn=",nn)
        if nn2>=nn1:
            s1=self.xs[:nn1]
            s2=self.xs[nn1:nn2]
            s3=self.xs[nn2:]
        else:
            s1=self.xs[:nn2]
            s2=self.xs[nn2:nn1]
            s3=self.xs[nn1:]
        s=[s1,s2,s3]
        return s

    # assign random value where xmin<=x<=xmax to gene
    def setbits(self):
        self.rand_float(self.N)

    def print(self):
        print("N=",self.N,"xs=",self.xs,"x=",self.x)

# assign s value where xmin<=x<=xmax to a gene
#return a new Gene
def setGene_str(s,xming,xmaxg):
```

```
    N=len(s)
    f=Gene(xming,xmaxg,N)
    f.xs=s
    f.xp=int(s,2)
    f.x=f.xming+(f.xmaxg-f.xming)/f.xN*f.xp
    return f

#creates a random int 0 or 1
def int_random():
    r=0
    if random.random()>=0.5:
        r=1
    return r

#creates a random int 0 or 1
def int3_random():
    r=0
    r1=random.random()
    if r1>=0.333 and r1<0.666:
        r=1
    else:
        r=2
    return r

#creates a random boolean True or False
def boolean_random():
    r=False
    if random.random()>=0.5:
        r=True
    return r

# assign x value where xmin<=x<=xmax to a gene
# return a new Gene
def setGene_float(x,xmin,xmax,N):
    f=Gene(xmin,xmax,N)
    f.x=x
    f.xp=int((f.x-f.xming)/(f.xmaxg-f.xming)*f.xN)
    ss=str(bin(f.xp))
    ss1=str(ss[2:])
    if len(ss1)!=N: ss1='0'+ss1
    f.xs=ss1
    return f

# cross correlation  1 of two Genes
def crossing1(f1,f2,N):
    s1=f1.split(N)
    s2=f2.split(N)
    ss1=s1[0]+s2[1]
    ss2=s2[0]+s1[1]
    ff1=setGene_str(ss1,f1.xming,f1.xmaxg)
    ff2=setGene_str(ss2,f2.xming,f2.xmaxg)
    if random.random()<=0.5:
        return ff1
    else:
        return ff2

# cross correlation2 of two Genes
def crossing2(f1,f2,N1,N2):
    s1=f1.split2(N1,N2)
    s2=f2.split2(N1,N2)
    ss1=s1[0]+s2[1]+s1[2]
    ss2=s2[0]+s1[1]+s2[2]
```

```
    ff1=setGene_str(ss1,f1.xming,f1.xmaxg)
    ff2=setGene_str(ss2,f2.xming,f2.xmaxg)
    if random.random()<=0.5:
        return ff1
    else:
        return ff2

# cross correlationN of two Genes
def crossingN(f1,f2):
    key1 = ""
    for i in range(f1.N):
        if boolean_random:
            key1=key1+f1.getBit(i)
        else:
            key1=key1+f2.getBit(i)
    fchild=setGene_str(key1,f1.xming,f1.xmaxg)
    return fchild

# cross correlationN of two Genes
# crossing1 crossing2 or crossing3 is selceted randomly
def crossingX(f1,f2):
    if int3_random()==0:
        N=int(f1.N*random.random())
        fchild=crossing1(f1,f2,N)
    elif int3_random()==1:
        N1=int(f1.N*random.random())
        N2=int(f1.N*random.random())
        if N1<=N2:
            fchild=crossing2(f1,f2,N1,N2)
        else:
            fchild=crossing2(f1,f2,N2,N1)
    else:
        fchild=crossingN(f1,f2)
    return fchild

# cross correlationN of two Genes
# crossing1 crossing2 or crossing3 is selceted randomly
def crossing(f1,f2):
    if int_random()==0:
        N=int(f1.N*random.random())
        fchild=crossing1(f1,f2,N)
    else:  # int_random()==1:
        N1=int(f1.N*random.random())
        N2=int(f1.N*random.random())
        if N1<=N2:
            fchild=crossing2(f1,f2,N1,N2)
        else:
            fchild=crossing2(f1,f2,N2,N1)
    return fchild
```

Few examples are given to explain gene processes. In the first example bit gene is created and cross correletion applied. The results are listed

GeneTest class creating random genes 32 bits between 0 to 3 limit values

```
 N=32
xmin=0.0
xmax=3.0
f1=Gene(xmin,xmax,N)
f1.print()
f2=Gene(xmin,xmax,N)
```

```
f2.print()
f3=crossing(f1,f2)
f3.print()
```

```
runfile('E:/okul/SCO1/Gene_test.py', wdir='E:/okul/SCO1')
Reloaded modules: f_xj, f_x, Gene, Genotype, Genetic
N= 32 xs= 11000001110101000110110100101110 x= 2.2714428632220818
N= 32 xs= 11110010101010111001100101111111 x= 2.8437927066916115
N= 32 xs= 11110010101010111001100101111110 x= 2.8437927059931196
```

Program 5.18.4-3 GeneTest1 class creating 63 bits between 0 to 10 limit values, value of 3.2

```
 from Gene import *
s="010100011110101110000101000111101011100001010001111011000000000"
f=setGene_str(s,0.0,10.0)
f.print()
print(f.split(5))
f1=setGene_float(3.2,0.0,10.0,f.N)
f1.print()
print(f1.split(5))
```

```
runfile('E:/okul/SCO1/Gene_test2.py', wdir='E:/okul/SCO1')
Reloaded modules: f_x
N= 63 xs= 010100011110101110000101000111101011100001010001111011000000000 x= 3.2
['01010', '001111010111000010100011110101110000101000111101100000000']
N= 63 xs= 010100011110101110000101000111101011100001010001111011000000000 x= 3.2
['01010', '001111010111000010100011110101110000101000111101100000000']

In [160]:
```

8.16.5 GENETIC ALGORITHMS, CLASSES GENOTYPE AND GENETIC

Genotype1 can be considered as vector of Gene class variables. In means of Genetics, it represents a chromosome. In mathematically, it is a set of independent variables **(x0,x1,x2,..xi..xn)**
The main class of genetic algorithms is class Genetic1. Genetic optimization process with all the stages defined above calculated in this class. Some of the variables of Genetic1 should be defined by user. These variables are:

```
int    POPSIZE; //number of individuals in population
int    MAXGENS; //  maximum number of generations
int    NVARS;   // number of variables in chromosome (number of independent variables)
 int    N;       //bit size of the binary genes
 double PXOVER; //probability of cross-over;
 double PMUTATION; //probability of mutation
```

method evaluate is the main method to do genetic algorithm calculations. It should be emhised again that all the function values in Genetic algorithms should be positive, if it is not the case, please add up a constant to convert the values into positive values. This proses does not change the optimum. Maximum of function $\mathbf{f(x_1,x_2,x_3,\ldots,x_n)}$ is same as the maximum of $\mathbf{g(x_1,x_2,x_3,\ldots,x_n) = f(x_1,x_2,x_3,\ldots,x_n) + C}$.
This process help to avoid errors during the calculations of cumulative percentage fitting function.

Class Genotype

```
 from abc import ABC, abstractmethod
from math import *
import random
```

```
from f_xj import *
from Gene import *


class Genotype(ABC):

    def __init__(self, f,NVARS,low1,high1,N):
        self.xmin=low1
        self.xmax=high1
        self.NVARS=NVARS
        self.N=N
        self.f=f
        self.G=[0.0 for j in range(self.NVARS)]
        self.x=[0.0 for j in range(self.NVARS)]
        for i in range(self.NVARS):
            self.G[i]=Gene(self.xmin[i],self.xmax[i],self.N)
            self.x[i]=self.G[i].x
        self.Fitness=self.f.func(self.x)
        self.RFitness=0.0
        self.CFitness=0.0
        self.Breeder=False

    def setGene(val):
        G[val]=Gene(self.xmin[val],self.xmax[val],self.N)
        return self.G[val]

    def getGene(val):
        return G[val]

    #print gene values
    def print(self):
        for i in range(self.NVARS):
            self.G[i].print()
        print("Fitness = ",self.Fitness)
        print("RFitness = ",self.RFitness)
        print("CFitness = ",self.CFitness)

    def setFitness(self):
        for i in range(self.NVARS):
            self.x[i]=self.G[i].x
        self.Fitness=self.f.func(self.x)

    def getFitness(self):
        return self.Fitness

    def setRFitness(self,sum):
        self.RFitness=self.Fitness/sum

    def setCFitness(self,x):
        self.CFitness=x

    def getCFitness(self):
        return self.CFitness

    def breed(self):
        self.Breeder=True

    def not_breed(self):
        self.Breeder=False

    def isBreed(self):
        return self.Breeder

    def copygenotype(self):
        g1=self
        return g1
    def mutate(self,PMUTATION):
        for i in range(self.NVARS):
            a=random.random()
            if a<PMUTATION:
```

```
            self.G[i].mutate()
def crossingGT(G1,G2):
    nv=G1.NVARS
    G3=Genotype(G1.f,G1.NVARS,G1.xmin,G1.xmax,G1.N)
    for i in range(nv):
        G3.G[i]=crossing(G1.G[i],G2.G[i])
```

Genotype examples:

Cross correlation and mutating processes

```
from f_xj import *
from  Genotype import *

N=32
xmin=[0.0,0.0]
xmax=[5.0,5.0]
class f1(f_xj):func=lambda self,x: x[0]*x[0]+x[1]*x[1]
f = f1()
NVARS=2
N=32
G1=Genotype(f,NVARS,xmin,xmax,N)
G2=Genotype(f,NVARS,xmin,xmax,N)
print("G1 = ")
G1.print()
print("G2 = ")
G2.print()
print("breed")
G3=crossingGT(G1,G2)
G3.print()
G3.mutate(0.1)
G3.print()
```

```
runfile('E:/okul/SCO1/Genotype_test1.py', wdir='E:/okul/SCO1')
Reloaded modules: f_x, Gene
G1 =
N= 32 xs= 01001111100000010011000011000011 x= 1.5528252014314814
N= 32 xs= 00000101110001011010001101101101 x= 0.11273486193100336
Fitness =  2.423975255295323
RFitness =  0.0
CFitness =  0.0
G2 =
N= 32 xs= 01110000101101101000001110011001 x= 2.2014247177172046
N= 32 xs= 00100110101001111111011001100010 x= 0.7550020168430642
Fitness =  5.416298833213369
RFitness =  0.0
CFitness =  0.0
breed
N= 32 xs= 01001111100000010011001111000011 x= 1.5528260955011532
N= 32 xs= 00100110101001111111011001100010 x= 0.7550020168430642
Fitness =  9.980521648925373
RFitness =  0.0
CFitness =  0.0
N= 32 xs= 01001111100000010011001111000011 x= 1.5528260955011532
N= 32 xs= 00100110101001111111011001100010 x= 0.7550020168430642
Fitness =  9.980521648925373
RFitness =  0.0
CFitness =  0.0
```

Class Genetic

```
from abc import ABC, abstractmethod
from math import *
import random
from f_xj import *
from Genotype import *


class Genetic(ABC):
    def __init__(self,f,low,high,POPSIZE,MAXGENS,NVARS,N,PXOVER,PMUTATION):
        # function derived from f_xj class f(x0,x1,..xn)
        self.POPSIZE=POPSIZE   # population size
```

```
        self.MAXGENS=MAXGENS   # maksimum number of generations
        self.NVARS=NVARS    # number of variables
        #number of independent variable of function
        #number of Genes in genotype
        self.N=N
        #number of bits in a gene
        self.PXOVER=PXOVER
        #probability of cross-over
        self.PMUTATION=PMUTATION
        #proportion of mutated variables
        self.NBreeder=int(0.8*POPSIZE)
        #number of survivers in a generation
        self.Best=0
        self.f=f
        self.xmin=[0.0 for j in range(self.NVARS)]
        self.xmax=[0.0 for j in range(self.NVARS)]
        for i in range(self.NVARS):
            self.xmin[i]=low[i]
            self.xmax[i]=high[i]
        #Best genotype in the population
        self.Population=[0.0 for j in range(self.POPSIZE+1)]
        self.setPopulation()
        self.setRFCF()
        self.evaluate()

    def print(self):
        print("Population size = ",self.POPSIZE)
        print("Maximum number of Generations = ",self.MAXGENS)
        print("Number of Bits in Gene = ",self.N)
        print("Probability of crossover =",self.PXOVER)
        print("Propertion of mutated variables",self.PMUTATION)
        for i in range(self.POPSIZE+1):
            self.Population[i].print()


    def setPopulation(self):
        for i in range(self.POPSIZE+1):
            self.Population[i]=Genotype( self.f,self.NVARS,self.xmin,self.xmax,self.N)
            self.Population[i].setFitness()
        self.setRFCF()
        self.evaluate()


    def setRFCF(self):
        sum=0.0
        for i in range(self.POPSIZE):
            self.Population[i].setFitness()
            sum=sum+self.Population[i].Fitness

        for i in range(self.POPSIZE):
            self.Population[i].setRFitness(sum)
        sum=0.0
        for i in range(self.POPSIZE):
            sum=sum+self.Population[i].RFitness
            self.Population[i].setCFitness(sum)

    def evaluate(self):
        for i in range(self.POPSIZE):
            if self.Population[i].Fitness>self.Population[self.POPSIZE].Fitness:
                self.Population[self.POPSIZE]=self.Population[i]

    def breding_individual(self):
        #select members for breeding
        j=0
        for i in range(self.POPSIZE):
            self.Population[i].not_breed()
        for i in range(self.POPSIZE):
            r=random.random()
            if i==0 and r<self.Population[i].CFitness:
                j=i
            elif  r>=self.Population[i-1].CFitness and  r<=self.Population[i].CFitness:
                j=i
        return j


    def crossover(self):
```

```
        for mem in range(self.POPSIZE):
            POne= int(self.breding_individual())
            PTwo=int(self.breding_individual())
            Po=[0.0 for j in range(2)]
            Po[0]=self.Population[POne]
            Po[1]=self.Population[PTwo]
            self.Population[POne]=crossingGT(Po[0],Po[1])
            self.Population[PTwo]=crossingGT(Po[0],Po[1])
            self.Population[POne].mutate(self.PMUTATION)
            self.Population[PTwo].mutate(self.PMUTATION)
            self.setRFCF()
            self.evaluate()

    def getBest(self):
        aa=[0.0 for j in range(self.NVARS+1)]
        for i in range(self.NVARS):
            aa[i]=self.Population[self.POPSIZE].x[i]
        aa[self.NVARS]=self.Population[self.POPSIZE].Fitness
        return aa

    def select(self):
        Po=[0.0 for j in range(self.POPSIZE+1)]
        for i in range(self.POPSIZE+1):
            mem=0
            r=random.random()
            gf1=self.Population[i].getCFitness()
            gf2=0.0
            if gf1>r:
                Po[i]=self.Population[mem].copygenotype()
            for mm in range(1,self.POPSIZE):
                gf2=self.Population[mem].getCFitness()
                if gf2>r and gf1<r:
                    Po[i]=self.Population[mem].copygenotype()
                    break
        for i in range(self.POPSIZE):
            self.Population[i]=Po[i]
        self.setRFCF()
        self.evaluate()

    def calculate(self):
         self.evaluate()
         Generation=0
         while Generation<self.MAXGENS:
             Generation=Generation+1
             self.crossover()
             self.evaluate()
         return self.getBest()

def setRFCF(Co):
    sum=0.0
    for i in range(Co.POPSIZE+1):
        Co.Population[i].setFitness()
        sum=sum+Co.Population[i].Fitness
    for i in range(Co.POPSIZE+1):
        Co.Population[i].setRFitness(sum)
        sum=0
        for i in range(Co.POPSIZE+1):
            sum=sum+Co.Population[i].RFitness
            Co.Population[i].setCFitness(sum)
```

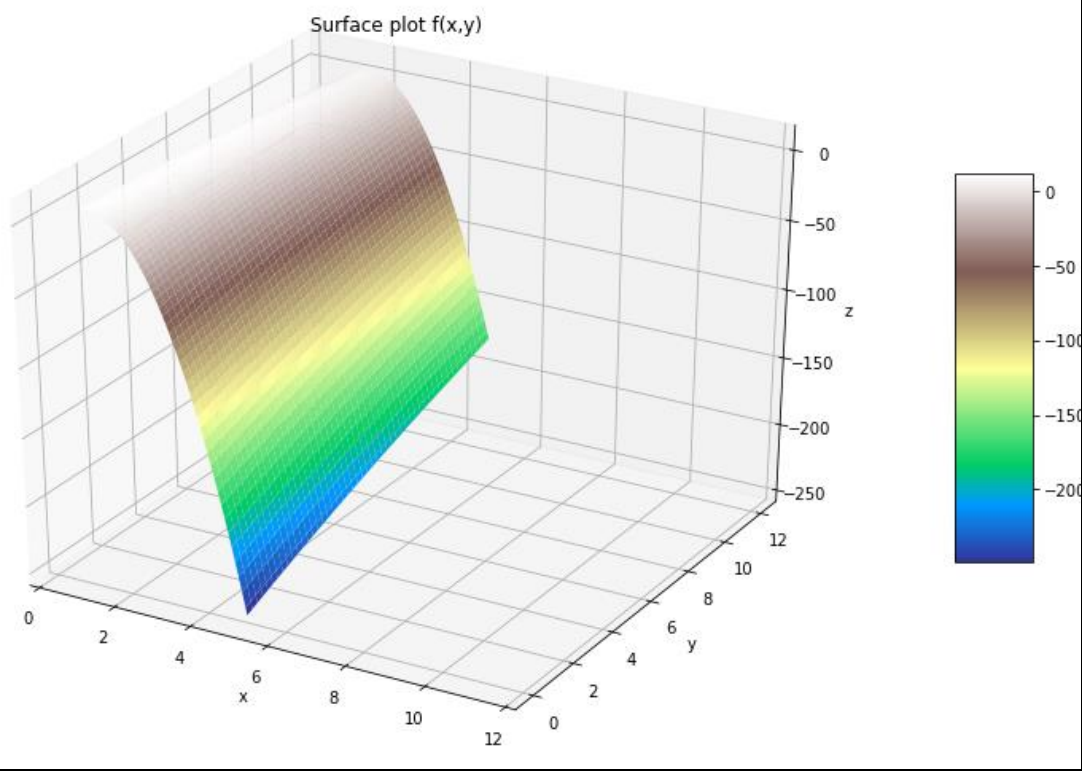
Now that a Genetic algorithm is available to us, optimization of actual functions can be carry out. The first example problem will be $f(x_0,x_1)=2x_0\ x_1+2x_0-x_0^2-2x_1^2+10$. This function was used in previous sub-sections of optimization, but an additional constant value of 10 is added to ensure positive cumulative numbers. Please remember that we do not have any dificulty solving this problem with previously given geometrical based optimization algorithms. Note that the result is probabilistic so that it will never be exactly the same like geometric methods.

Genetic class test program

```
# -*- coding: utf-8 -*-
"""
Created on Mon Nov  9 04:02:18 2020
```

```
@author: M. Turhan Çoban
"""
from f_xj import *
from Genetic import *
import  plot3D1 as P3D



class f1(f_xj):func=lambda self,x:2.0*x[0]*x[1]+2.0*x[0]-x[0]*x[0]-2.0*x[1]*x[1]+10.0
f=f1()

N=32 #number of digits in binary genes
POPSIZE=100 #population size
MAXGENS=50 # maximum number of generations
NVARS=2  #number of variables (member of genes in genome
PXOVER=0.3 #probability of crossover
PMUTATION=0.02 #probability of mutation
low1=[1.0,1.0]
high1=[5.0,5.0]
g=Genetic(f,low1,high1,POPSIZE,MAXGENS,NVARS,N,PXOVER,PMUTATION)
print(g.calculate())
xmin=low1[0]
xmax=high1[0]
Nx=100
ymin=0.0
ymax=12.0
Ny=100
PLabel="Optimisation: Genetic Algorithms"
XLabel="x"
YLabel="y"
ZLabel="z"
P3D.plot(f,xmin,xmax,Nx,ymin,ymax,Ny,PLabel,XLabel,YLabel,ZLabel)
```

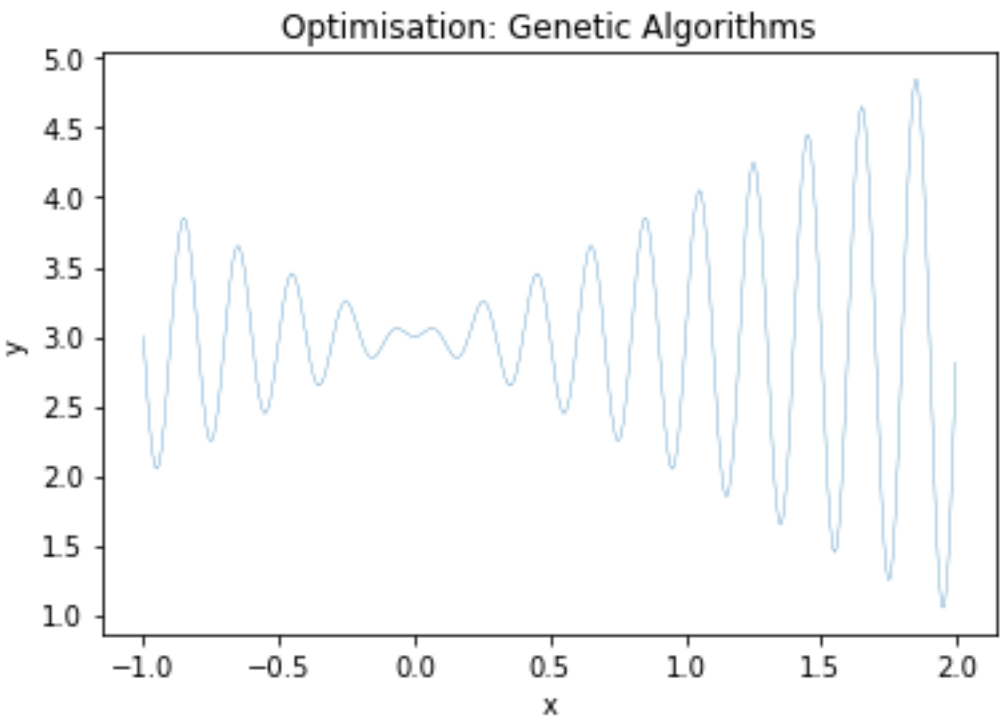
```
runfile('E:/okul/SCO1/Genetic_test2.py', wdir='E:/okul/SCO1')
Reloaded modules: f_xj, f_x, Gene, Genotype, Genetic
[2.1468761491465562, 1.059217885150392, 11.988809270831494]
```



As a second example a one dimensional function $f(x_0)=x_0 \sin(10\pi x_0)+3$ function will be investigated . The plot of the function shows the difficulty of this function: it has many local maximum. It is very hard for geometric methods to get a solution of such function unless some jump mechanisms built into the solution.

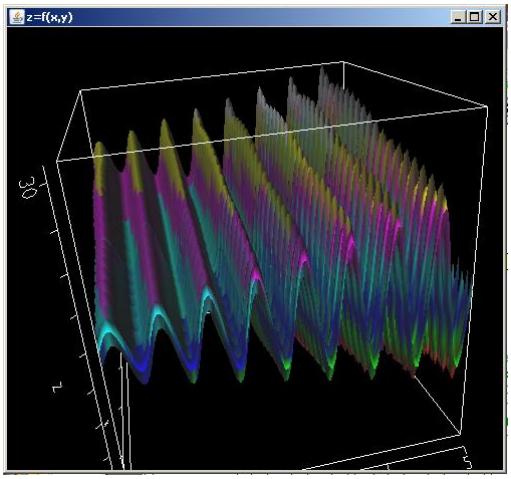
```
# -*- coding: utf-8 -*-
"""
Created on Mon Nov  9 04:02:18 2020
@author: M. Turhan Çoban
"""
from f_xj import *
from Genetic import *
import numpy as np;
import matplotlib.pyplot as plt;



class f1(f_xj):func=lambda self,x:x[0]*sin(10.0*pi*x[0])+3.0
f=f1()
class f2(f_x):func=lambda self,x:x*sin(10.0*pi*x)+3.0
ff=f2()
N=22 #number of digits in binary genes
POPSIZE=50 #population size
MAXGENS=50 # maximum number of generations
NVARS=1  #number of variables (member of genes in genome
PXOVER=0.3 #probability of crossover
PMUTATION=0.02 #probability of mutation
low1=[-1.0]
high1=[2.0]
g=Genetic(f,low1,high1,POPSIZE,MAXGENS,NVARS,N,PXOVER,PMUTATION)
print(g.calculate())

dx=3.0/1000.0
n=1000
y=[0 for z in range(n)];
x=[0 for z in range(n)];
for i in range(n):
    x[i]=-1.0+dx*i
    y[i]=ff.func(x[i])
plt.plot(x,y,linewidth=0.3)
plt.xlabel("x")
plt.ylabel("y")
plt.title('Optimisation: Genetic Algorithms')
plt.show()
```

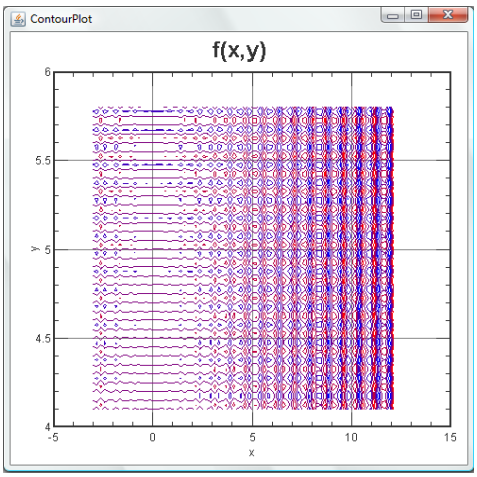
```
runfile('E:/okul/SCO1/Genetic_test.py', wdir='E:/okul/SCO1')
Reloaded modules: f_xj, f_x, Gene, Genotype, Genetic
[1.8501109719541007, 4.850099728660397]
```



As a last example of class Genetic1 a two variable nonlimear function is defined as
f(x,y)=21.5+x*sin(4.0*π*y)+y*Math.sin(20.0* π *y)

The 3D plot of the function



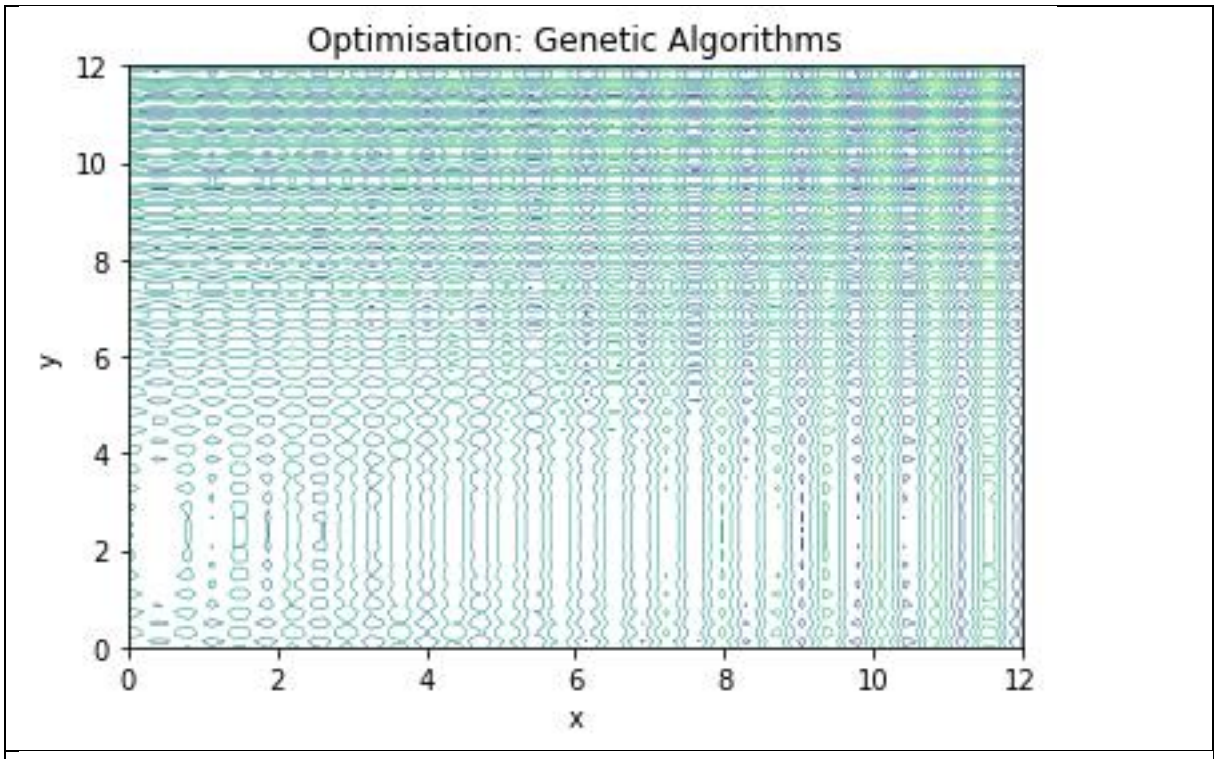


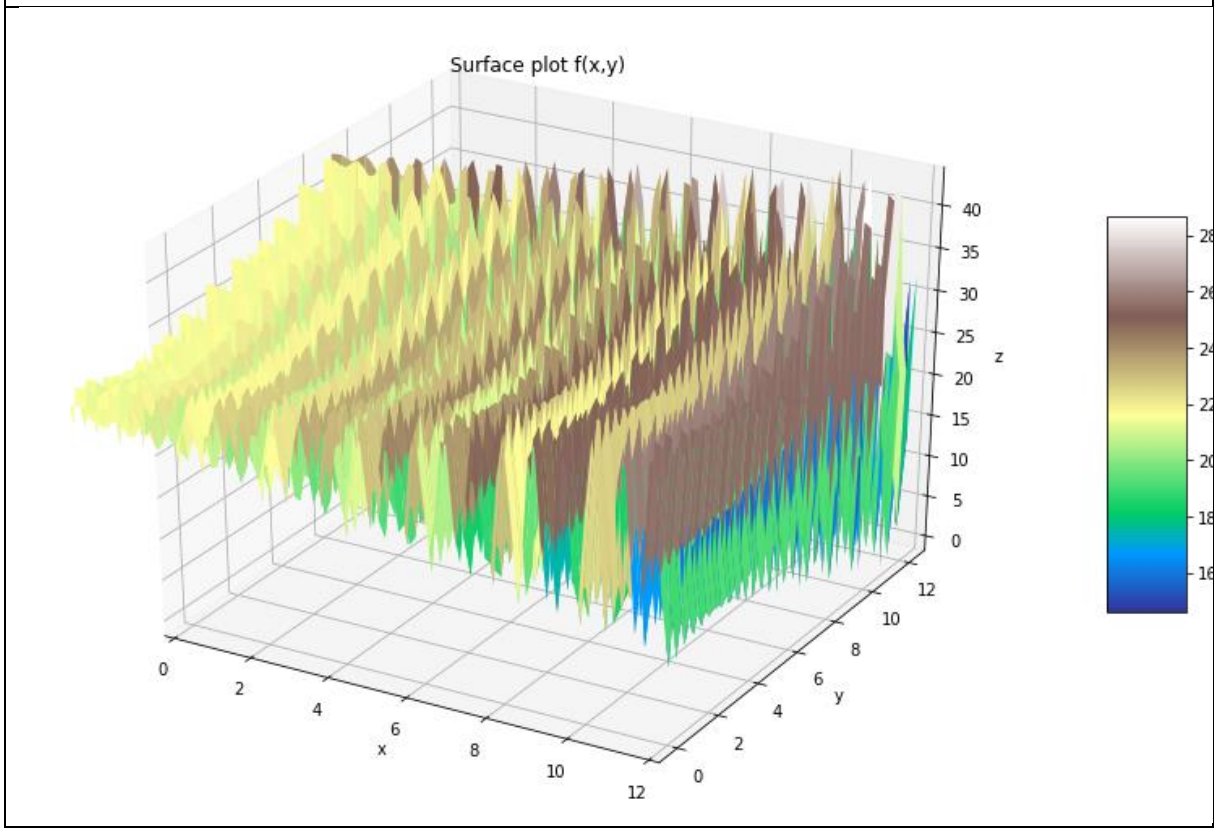
It is clear from the plot that this problem also have the same kind of difficulty as the previous problem.

Program 5.18.5-4 Genetic1 class test program NA106

```
 # -*- coding: utf-8 -*-
"""
Created on Mon Nov  9 04:02:18 2020
@author: M. Turhan Çoban
"""
from f_xj import *
from Genetic import *
import  plot3D1 as P3D



class f1(f_xj):func=lambda self,x:21.5+x[0]*sin(4.0*pi*x[0])+x[1]*sin(20.0*pi*x[1])
f=f1()

N=32 #number of digits in binary genes
POPSIZE=100 #population size
MAXGENS=50 # maximum number of generations
NVARS=2  #number of variables (member of genes in genome
PXOVER=0.3 #probability of crossover
PMUTATION=0.02 #probability of mutation
low1=[-3.0,4.1]
high1=[12.1,5.8]
g=Genetic(f,low1,high1,POPSIZE,MAXGENS,NVARS,N,PXOVER,PMUTATION)
print(g.calculate())
xmin=low1[0]
xmax=high1[0]
Nx=100
ymin=0.0
ymax=30.0
Ny=100
PLabel="Optimisation: Genetic Algorithms"
XLabel="x"
YLabel="y"
ZLabel="z"
P3D.plot(f,xmin,xmax,Nx,ymin,ymax,Ny,PLabel,XLabel,YLabel,ZLabel)
```

```
runfile('E:/okul/SCO1/Genetic_test1.py', wdir='E:/okul/SCO1')
Reloaded modules: f_xj, f_x, Gene, Genotype, Genetic
[11.634943575466737, 5.224075439251976, 38.259492805578425]
```





8.16.7 GENETİC ALGORITHMS BY USING A LONG TYPE VARIABLE AS A GENE

In a gene structure number of digits in the gene should be constant. It is similar in the natural system as well. In order to produce a child gene structure of both parents should be similar in structure . **A long integer with a setted number of digits** will be used as a gene structure in the next genetic algorithm code. In this two variable systems long type integer value bits can take all the possible values 0 to 9 in each digit, while double value can

only corresponds the working space. As a maximum long number 18 digit x'_max=999999999999999999 is taken. In this case the relation between long number x' and double number x can be given as:

$$x = x_{min} + x'\left(\frac{x_{max} - x_{min}}{999999999999999999}\right)$$

The double number can be converted back to the long integer gene number the same way.

$$x' = 999999999999999999\left(\frac{x - x_{min}}{x_{max} - x_{min}}\right)$$

The remaining of genetic algorithm is the same as the previous algorithms. It should be note that even though the number of the gens are constant (18) for this algorithm, gen number input variable is kept in the program constructor, in order to make it same as the previous versions with the variable gene numbers. Please note that 18 digit(of 10 numbers) is equal 90 digits(2 numbers) binary.

int_gene constructed from integer numbers of 18 digits.

```
 from abc import ABC, abstractmethod
from math import *
import random
from f_x import *


class int_gene(ABC):

    def __init__(self, xming,xmaxg):
        self.xming=xming
        self.xmaxg=xmaxg
        self.imax=999999999999999999
        self.xp=int(random.random()*self.imax)
        self.x=self.xming+self.xp*(self.xmaxg-self.xming)/self.imax
        self.xs=str(self.xp)
        self.N=18


    def rand_xp(self):
        self.xp=int(random.random()*self.imax)
    def rand_x(self):
        self.xp=int(random.random()*self.imax)
        self.x=self.xming+self.xp*(self.xmaxg-self.xming)/self.imax
    def mutate(self):
        n1=str(int(10.0*random.random()))
        nn=int(18*random.random())
        n2=str(self.xs[nn:nn+1])
        if(n1==n2 and n1==str(9)):
            n1=str(8)
        elif(n1==n2 and n1==str()):
            n1=str(1)
        elif(n1==n2):
            n1=str(int(n2)+1)
        self.xs=self.xs[:nn]+n1+self.xs[nn+1:]
        self.xp=int(self.xs)
        self.x=self.xming+self.xp*(self.xmaxg-self.xming)/self.imax

    def print(self):
        print("min=",self.xming,"max=",self.xmaxg,"xp=",self.xp,"x=",self.x)

    # seperated at point n
    def split(self,nn):
        #nn=int(random.random()*self.N)
        #print("nn=",nn)
        s1=self.xs[:nn]
        s2=self.xs[nn:]
        s=[s1,s2]
        return s

    # split string binary gene into three pieces
    # seperated at point n
    def split2(self,nn1,nn2):
        #nn=int(random.random()*self.N)
        #print("nn=",nn)
        if nn2>=nn1:
            s1=self.xs[:nn1]
            s2=self.xs[nn1:nn2]
            s3=self.xs[nn2:]
```

```
        else:
            s1=self.xs[:nn2]
            s2=self.xs[nn2:nn1]
            s3=self.xs[nn1:]
        s=[s1,s2,s3]
        return s
    # cross correlation  1 of two Genes
def crossing1(f1,f2,N):
    s1=f1.split(N)
    s2=f2.split(N)
    ss1=s1[0]+s2[1]
    ss2=s2[0]+s1[1]
    ff1=int(ss1)
    ff2=int(ss2)
    if random.random()<=0.5:
        ff=set_int_gene(f1.xming,f1.xmaxg,ff1)
    else:
        ff=set_int_gene(f1.xming,f1.xmaxg,ff2)
    return ff

# cross correlation2 of two Genes
def crossing2(f1,f2,N1,N2):
    s1=f1.split2(N1,N2)
    s2=f2.split2(N1,N2)
    ss1=s1[0]+s2[1]+s1[2]
    ss2=s2[0]+s1[1]+s2[2]
    ff1=int(ss1)
    ff2=int(ss2)
    if random.random()<=0.5:
        ff=set_int_gene(f1.xming,f1.xmaxg,ff1)
    else:
        ff=set_int_gene(f1.xming,f1.xmaxg,ff2)
    return ff

# crossing1 crossing2 or crossing3 is selceted randomly
def crossing(f1,f2):
        if int_random()==0:
            N=int(f1.N*random.random())
            fchild=crossing1(f1,f2,N)
        else:  # int_random()==1:
            N1=int(f1.N*random.random())
            N2=int(f1.N*random.random())
            if N1<=N2:
                fchild=crossing2(f1,f2,N1,N2)
            else:
                fchild=crossing2(f1,f2,N2,N1)
        return fchild

def set_int_gene(xming,xmaxg,xp):
    g=int_gene (xming,xmaxg)
    g.xp=xp
    g.x=g.xming+g.xp*(g.xmaxg-g.xming)/g.imax
    g.xs=str(xp)
    return g

#creates a random int 0 or 1
def int_random():
    r=0
    if random.random()>=0.5:
        r=1
    return r
```

A test for int_gene (mutation)

```
from int_gene import *

xmin=0.0
xmax=3.0
f1=int_gene(xmin,xmax)
f1.print()
f2=int_gene(xmin,xmax)
f2.print()
print("f1 mutation")
```

```
f1.mutate()
f1.print()
print("f2 mutation")
f2.mutate()
f2.print()
```

```
runfile('E:/okul/SCO1/int_gene_test.py', wdir='E:/okul/SCO1')
Reloaded modules: f_x, int_gene
min= 0.0 max= 3.0 xp= 3530173074102 96384 x= 1.0590519222308892
min= 0.0 max= 3.0 xp= 381159414778967552 x= 1.1434782443369027
f1 mutation
min= 0.0 max= 3.0 xp= 3530173074104 96384 x= 1.0590519222314891
f2 mutation
min= 0.0 max= 3.0 xp= 381158414778967552 x= 1.1434752443369027
```

A test for int_gene (mutation)

```
from int_gene import *

xmin=0.0
xmax=3.0
f1=int_gene(xmin,xmax)
f1.print()
f2=int_gene(xmin,xmax)
f2.print()
print("crossing of f1 and f2")
f3=crossing(f1,f2)
f3.print()
```

```
runfile('E:/okul/SCO1/int_gene_test1.py', wdir='E:/okul/SCO1')
Reloaded modules: f_x, int_gene
min= 0.0 max= 3.0 xp= 32066133037008 3840 x= 0.9619839911102516
min= 0.0 max= 3.0 xp= 946662408866570112 x= 2.8399872265997104
crossing of f1 and f2
min= 0.0 max= 3.0 xp= 9466624088665738 40 x= 2.8399872265997215
```

Integer Genotype int_Genotype

```
from abc import ABC, abstractmethod
from math import *
import random
from f_xj import *
from int_gene import *


class int_Genotype(ABC):

    def __init__(self, f,NVARS,low1,high1):
        self.xmin=low1
        self.xmax=high1
        self.NVARS=NVARS
        self.N=18
        self.f=f
        self.G=[0.0 for j in range(self.NVARS)]
        self.x=[0.0 for j in range(self.NVARS)]
        for i in range(self.NVARS):
            self.G[i]=int_gene(self.xmin[i],self.xmax[i])
            self.x[i]=self.G[i].x
        self.Fitness=self.f.func(self.x)
        self.RFitness=0.0
        self.CFitness=0.0
        self.Breeder=False
```

```
    def setGene(val):
        G[val]=int_gene(self.xmin[val],self.xmax[val])
        return self.G[val]

    def getGene(val):
        return G[val]

    #print gene values
    def print(self):
        for i in range(self.NVARS):
            self.G[i].print()
        print("Fitness = ",self.Fitness)
        print("RFitness = ",self.RFitness)
        print("CFitness = ",self.CFitness)

    def setFitness(self):
        for i in range(self.NVARS):
            self.x[i]=self.G[i].x
        self.Fitness=self.f.func(self.x)

    def getFitness(self):
        return self.Fitness

    def setRFitness(self,sum):
        self.RFitness=self.Fitness/sum

    def setCFitness(self,x):
        self.CFitness=x

    def getCFitness(self):
        return self.CFitness

    def breed(self):
        self.Breeder=True

    def not_breed(self):
        self.Breeder=False

    def isBreed(self):
        return self.Breeder

    def copygenotype(self):
        g1=self
        return g1
    def mutate(self,PMUTATION):
        for i in range(self.NVARS):
            a=random.random()
            if a<PMUTATION:
                self.G[i].mutate()
def crossingGT(G1,G2):
    nv=G1.NVARS
    G3=int_Genotype(G1.f,G1.NVARS,G1.xmin,G1.xmax)
    for i in range(nv):
        G3.G[i]=crossing(G1.G[i],G2.G[i])
    return G3
```

int_Genotype example program:

```
from f_xj import *
from  int_Genotype import *
```

```
N=32
xmin=[0.0,0.0]
xmax=[5.0,5.0]
class f1(f_xj):func=lambda self,x: x[0]*x[0]+x[1]*x[1]
f = f1()
NVARS=2
N=32
G1=int_Genotype(f,NVARS,xmin,xmax)
G2=int_Genotype(f,NVARS,xmin,xmax)
print("G1 = ")
G1.print()
print("G2 = ")
G2.print()
print("G3 breed")
G3=crossingGT(G1,G2)
G3.print()
print("G3 mutate")
G3.mutate(1.0)
G3.print()
```

```
runfile('E:/okul/SCO1/int_Genotype_test1.py', wdir='E:/okul/SCO1')
Reloaded modules: f_x, int_gene
G1 =
min= 0.0 max= 5.0 xp= 354311067151272704 x= 1.7715553357563636
min= 0.0 max= 5.0 xp= 296277299780326144 x= 1.4813864989016308
Fitness =  5.332914266774873
RFitness =  0.0
CFitness =  0.0
G2 =
min= 0.0 max= 5.0 xp= 222301085691244288 x= 1.1115054284562214
min= 0.0 max= 5.0 xp= 379478200327386432 x= 1.8973910016369322
Fitness =  4.835536930580449
RFitness =  0.0
CFitness =  0.0
G3 breed
min= 0.0 max= 5.0 xp= 354311067151272704 x= 1.7715553357563636
min= 0.0 max= 5.0 xp= 379477299327386432 x= 1.897386496636932
Fitness =  7.447702320452942
RFitness =  0.0
CFitness =  0.0
G3 mutate
min= 0.0 max= 5.0 xp= 354311067154272704 x= 1.7715553357713636
min= 0.0 max= 5.0 xp= 379477299377386432 x= 1.8973864968869323
Fitness =  7.447702320452942
RFitness =  0.0
CFitness =  0.0
```

Genetic algorithm constructed from long integer numbers of 18 digits. int_Genetic

```
 from abc import ABC, abstractmethod
from math import *
import random
from f_xj import *
from int_Genotype import *


class int_Genetic(ABC):
    def __init__(self,f,low,high,POPSIZE,MAXGENS,NVARS,PXOVER,PMUTATION):
        # function derived from f_xj class f(x0,x1,..xn)
        self.POPSIZE=POPSIZE   # population size
        self.MAXGENS=MAXGENS   # maksimum number of generations
        self.NVARS=NVARS    # number of variables
        #number of independent variable of function
```

```
        #number of Genes in genotype
        self.N=18
        #number of bits in a gene
        self.PXOVER=PXOVER
        #probability of cross-over
        self.PMUTATION=PMUTATION
        #proportion of mutated variables
        self.NBreeder=int(0.8*POPSIZE)
        #number of survivers in a generation
        self.Best=0
        self.f=f
        self.xmin=[0.0 for j in range(self.NVARS)]
        self.xmax=[0.0 for j in range(self.NVARS)]
        for i in range(self.NVARS):
            self.xmin[i]=low[i]
            self.xmax[i]=high[i]
        #Best genotype in the population
        self.Population=[0.0 for j in range(self.POPSIZE+1)]
        self.setPopulation()
        self.setRFCF()
        self.evaluate()

    def print(self):
        print("Population size = ",self.POPSIZE)
        print("Maximum number of Generations = ",self.MAXGENS)
        print("Number of Bits in Gene = ",self.N)
        print("Probability of crossover =",self.PXOVER)
        print("Propertion of mutated variables",self.PMUTATION)
        for i in range(self.POPSIZE+1):
            self.Population[i].print()


    def setPopulation(self):
        for i in range(self.POPSIZE+1):
            self.Population[i]=int_Genotype( self.f,self.NVARS,self.xmin,self.xmax)
            self.Population[i].setFitness()
        self.setRFCF()
        self.evaluate()


    def setRFCF(self):
        sum=0.0
        for i in range(self.POPSIZE):
            self.Population[i].setFitness()
            sum=sum+self.Population[i].Fitness

        for i in range(self.POPSIZE):
            self.Population[i].setRFitness(sum)
        sum=0.0
        for i in range(self.POPSIZE):
            sum=sum+self.Population[i].RFitness
            self.Population[i].setCFitness(sum)

    def evaluate(self):
        for i in range(self.POPSIZE):
            if self.Population[i].Fitness>self.Population[self.POPSIZE].Fitness:
                self.Population[self.POPSIZE]=self.Population[i]

    def breding_individual(self):
        #select members for breeding
        j=0
        for i in range(self.POPSIZE):
            self.Population[i].not_breed()
        for i in range(self.POPSIZE):
            r=random.random()
            if i==0 and r<self.Population[i].CFitness:
                j=i
            elif  r>=self.Population[i-1].CFitness and  r<=self.Population[i].CFitness:
                j=i
        return j
```

```
    def crossover(self):
        for mem in range(self.POPSIZE):
            POne= int(self.breding_individual())
            PTwo=int(self.breding_individual())
            Po=[0.0 for j in range(2)]
            Po[0]=self.Population[POne]
            Po[1]=self.Population[PTwo]
            self.Population[POne]=crossingGT(Po[0],Po[1])
            self.Population[PTwo]=crossingGT(Po[0],Po[1])
            self.Population[POne].mutate(self.PMUTATION)
            self.Population[PTwo].mutate(self.PMUTATION)
            self.setRFCF()
            self.evaluate()

    def getBest(self):
        aa=[0.0 for j in range(self.NVARS+1)]
        for i in range(self.NVARS):
            aa[i]=self.Population[self.POPSIZE].x[i]
        aa[self.NVARS]=self.Population[self.POPSIZE].Fitness
        return aa

    def select(self):
        Po=[0.0 for j in range(self.POPSIZE+1)]
        for i in range(self.POPSIZE+1):
            mem=0
            r=random.random()
            gf1=self.Population[i].getCFitness()
            gf2=0.0
            if gf1>r:
                Po[i]=self.Population[mem].copygenotype()
            for mm in range(1,self.POPSIZE):
                gf2=self.Population[mem].getCFitness()
                if gf2>r and gf1<r:
                    Po[i]=self.Population[mem].copygenotype()
                    break
        for i in range(self.POPSIZE):
            self.Population[i]=Po[i]
        self.setRFCF()
        self.evaluate()

    def calculate(self):
         self.evaluate()
         Generation=0
         while Generation<self.MAXGENS:
             Generation=Generation+1
             self.crossover()
             self.evaluate()
         return self.getBest()

def setRFCF(Co):
    sum=0.0
    for i in range(Co.POPSIZE+1):
        Co.Population[i].setFitness()
        sum=sum+Co.Population[i].Fitness
    for i in range(Co.POPSIZE+1):
        Co.Population[i].setRFitness(sum)
        sum=0
        for i in range(Co.POPSIZE+1):
            sum=sum+Co.Population[i].RFitness
            Co.Population[i].setCFitness(sum)
```

Test program for Gene constructed from integer numbers of 18 digits.

```
 # -*- coding: utf-8 -*-
"""
Created on Mon Nov  9 04:02:18 2020
@author: M. Turhan Çoban
"""
```

```
from f_xj import *
from int_Genetic import *
import numpy as np;
import matplotlib.pyplot as plt;



class f1(f_xj):func=lambda self,x:x[0]*sin(10.0*pi*x[0])+3.0
f=f1()
class f2(f_x):func=lambda self,x:x*sin(10.0*pi*x)+3.0
ff=f2()
N=22 #number of digits in binary genes
POPSIZE=50 #population size
MAXGENS=50 # maximum number of generations
NVARS=1  #number of variables (member of genes in genome
PXOVER=0.3 #probability of crossover
PMUTATION=0.02 #probability of mutation
low1=[-1.0]
high1=[2.0]
g=int_Genetic(f,low1,high1,POPSIZE,MAXGENS,NVARS,PXOVER,PMUTATION)
print(g.calculate())

dx=3.0/1000.0
n=1000
y=[0 for z in range(n)];
x=[0 for z in range(n)];
for i in range(n):
    x[i]=-1.0+dx*i
    y[i]=ff.func(x[i])
plt.plot(x,y,linewidth=0.3)
plt.xlabel("x")
plt.ylabel("y")
plt.title('Optimisation: Genetic Algorithms')
plt.show()
```

```
# -*- coding: utf-8 -*-
"""
Created on Mon Nov  9 04:02:18 2020
@author: M. Turhan Çoban
"""
from f_xj import *
from int_Genetic import *
import numpy as np;
import matplotlib.pyplot as plt;



class f1(f_xj):func=lambda self,x:x[0]*sin(10.0*pi*x[0])+3.0
f=f1()
class f2(f_x):func=lambda self,x:x*sin(10.0*pi*x)+3.0
ff=f2()
N=22 #number of digits in binary genes
POPSIZE=100 #population size
MAXGENS=100 # maximum number of generations
NVARS=1  #number of variables (member of genes in genome
PXOVER=0.3 #probability of crossover
PMUTATION=0.02 #probability of mutation
low1=[-1.0]
high1=[2.0]
g=int_Genetic(f,low1,high1,POPSIZE,MAXGENS,NVARS,PXOVER,PMUTATION)
print(g.calculate())

dx=3.0/1000.0
n=1000
y=[0 for z in range(n)];
x=[0 for z in range(n)];
for i in range(n):
    x[i]=-1.0+dx*i
    y[i]=ff.func(x[i])
plt.plot(x,y,linewidth=0.3)
plt.xlabel("x")
```

```
plt.ylabel("y")
plt.title('Optimisation: Genetic Algorithms')
plt.show()
```

```
runfile('E:/okul/SCO1/int_Genetic_test.py', wdir='E:/okul/SCO1')
Reloaded modules: f_xj, f_x, int_gene, int_Genotype, int_Genetic
[1.8502416190695525, 4.85018831522604]
```

```
# -*- coding: utf-8 -*-
"""
Created on Mon Nov  9 04:02:18 2020
@author: M. Turhan Çoban
"""
from f_xj import *
from int_Genetic import *
import  plot3D1 as P3D



class f1(f_xj):func=lambda self,x:2.0*x[0]*x[1]+2.0*x[0]-x[0]*x[0]-2.0*x[1]*x[1]+10.0
f=f1()

N=18 #number of digits in integer genes
POPSIZE=100 #population size
MAXGENS=50 # maximum number of generations
NVARS=2  #number of variables (member of genes in genome
PXOVER=0.3 #probability of crossover
PMUTATION=0.02 #probability of mutation
low1=[1.0,1.0]
high1=[5.0,5.0]
g=int_Genetic(f,low1,high1,POPSIZE,MAXGENS,NVARS,PXOVER,PMUTATION)
print(g.calculate())
xmin=low1[0]
xmax=high1[0]
Nx=100
ymin=0.0
ymax=12.0
Ny=100
PLabel="Optimisation: Genetic Algorithms"
XLabel="x"
YLabel="y"
ZLabel="z"
P3D.plot(f,xmin,xmax,Nx,ymin,ymax,Ny,PLabel,XLabel,YLabel,ZLabel)
```

```
runfile('E:/okul/SCO1/int_Genetic_test2.py', wdir='E:/okul/SCO1')
Reloaded modules: f_xj, f_x, int_gene, int_Genotype, int_Genetic
[2.0299727032780286, 1.0563244328352681, 11.996133144615126]
```

```
# -*- coding: utf-8 -*-
"""
Created on Mon Nov  9 04:02:18 2020
@author: M. Turhan Çoban
"""
from f_xj import *
from int_Genetic import *
import  plot3D1 as P3D



class f1(f_xj):func=lambda self,x:21.5+x[0]*sin(4.0*pi*x[0])+x[1]*sin(20.0*pi*x[1])
f=f1()
```

```
N=18 #number of digits in binary genes
POPSIZE=100 #population size
MAXGENS=50 # maximum number of generations
NVARS=2  #number of variables (member of genes in genome
PXOVER=0.3 #probability of crossover
PMUTATION=0.02 #probability of mutation
low1=[-3.0,4.1]
high1=[12.1,5.8]
g=int_Genetic(f,low1,high1,POPSIZE,MAXGENS,NVARS,PXOVER,PMUTATION)
print(g.calculate())
xmin=low1[0]
xmax=high1[0]
Nx=100
ymin=0.0
ymax=12.0
Ny=100
PLabel="Optimisation: Genetic Algorithms"
XLabel="x"
YLabel="y"
ZLabel="z"
P3D.plot(f,xmin,xmax,Nx,ymin,ymax,Ny,PLabel,XLabel,YLabel,ZLabel)
P3D.contour_plot(f,xmin,xmax,Nx,ymin,ymax,Ny,PLabel,XLabel,YLabel,ZLabel)
```

```
runfile('E:/okul/SCO1/int_Genetic_test1.py', wdir='E:/okul/SCO1')
Reloaded modules: f_xj, f_x, int_gene, int_Genotype, int_Genetic, plot3D1
[12.08911500264852, 4.4257205915714986, 36.80182408286153]
```

8.17 STOCHASTIC METHODS: MONTE-CARLO METHOD

Monte carlo optimization as a method is very simple. Points between the given limits are selected randomly and evaluated. When number of points increase to very high numbers, statistically there will be a homogeneous coverege of all the search region. Record the best optimum found so far.What is achived is similar to plotting data, but in monte carlo method taken constant steps and walk through the steps are not needed because randomness takes care of uniformity of search when the total number is big enough.

```
# Dr. M. Turhan ÇOBAN
#
# EGE University School of Engineering, Mechanical Engineering Department
# package : Numerical analysis package
# subject : optimization
# group   : Monte-Carlo optimization(maximum) Algorithms
# monte_carlo_opt Class
# class selects random points at the given area and selects the best
# results found

from abc import ABC, abstractmethod
from math import *
import random
from f_xj import *
from Genotype import *

class monte_carlo_opti(ABC):
    def __init__(self,Ni,imin,imax):
        n=len(imin)
        self.max=[0.0 for j in range(n)]
        self.min=[0.0 for j in range(n)]
        self.N=Ni
        for i in range(n):
            self.min[i]=imin[i]
            self.max[i]=imax[i]

    def monte_carlo_max(self,fi):
        n=len(self.min)
        xmax=[0.0 for j in range(n+1)]
        x=[0.0 for j in range(n)]
```

```
        max_number=-1.0e99
        f=0.0
        for k in range(self.N):
            for i in range(n):
                x[i]=self.min[i]+(self.max[i]-self.min[i])*random.random()
            f=fi.func(x)
            if f>max_number:
                max_number=f
                for i in range(n):
                    xmax[i]=x[i]
                xmax[n]=f
        return xmax
```

Sample test program:

```
# -*- coding: utf-8 -*-
"""
Created on Mon Nov  9 04:02:18 2020
@author: M. Turhan Çoban
"""
from math import *
from f_xj import *
from monte_carlo_opti import *
import  plot3D1 as P3D




class f1(f_xj):func=lambda self,x:x[0]*sin(10.0*pi*x[0])+3.0
f=f1()

low1=[-1.0]
high1=[2.0]
mco=monte_carlo_opti(100000,low1,high1)
print(mco.monte_carlo_max(f))
```

```
runfile('E:/okul/SCO1/monte_carlo_opti_test.py', wdir='E:/okul/SCO1')
Reloaded modules: f_xj, f_x, Gene, Genotype
[1.8505509890910292, 4.850273755428534]
```

Sample test program:

```
# -*- coding: utf-8 -*-
"""
Created on Mon Nov  9 04:02:18 2020
@author: M. Turhan Çoban
"""
from math import *
from f_xj import *
from monte_carlo_opti import *
import  plot3D1 as P3D

class f1(f_xj):func=lambda self,x:1.0+cos(pi*(x[0]-3.0))*cos(2.0*pi*(x[1]-2.0))/(1+(x[0]-3.0)*(x[0]-3.0)+(x[1]-2.0)*(x[1]-2.0));
f=f1()

low1=[-10.0,-10.0]
high1=[10.0,10.0]
mco=monte_carlo_opti(1000000,low1,high1)
print(mco.monte_carlo_max(f))
```

```
runfile('E:/okul/SCO1/monte_carlo_opti_test1.py', wdir='E:/okul/SCO1')
Reloaded modules: f_xj, f_x, Gene, Genotype, monte_carlo_opti, plot3D1
[3.012231283238844, 2.006510252512715, 1.9982342941634572]
```

8.18 STOCHASTIC METHODS: FIREFLY ALGORITHM

You might notice amazing flashing lights of fireflys in the summer nights. The flashing light is produced by a

process of bioluminescence. The true main purpose of such flashes are still on debate but one of the fundamental function is attracting mate. The rithmic flash, the rate of flashing and the amount of time form part of the signal system that brings both sexes together. We know that the light intensity at a particular distance from the light source obeys the inverse square law. That is to say, the light intensity decreases as the distance increases. Furthermore, the air absorbs light which becomes weaker and weaker as the distance increases. This physical factors are utilised by fireflies to find each other. The same principle can be utilised to create an artificial firefly for optimisation .The Firefly Algorithm was developed by Xin-She Yang [52], based on the idealization of the flashing characteristics of fireflies. In creating artificial firefly it is assumed that they are unisex, i.e. any firefly in the population can be attracted to any other firefly, Attractiveness is function of brightness. Brightness will decrease as a function of distance between. The less brighter one will move through the brighter one. If there are no brighter fire flies around, it will fly randomly. The brightness of the artificial firefly will directly be determined by objective function value. By using these principles, a pseudocode of the firefly algorithm can be written as:

Objective function $f(X)$, $X = (x_0, ..., x_n)$
Generate initial population of fireflies X^i $(i = 1,2, ...,m)$ *randomly*
Light intensity Ii at x≫ is determined by $f(X^i)$
Define light absorption coefficient γ
while $(t$ <MaxGeneration)
for $i = 1 : n$ all n fireflies
    for $j = 1 : n$ all n fireflies (inner loop)
        if $(Ii < Ij)$, Move firefly i towards j; end i f
        Vary attractiveness with distance r as

$$\beta = \beta_0 e^{-\gamma d^2} \text{ or } \beta = \frac{\beta_0}{1+\gamma d^2}$$

$$d_{ij} = \sqrt{\sum_{k=1}^{n} (x_{i,k} - x_{i,k})^2}$$

        Evaluate new solutions and update light intensity

$$X^i = (1-\beta)X^i + \beta X^j + \alpha\,(gaussrandom() - \frac{1}{2})$$

    end for j
end for i
Rank the fireflies and find the current global best *gt*
end while
Postprocess results and visualization

```
 from math import *
from f_xj import *
from matplotlib import pyplot as plt
import numpy as np
from random import *

"""
class f1(f_xj):
  def func(self,x):
     y=(1.0-x[0])*(1.0-x[0])+100.0*(x[1]-x[0]*x[0])*(x[1]-x[0]*x[0])
     return y;
"""
class fireflyi():
# double gamma,alpha,beta;
# double alpha0,alphan;
# int m;
# int n;
# m number of fireflies
# n number of variables for each fireflyi
# I0 light intensity at source
# gamma absorbtion coefficient
# alpha size of the random step
# beta attractiveness
# double x[][];
# double I0[];
# double ymin;
```

```
# double xmin[];
# int imin;
# int ngeneration;
    def __init__(self,ff,mi, xxmin,xxmax,gammai,alpha0i,alphani,ngenerationi):
        #initial population set
        self.n=len(xxmin) #number of objective function
        self.m=mi
        print("m=",self.m,"n=",self.n)
        self.x= [[0 for j in range(self.n)] for i in range(self.m+1)]
        self.I0=[0 for j in range(self.m+1)]
        self.xmin=[0 for i in range(self.n)]
        #x=new double[m+1][n];
        #I0=new double[m+1];
        #xmin=new double[n];
        self.gamma=gammai
        self.alpha0=alpha0i
        self.alphan=alphani
        self.alpha=alpha0i
        self.ngeneration=ngenerationi
        for j in range(self.n):
            self.xmin[j]=xxmin[j]+random()*(xxmax[j]-xxmin[j])
        self.ymin=ff.func(self.xmin)
        #end of init
        l=0;
        d=0.0;
        t=0.0;
        while l<ngeneration:
            # find minimum of population
            for i in range(self.m):
                for j in range(self.n):
                    self.x[i][j]=xxmin[j]+random()*(xxmax[j]-xxmin[j])
                self.I0[i]=ff.func(self.x[i])
                if self.I0[i]<self.ymin:
                    self.ymin=self.I0[i]
                    self.imin=i
                    for k in range(self.n):
                        self.xmin[k]=self.x[i][k]
            for j in range(self.n):
                self.x[m][j]=self.xmin[j]
            self.I0[m]=self.ymin
            l=l+1
            fi=0.0
            fj=0.0
            for i in range(self.m+1):
                for j in range(self.n):
                    if self.I0[i]<self.I0[j]:
                        self.beta=self.attractiveness(i,j)
                        for k in range(self.n):
                            #The best one will remain the rest will change
                            if i!=self.imin:
                                self.x[i][k]=(1.0-self.beta)*self.x[i][k]+self.beta*self.x[j][k]+self.alpha*(random()-0.5)
                    else:
                        for k in range(self.n):
                            if i!=self.imin:
                                self.x[i][k]=self.x[i][k]+self.alpha*(random()-0.5)
            #alpha will change by time
            #shrink to a smaller value
            self.alpha=self.alphan+(self.alpha0-self.alphan)*exp(-t)
            t=0.1*l;


    def distance(self,i,j):
        d=0
        for k in range(self.n):
            d+=(self.x[i][k]-self.x[j][k])*(self.x[i][k]-self.x[j][k])
        d=sqrt(d)
        return d

    def attractiveness(self,i,j):
        d=self.distance(i,j)
        #return self.I0[i]/(1+self.gamma*d*d)
        return self.I0[i]*exp(-self.gamma*d*d)

class f1(f_xj):func=lambda self,x:(1.0-x[0])*(1.0-x[0])+100.0*(x[1]-x[0]*x[0])*(x[1]-x[0]*x[0])
f=f1();
xmin=[0.0,0.0];
```

```
xmax=[2.0,2.0];
m=500
gamma=1.0
alpha0=0.5
alphan=0.01
ngeneration=1000
r1= fireflyi(f,m, xmin,xmax,gamma,alpha0,alphan,ngeneration)
z=r1.xmin;
print("xmin = ",z,"fmin=",f.func(z));
```

```
runfile('E:/okul/SCO1/fireflyi.py', wdir='E:/okul/SCO1')
Reloaded modules: f_xj
m= 500 n= 2
xmin =  [0.9990640938391031, 0.9986806719748089] fmin= 3.130310041809811e-05
```

8.19 STOCHASTIC METHODS: PARTICLE SWARM ALGORITHM

Particle swarm optimization (PSO) was developed by Kennedy and Eberhart in 1995, based on swarm behaviour such as fish and bird schooling in nature. PSO searches the space of an objective function by adjusting the trajectories of individual agents, called particles. The movement of the particles has both stochastic and deterministic elements. Each particle is attracted toward the best location of the given time and the best location of the near past (a current global). It has also tendency to move randomly. When a particle finds a location that is better than any previously found locations, then it updates it as the new current best for particle *i*.It also remembers all the currents best values to update a current global value. The aim is to find the global best among all the current best values available until no more improvements are available.

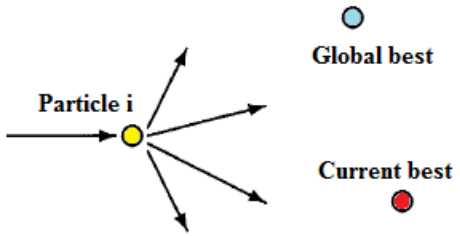


Figure 5.23.1 Particle will attract toward the global best, current best and also move randomly

The method is given in Table 5.23.1 as pseodocode. If x_i and v_i position vector and velocity for particle i, the new velocity vector is determined as:

$$v_i^{t+1} = v_i^t + \alpha^* \varepsilon_1^* (g^* - x_i^t) + \beta^* \varepsilon_2^* (x_i^* - x_i^t)$$

Where ε_1 and ε_1 are two random vectors between 0 and 1, g^* is the current global best x_i^* is local best. The parameters α and β are the learning parameters or acceleration constants. The initial velocity of the particles can be taken as zero, $v_i^{t=0} = 0$. The new position can then be updated by $x_i^{t+1} = x_i^t + v_i^{t+1}$. Although v_i can be any values, it is bounded in some range $[0,v_{max}]$.

There are many variants of the standart PSO algorithm. The most noticiable one is the version in which previous velocity function multiplied with an inertia function. This is called an accelerated PSO.

$$v_i^{t+1} = \theta v_i^t + \alpha^* \varepsilon_1^* (g^* - x_i^t) + \beta^* \varepsilon_2^* (x_i^* - x_i^t)$$

This is equivalent to introduce a virtual mass to stabilise the motion of the particles.

Table 5.23.1 Particle swarm algorithm

Objective function $f(P_i)$, $P_i = (x_0, ...,x_m)_i$ for particle i and m space dimension variable
Initialize locations P_i and velocity V_i of n particles by random selection Use as velocity vector: $V_i^{t+1} = \theta V_i^t + \alpha^* random()[GPmin - x_i^t] + \beta^* random()[Pmin - x_i^t]$ Where θ is inertia function(an iterative variable between 0 and 1) α and β are learning parameters or acceleration constants $\alpha = \alpha_0 \gamma^t$ where $0 \le \gamma \le 1$ $\alpha_0 \cong 0.5..1$ for t is the virtual time Find Global best GPmin from min{ $f(P_0)\ldots f(P_n)$} (at t = 0)
while (criterion)

t = t + 1 (pseudo time or iteration counter)
for loop over all n particles and all m dimensions
Generate new velocity V_i^{t+1}
Calculate new locations $P_i^{t+1} = P_i^t + V_i^{t+1}$
Evaluate objective functions at new locations
Find the current best for each particle Pmin
Find the global best for each particle GPmin
end for
end while
Output the final results Pmin and GPmin

```
 from math import *
from f_xj import *
from matplotlib import pyplot as plt
import numpy as np
from random import *


class particle_swarm():

#class variables
#int m,n
#double alpha,beta,beta0,teta,gamma
#m number of population
#n number of variables for each individual
#double x[][];  location for each population member
#double y[];    function value of each population member
#double yxmin;  current best minimum
#double ygmin;  global best minimum
#double xmin[]; current best x
#double gmin[]; global best x
#double v[][];  velocity vector
#int ngeneration;  number of generations in iteration


    def __init__(self,ff,mi, xxmin,xxmax,alphai,betai,ngenerationi):
        #initial population set
        self.teta=0.5
        self.gamma=0.5
        #initial population set
        self.n=len(xxmin);
        self.m=mi;
        self.alpha=alphai
        self.beta0=betai
        self.beta=self.beta0
        self.ngeneration=ngenerationi
        self.x= [[0 for j in range(self.n)] for i in range(self.m)]
        self.v= [[0 for j in range(self.n)] for i in range(self.m)]
        self.y=[0 for i in range(self.m)]
        self.xmin=[0 for i in range(self.n)]
        self.gmin=[0 for i in range(self.n)]
        self.ngeneration=ngenerationi
        yxmin=1.0e50
        for i in range(self.m):
            for j in range(self.n):
                self.x[i][j]=xxmin[j]+random()*(xxmax[j]-xxmin[j])
                self.v[i][j]=random()
            self.y[i]=ff.func(self.x[i]);
            if self.y[i]<yxmin:
                yxmin=self.y[i]
                ygmin=self.y[i]
                for k in range(self.n):
                    self.xmin[k]=self.x[i][k]
                    self.gmin[k]=self.x[i][k]
        t=0
        while t<self.ngeneration:
            yxmin=1.0e50
            for i in range(self.m):
                for j in range(self.n):
                    self.x[i][j]=self.x[i][j]+self.v[i][j]
                    if self.x[i][j]<xxmin[j]:
```

```
                self.x[i][j]=xxmin[j]
            if self.x[i][j]>xxmax[j]:
                self.x[i][j]=xxmax[j]
            self.y[i]=ff.func(self.x[i])
            if self.y[i]<yxmin:
                yxmin=self.y[i]
                for k in range(self.n):
                    self.xmin[k]=self.x[i][k]
        for i in range(self.m):
            if self.y[i]<ygmin:
                ygmin=self.y[i]
                for k in range(self.n):
                    self.gmin[k]=self.x[i][k]
        for i in range(self.m):
            for j in range(self.n):
                self.v[i][j]=self.teta*self.v[i][j]+self.alpha*random()*(self.gmin[j]-self.xmin[j])+self.beta*random()*(self.xmin[j]-
self.x[i][j])
        t=t+1
        self.beta=self.beta0*pow(self.gamma,0.01*t)
```

```
from math import *
from f_xj import *
from matplotlib import pyplot as plt
import numpy as np
from random import *
import particle_swarm as ps


class f1(f_xj):func=lambda self,x:(1.0-x[0])*(1.0-x[0])+100.0*(x[1]-x[0]*x[0])*(x[1]-x[0]*x[0])
f=f1();
xmin=[0.0,0.0];
xmax=[2.0,2.0];
m=500
alpha=2.0
beta=0.7
ngeneration=500

r1= ps.particle_swarm(f,m,xmin,xmax,alpha,beta,ngeneration)
z=r1.xmin;
print("xmin = ",z,"fmin=",f.func(z));
```

```
runfile('E:/okul/SCO1/particle_swarm2.py', wdir='E:/okul/SCO1')
Reloaded modules: f_xj, particle_swarm
xmin =  [1.0, 1.0] fmin= 0.0
```

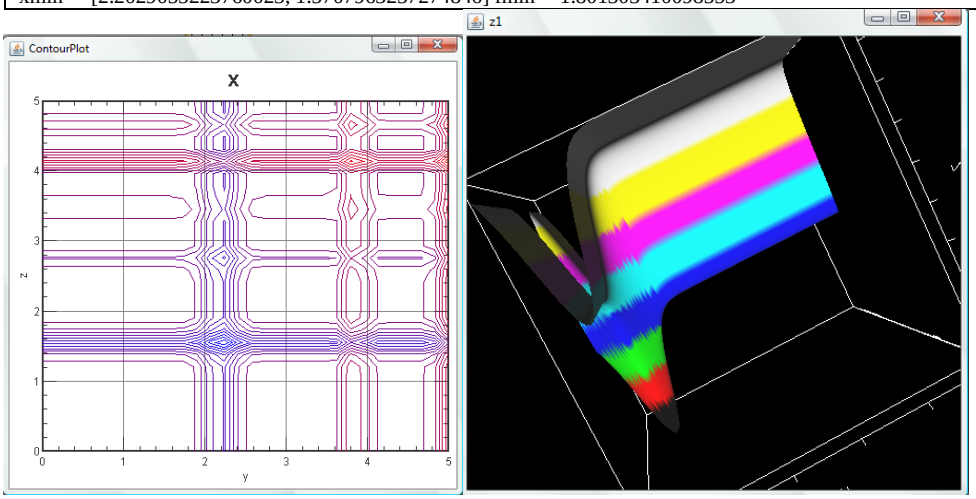
As another test function Michaelewicz function is given

$$f(x) = -\sum_{i=0}^{n}\left[\sin(x_i)\left[\sin\left(\frac{(i+1)x_i^2}{\pi}\right)\right]^{2m}\right]$$

```
from math import *
from f_xj import *
from matplotlib import pyplot as plt
import numpy as np
from random import *
import particle_swarm as ps

class f1(f_xj):
  def func(self,x):
      #Michaelewicz function
      ff=0.0
      n=len(x)
      for i in range(n):
          ff=ff-sin(x[i])*pow(sin((i+1)*x[i]*x[i]/pi),20)
      return ff

f=f1();
xmin=[0.0,0.0];
xmax=[2.5,2.5];
m=500
alpha=2.0
beta=0.7
ngeneration=500
```

```
r1= ps.particle_swarm(f,m,xmin,xmax,alpha,beta,ngeneration)
z=r1.xmin;
print("xmin = ",z,"fmin=",f.func(z))
```

```
runfile('E:/okul/SCO1/particle_swarm1.py', wdir='E:/okul/SCO1')
Reloaded modules: f_xj
xmin =  [2.2029055223760023, 1.5707963257274846] fmin= -1.801303410098553
```



8.20 STOCHASTIC METHODS: CUCKOO SEARCH ALGORITHM

This algorithm which is introduced by Yang and Deb[56] is constructed on brood parasitism of the cuckoos. Brood parasitism is a evoltionary strategy for cuckoos which is accomplished by laying their eggs in other kind of birds. If this alien egg is found by host bird, the nest(a place where eggs are hold) will be rejected or this egg will be thrown away. This behaviour is named as a evolutionary phenomena to mimic the egg apperance of local host birds. Regarding to this concept, following rules are employed .

- Cuckoos are allowed to lay one egg at a specific time, and drops it in a random nest
- Nest that contains high quality of eggs (nets with best fitness values) will carry over to next generation.
- The number of nest is fixed and a host bird can find an alien egg with a specified probability $p_a \in [0,1]$. If it happens, eggs which are dropped in nests will be thrown away or present nest will be built another place.

With this assumptions Cuckoo Search algorithm is developed . In order to carry out this algorithm effectively, Levy Flights are used to perform local and global searches in solution domain. Levy flights are kind of a random walk and simulated as foraging path of an animal and its next movement is based on its previous one. This pattern is also appliciable for natural phenomenas as biology, astronomy and physics. Behaviour of Levy Flight is proposed by

$$x_i^{t+1} = x_i^t + \alpha\theta Levy(\lambda)$$

Θ is donated by entery-wise multiplication, α is a step size and Lévy(λ) is Levy distribution. With the help of Levy distribution, displaced egg's position will be updated if this new position is better than the former one. According to literature survey, α value can be set to 1.0 . And also p_a parameter is suggested to be used as 0.25. Cuckoo Search algorithm pseudocode is given below

Cuckoo Search algorithm

```
"""//////////////////////////////////////////////////////////////////////
// Enginering Optimisation by Cuckoo Search                          //
//                                                                   //
//          Xin-She Yang          Suash Deb                          //
//////////////////////////////////////////////////////////////////////
// Programmed in Java language by : Oğuz Emrah Turgut
// Translated into python by : M. Turhan ÇOBAN

//REFERANCE
//Papers -- Citation Details:
// 1) X.-S. Yang, S. Deb, Cuckoo search via Levy flights,
// in: Proc. of World Congress on Nature & Biologically Inspired
// Computing (NaBIC 2009), December 2009, India,
// IEEE Publications, USA, pp. 210-214 (2009).
```

```
//http://arxiv.org/PS_cache/arxiv/pdf/1003/1003.1594v1.pdf
// 2) X.-S. Yang, S. Deb, Engineering optimization by cuckoo search,
//Int. J. Mathematical Modelling and Numerical Optimisation,
//Vol. 1, No. 4, 330-343 (2010).
//http://arxiv.org/PS_cache/arxiv/pdf/1005/1005.2908v2.pdf
"""

from abc import ABC, abstractmethod
from math import *
import random
from f_xj import *


class cuckoo_search1i(ABC):
    """
    int n;
    int nd;
    double pa;
    double tol;
    double nest[][];
    double xmin[];
    double xmax[];
    """

    def __init__(self,iff,in1,ipa,itol,ixmin,ixmax):
        self.n=in1
        self.nd=len(ixmin)
        self.pa=ipa
        self.tol=itol
        self.nest=[[0 for j in range(self.nd)] for i in range(self.n)]
        self.xmin=[0 for j in range(self.nd)]
        self.xmax=[0 for j in range(self.nd)]
        self.ff=iff
        for i in range(self.nd):
            self.xmin[i]=ixmin[i]
            self.xmax[i]=ixmax[i]


    def ones(self):
        v=[0 for j in range(self.nd)]
        for i in range(self.nd):
            v[i]=1.0
        return v

    def randn(self):
        a=[0 for j in range(self.nd)]
        for i in range(self.nd):
            a[i]=random.random()
        return a
    def initialize(self):
        for i in range(self.n):
            for j in range(self.nd):
                self.nest[i][j]=self.xmin[j]+(self.xmax[j]-self.xmin[j])*random.random()
        fitness=[0 for j in range(self.n)]
        for i in range(self.n):
            fitness[i]=1.0e7
        dp=[[[0 for j in range(self.nd)] for i in range(self.n)] for k in range(2)]
        for i in range(self.n):
            for j in range(self.nd):
                dp[0][i][j]=self.nest[i][j]
        for i in range(self.n):
            for j in range(self.nd):
                dp[1][i][j]=fitness[i]
        return dp

    def getminval_index(self,a):
        m=0.0
        m1=len(a)
        minval=a[0]
        for i in range(m1):
            if a[i]<minval:
                minval=a[i]
                m=i
        dep=[0 for j in range(2)]
        dep[0]=minval
        dep[1]=m
```

```
        return dep

    def get_best_nest(self,nest,newnest,fitness):
        fnew=0.0
        for j in range(self.n):
            fnew=self.ff.func(newnest[j])
            if fnew<fitness[j]:
                fitness[j]=fnew;self.nest[j]=newnest[j]
        dep=self.getminval_index(fitness)
        fmin=dep[0]
        index=int(dep[1])
        best=[0.0 for j in range(self.nd)]
        for i in range(self.nd):
            best[i]=self.nest[index][i]
        depp=[[[0.0 for j in range(self.nd)] for i in range(self.n)] for k in range(4)]
        depp[0][0][0]=fmin
        for i in range(self.n):
            depp[1][i][0]=fitness[i]
        for i in range(self.nd):
            depp[2][0][i]=best[i]
        for i in range(self.n):
            for j in range(self.nd):
                depp[3][i][j]=nest[i][j]
        return depp

                # Random permutation of matrix
    def randperm(self,a):
        n1=len(a)
        m1=len(a[0])
        a=[[0.0 for j in range(m1)] for i in range(n1)]
        y=[[0.0 for j in range(m1)] for i in range(n1)]
        sw=[0.0 for j in range(m1)]
        for i in range(n1):
            y[i]=a[i]
        mp=0
        for i in range(n1-1):
            rp=int((n1-i)*random.random()+i)
            sw=y[i]
            y[i]=y[rp]
            y[rp]=sw
        return y

    def get_cuckoos(self,nest,best):
        s=[0.0 for j in range(self.nd)]
        u=[0.0 for j in range(self.nd)]
        v=[0.0 for j in range(self.nd)]
        stepsize=[0 for j in range(self.nd)]
        beta=1.5
        sigma=pow((gamma(1+beta)*sin(pi*beta/2.0)/(gamma((1.0+beta)/2.0)*beta*pow(2.0,(beta-1.0)/2.0))),(1.0/beta))
        sk=[0.0 for j in range(self.nd)]
        for j in range(self.n):
            s=nest[j]
            for i in range(self.nd):
                u[i]=random.gauss(0.0,1.0)*sigma
                v[i]=random.gauss(0.0,1.0)
                step[i]=u[i]/pow((abs(v[i])),(1/beta) )
                stepsize[i]=0.01*step[i]*(s[i]-best[i])
            for k in range(self.nd):
                stepsize[k]*rndm.nextGaussian()
                s[k]=sk[k]+s[k];
                nest[j]=simplebounds(s)
        return nest

    def simplebounds(self,s):
        for i in range(self.nd):
            if s[i]<self.xmin[i]:
                s[i]=self.xmin[i]
            if s[i]>self.xmax[i]:
                s[i]=self.xmax[i]
        return s

    def emptynests(self,nest):
        K=[[0.0 for j in range(self.nd)] for i in range(self.n)]
        randmat=[[0.0 for j in range(self.nd)] for i in range(self.n)]
        perm1=[0.0 for i in range(self.n)]
        perm2=[0.0 for i in range(self.n)]
```

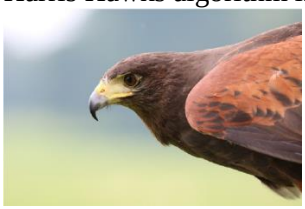
```
        nest1=[[0.0 for j in range(self.nd)] for i in range(self.n)]
        nest2=[[0.0 for j in range(self.nd)] for i in range(self.n)]
        nest3=[[0.0 for j in range(self.nd)] for i in range(self.n)]
        nest4=[[0.0 for j in range(self.nd)] for i in range(self.n)]
        stepsize=[[0.0 for j in range(self.nd)] for i in range(self.n)]
        randmat2=[[0.0 for j in range(self.nd)] for i in range(self.n)]
        newnest=[[0.0 for j in range(self.nd)] for i in range(self.n)]
        for i in range(self.n):
            for j in range(self.nd):
                randmat[i][j]=random.random()
                if randmat[i][j]<self.pa:
                    K[i][j]=0.0
                if randmat[i][j]>self.pa:
                    K[i][j]=1.0
        for i in range(self.n):
            for j in range(self.nd):
                nest1[i][j]=nest[i][j];nest2[i][j]=nest[i][j]
        a1=self.randperm(nest1)
        a2=self.randperm(nest2)
        for i in range(self.n):
            for j in range(self.nd):
                nest3[i][j]=a1[i][j]-a2[i][j]
        for i in range(self.n):
            for j in range(self.nd):
                stepsize[i][j]=nest3[i][j]*random.random()
        for i in range(self.n):
            for j in range(self.nd):
                nest4[i][j]=stepsize[i][j]*K[i][j]
        for i in range(self.n):
            for j in range(self.nd):
                newnest[i][j]=self.nest[i][j]+nest4[i][j]
        return newnest
    def solution(self):
        nestt=[[0.0 for j in range(self.nd)] for i in range(self.n)]
        fitness=[0.0 for i in range(self.n)]
        bestnest=[0.0 for i in range(self.nd)]
        bestnest=[0.0 for i in range(self.nd)]
        newnest=[0.0 for i in range(self.nd)]
        d1=self.initialize()
        for i in range(self.n):
            for j in range(self.nd):
                nestt[i][j]=d1[0][i][j]
        for i in range(self.n):
            fitness[i]=d1[1][i][0]
        d2=self.get_best_nest(nestt,nestt,fitness)
        fmin=d2[0][0][0]
        for i in range(self.n):
            fitness[i]=d2[1][i][0]
        for i in range(self.nd):
            bestnest[i]=d2[2][0][i]
        for i in range(self.n):
            for j in range(self.nd):
                nestt[i][j]=d2[3][i][j]
        iter=0
        fnew=0.0
        nestt=self.emptynests(nestt)
        while fmin>self.tol:
            newnest=get_cuckoos(nest,bestnest)
            d3=self.get_best_nest(nestt,newnest,fitness)
            fnew=d3[0][0][0]
            for i in range(self.n):
                fitness[i]=d3[1][i][0]
            for i in range(self.nd):
                best[i]=d3[2][0][i]
            for i in range(self.n):
                for j in range(self.nd):
                    nestt[i][j]=d3[3][i][j]
            iter=iter+n
            newnest=self.emptynests(nestt)
            d4=self.get_best_nest(nestt,newnest,fitness)
            fnew=d4[0][0][0]
            for i in range(self.n):
                fitness[i]=d4[1][i][0]
            for i in range(self.nd):
                best[i]=d4[2][0][i]
            for i in range(self.n):
```

```
            for j in range(self.nd):
                nestt[i][j]=d4[3][i][j]
        iter=iter+n
        if fnew<fmin:
            fmin=fnew;bestnest=best
    print("Optimum value of a function = ",fmin)
    m1=len(bestnest)
    for  i in range(m1):
        print("x[",i,"] = ",bestnest[i])
```

```
Reloaded modules: f_xj, cuckoo_search1i, plot3D1
Optimum value of a function =  -1.0979875810075364
x[ 0 ] =  1.0911747443128963
x[ 1 ] =  1.437869069358219
```

5.21 STOCHASTIC METHODS: HARRIS' HAWKS ALGORITHM

Harris Hawks algorithm is a stachastic method based on hunting behavior of harris hawks[93].



Harris Hawk

Harris hawk, unlike other hawk species develop a group cooperative hunting behavior. Harris hawks shows evolved innovative team chasing capabilities in tracing, encircling, flushing out, and eventually attacking and catching their pray.

Their group behavior has Exploration and exploitation phases. In the exploration phase Haris' hawks track and detect their prey. But prey can not be seen directly all the time, in this case they fly randomly and perch to some position to detect the prays. Perching point will be selected on two different strategy, based on the position of the other members of the team and base on the possible position of the prey. As computer simulation of this can be given with the following formulations: Assume a random number q, and if q≥0.5 perch in a random position, if q<0.5 select a position based on position of the other members and possible prey location.

$$X(t+1) = \begin{cases} X_{rand}(t) - r_1[X_{rand}(t) - 2r_2X(t)] & if \quad \mathrm{q} \geq 0.5 \\ \left(X_{prey}(t) - X_m(t)\right) - r_1[LB + r_4(UB - LB)] & if \quad \mathrm{q} < 0.5 \end{cases}$$

Where $X(t+1)$ is the position vector of the hawks in the next iteration. $X_{prey}(t)$ is the position of the prey, $X(t)$ is the current position of hawks, r_1, r_2, r_3, r_4 and q are random numbers inside (0,1) range, which are updated in each iteration. LB and UB are search limits (upper and lower boundaries of variables). $X_{rand}(t)$ is a randomly selected hawk from the hawk group. $X_m(t)$ is the average position of the hawks. The average position of the hawks group is calculated by using mathematical averahe formula $X_m(t) = \frac{1}{N}\sum_{i=1}^{N} X_i(t)$ where $X_i(t)$ is the location of the each hawk fort he iteration t, N is the total number of hawks.

In order to translate from Exploration to exploitation phase prey energy is observed. Energy of the prey is modelled as:

$E = 2E_0\left(1 - \frac{t}{T}\right)$ where E indicates escaping energy of the prey, T is yhe maximum number of iterations, and E_0 is the initial state of its energy. When E is positive, prey is stenthening, when E is negative the prey is weakening. Exploration happens when $|E| \geq 1$ and explotation happens when $|E| < 1$. In exploitation phase, attacks the intented prey. Howewer prey often attemps to escape from dangerous situation. Hence, different chasing styles occur in real situation. Suppose that r is the chance of a prey successfully escaping(r<0.5) or not successfully escaping(r≥0.5) before the surprise attack. The hawk behave two different mechanism to respond these, a hard or soft besiege.

When r≥ 0 and $|E| \geq 0.5$ soft besiege happens. In this case the prey has enough energy , and try to escape by some random movements. The Harris' hawks encircle it softly to make rabbit more exhausted the performs the surprise pounce. It is modelled as the following equations:

$$X(t+1) = \Delta X(t) - E\left[JX_{prey}(t) - X(t)\right]$$
$$\Delta X(t) = JX_{prey}(t) - X(t)$$
$$J = 2(1 - r_5)$$

Where $\Delta X(t)$ is the difference between the position vector of the rabbit and the current location in the iteration t, and r_5 is a random number. The hawks behave according to following pattern:

$$Y = X_{prey}(t) - E\left[JX_{prey}(t) - X(t)\right]$$
$$Z = Y + S * LF(D)$$

Where D is dimension of the problem aans S is a random vector of size 1xD and LF is the levy flight function.

$$LF(x) = 0.01\frac{u\sigma}{|v|^{\frac{1}{\beta}}}$$

$$\sigma = \left(\frac{\Gamma(1+\beta)sin\left(\frac{\beta\pi}{2}\right)}{\Gamma(\frac{1+\beta}{2})\beta\, 2^{\left(\frac{\beta-1}{2}\right)}}\right)^{\frac{1}{\beta}}$$

Where u,v are random numbers in (0,1) range, $\Gamma(z)$ is gamma function which is defined as:

$$\Gamma(z) = \int_{z=0}^{\infty} t^{z-1}e^{-t}dt = \frac{e^{-\gamma z}}{z}\prod_{n=1}^{\infty}\left(1+\frac{z}{n}\right)^{-1} e^{t/n}$$

where $\gamma = 0.57721566490153286\ldots$ is the Euler-Mascheroni constant

Hence the final strategy becomes:

$$X(t+1) = \begin{Bmatrix} Y & if\ F(Y) < F(X(t)) \\ Z & if\ F(Z) < F(X(t)) \end{Bmatrix}$$

When r< 0.5 and $|E| < 0.5$ the prey does not have enough energy so hard besiege happens. Hawks directly attempts the surprise pounce to kill the prey.

$$Y = X_{prey}(t) - E\left[JX_{prey}(t) - X_m(t)\right]$$
$$Z = Y + S * LF(D)$$
$$X(t+1) = \begin{Bmatrix} Y & if\ F(Y) < F(X(t)) \\ Z & if\ F(Z) < F(X(t)) \end{Bmatrix}$$

8.21 NON LINEAR OPTIMIZATION WITH BOUNDARY RESTRICTIONS

A non-linear optimization problem with boundary restrictions can be defined as
Minimum f(x)
Boundary restrictions : p(x)=0
$q(x) \geq 0$
In general these type of boundary restrictions are added up into the general equation as
For the first equation p(x)=0 define

$$P(x) = P(x_0, x_1, \ldots) = \sum_{k=0}^{r} p_k^2(x)$$

For the second equation $q(x) \geq 0$

$$Q(x) = Q(x_0, x_1, \ldots) = \sum_{k=0}^{r} minimum^2\{0, q(x)\}$$

Function min selects the minimum of 0 and $q_k(x)$.
The general function then becomes

$$F(x) = f(x) + \mu[P(x) + Q(x)]$$

μ in the equation is called punishment coefficient. For the second equation for $q(x) \geq 0$ equation Q(x) can be also written as

$$Q(x) = Q(x_0, x_1, \dots) = \sum_{k=0}^{r} minimum^2\{0, [q(x) - \varepsilon]\}$$

In this equation $\varepsilon > 0$ is a small number. With this small change refusal of the function when the value is exactly equal to the restriction limit can be evoided. In the sample problem Nelder-Mead Optimization method is used, but any unrestricted optimization methods can be used for this purpose.

In the first trial unrestricted case will be run ($\mu = 0$)

```
from f_xj import *
from math import *

class f2(f_xj):
    def __init__(self,mu,epsilon):
        self.mu=mu
        self.eps=epsilon
    def func(self,x):
      y=(x[0]-1.0)*(x[0]-1.0)+(x[1]-2.0)*(x[1]-2.0)+(x[2]-3.0)*(x[2]-3.0)
      return y+self.mu*(self.P(x)+self.Q(x));

    def P(self,x):
        # p(x)=0 boundary restriction
        n=len(x)
        y=(x[0]+x[1]-2.0)*(x[0]+x[1]-2.0)
        return y

    def Q(self,x):
        # q(x)>=0 boundary restriction
        n=len(x)
        ax=x[1]-x[2]-3.0-self.eps;
        bx=0;
        if ax<0:
            bx=ax*ax
        return bx

def  nelder(fnelder,a,da,maxiteration,tolerance,printlist):
    #print("a",a)
    #print("da",da)
    n=len(a)
    m=n+1
    x=[[0.0 for j in range(n)] for i in range(n+1)]
    p=[[0.0 for j in range(m)] for i in range(m)]
    s=""
    NDIMS = len(x)-1
    NPTS = len(x)
    FUNC = NDIMS
    ncalls = 0;
    for i in range(m):
        for j in range(n):
            if i==j:
                x[i][j]=a[j]+da[j]
                p[i][j]=x[i][j]
            else:
                x[i][j]=a[j]
                p[i][j]=x[i][j]
        p[i][FUNC] = fnelder.func(p[i])
        #print("x[",i,"]=",x[i])
        #print("p[",i,"]=",p[i])
    #print("p",p)
    #Inlet variable definitions
    #fnelder : abstract multivariable function f(x)
    #x : independent variable set of n+1 simplex elements
    #maxiteration : maximum iteration number
    #tolerance :

    #///// construct the starting simplex ////////////////////
    z=[0.0 for i in range(NDIMS)]
    best = 1.0E99;
    #/////////////// calculate the first function values for the simplex ////////////////
    iter=0;
    for iter in range(1,maxiteration):
```

```
#////////  define lo, nhi, hi (low high next_to_high //////////////
ilo=0
ihi=0
inhi = -1
# -1 means missing
flo = p[0][FUNC];
fhi = flo
#print("flo=",flo,"fhi=",fhi)
#double pavg,sterr;
for i in range(1,NPTS):
    if p[i][FUNC] < flo:
        flo=p[i][FUNC]
        ilo=i
    if  p[i][FUNC] > fhi:
        fhi=p[i][FUNC]
        ihi=i
    #print("i=",i,"flo=",flo,"fhi=",fhi,"ilo=",ilo,"ihi=",ihi,"pifunc=",p[i][FUNC])
fnhi = flo
inhi = ilo
#print("fnhi=",fnhi,"inhi=",inhi)
for i in range(NPTS):
    if (i != ihi) and (p[i][FUNC] > fnhi):
        fnhi=p[i][FUNC]
        inhi=i
#///////// exit criteria //////////////
if (iter % 4*NDIMS) == 0:
    #calculate the avarage (including maximum value)
    pavg=0
    for i in range(NPTS):
         pavg=pavg+p[i][FUNC]
    pavg=pavg/NPTS
    tot=0.0
    if printlist!=0:
        print(iter)
        for j in range(NDIMS):
            print(p[ilo][j]," ")
    for i in range(NPTS):
        tot=(p[i][FUNC]-pavg)*(p[i][FUNC]-pavg)
    sterr=sqrt(tot/NPTS)
    for j in range(NDIMS):
        z[j]=p[ilo][j]
    best = p[ilo][FUNC];
# end of if iter%4*NDMS)==0
#//// calculate avarage without maximum value //////
ave=[0.0 for i in range(NDIMS)]
for j in range(NDIMS):
    ave[j] = 0
for i in range(NPTS):
    if i != ihi:
        for j in range(NDIMS):
            ave[j] = ave[j]+p[i][j];
for j in range(NDIMS):
    ave[j]=ave[j]/(NPTS-1)
#print("ave[",j,"]=",ave[j])
#//////// reflect ////////////////
#print("reflect")
r=[0.0 for i in range(NDIMS)]
for j in range(NDIMS):
    r[j] = 2.0*ave[j] - p[ihi][j]
fr = fnelder.func(r)
if (flo <= fr) and (fr < fnhi): #in zone: accept
    for j in range(NDIMS):
        p[ihi][j] = r[j]
    p[ihi][FUNC] = fr
    continue
#///(//////////// expand
if fr < flo:
    #print("expand")
    e=[0.0 for i in range(NDIMS)]
    for j in range(NDIMS):
        e[j] = 3.0*ave[j] - 2.0*p[ihi][j];
    fe = fnelder.func(e);
    if fe < fr:
        for j in range(NDIMS):
            p[ihi][j] = e[j]
        p[ihi][FUNC] = fe
```

```
            continue
        else:
            for j in range(NDIMS):
                p[ihi][j] = r[j]
            p[ihi][FUNC] = fr
            continue
    #//////////// shrink:
    if fr < fhi:
        #print("srink")
        c=[0.0 for i in range(NDIMS)]
        for j in range(NDIMS):
            c[j] = 1.5*ave[j] - 0.5*p[ihi][j]
        fc = fnelder.func(c);
        if fc <= fr:
            for j in range(NDIMS):
                p[ihi][j] = c[j];
            p[ihi][FUNC] = fc
            continue
        else:   #////// shrink
            for i in range(NPTS):
                if i != ilo:
                    for j in range(NDIMS):
                        p[i][j] = 0.5*p[ilo][j] + 0.5*p[i][j]
                    p[i][FUNC] = fnelder.func(p[i])
            continue
    #/////////////// contract
    if fr >= fhi:
        #print("contract")
        cc=[0.0 for i in range(NDIMS)]
        for j in range(NDIMS):
            cc[j] = 0.5*ave[j] + 0.5*p[ihi][j];
        fcc = fnelder.func(cc)
        if fcc < fhi:
            for j in range(NDIMS):
                p[ihi][j] = cc[j];
            p[ihi][FUNC] = fcc
            continue
        else:
            for i in range(NPTS):
                if i != ilo:
                    for j in range(NDIMS):
                        p[i][j] = 0.5*p[ilo][j] + 0.5*p[i][j]
                    p[i][FUNC] = fnelder.func(p[i]);
    return z

a=[1,1,1]
da=[0.5122,0.643,0.234]
mu=0
eps=1.0e-6
f=f2(mu,eps)
r=nelder(f,a,da,100,1.0e-15,0)
print("r = ",r)
```

```
runfile('E:/okul/SCO1/restricted_nelder.py', wdir='E:/okul/SCO1')
Reloaded modules: f_xj
r =  [0.9999997496793286, 1.999999612703119, 2.999999903270221]
```

Now two boundary restriction will apply to the same function

$p(x) = x_0 + x_1 - 2.0 = 0$

$q(x) = x_1 - x_2 - 3.0 - \varepsilon \geq 0$

($\mu = 0.01$) relatively small effect of restrictions

```
a=[1,1,1]
da=[0.5122,0.643,0.234]
mu=0.01
eps=1.0e-6
f=f2(mu,eps)
r=nelder(f,a,da,100,1.0e-15,0)
print("r = ",r)
```

```
runfile('E:/okul/SCO1/restricted_nelder.py', wdir='E:/okul/SCO1')
Reloaded modules: f_xj
r =  [0.9898109093334182, 2.0291264348654634, 2.9606842523705]
```

$(\mu = 1)$

```
a=[1,1,1]
da=[0.5122,0.643,0.234]
mu=1
eps=1.0e-6
f=f2(mu,eps)
r=nelder(f,a,da,100,1.0e-15,0)
print("r = ",r)
```

```
runfile('E:/okul/SCO1/restricted_nelder.py', wdir='E:/okul/SCO1')
Reloaded modules: f_xj
r =  [0.124999962481402, 2.75000014097942, 1.3749995169144549]
```

$(\mu = 10)$

```
a=[1,1,1]
da=[0.5122,0.643,0.234]
mu=10
eps=1.0e-6
f=f2(mu,eps)
r=nelder(f,a,da,100,1.0e-15,0)
print("r = ",r)
```

```
runfile('E:/okul/SCO1/restricted_nelder.py', wdir='E:/okul/SCO1')
Reloaded modules: f_xj
r =  [-0.7888566745099646, 2.967742329493556, 0.24340119141917466]
```

$(\mu = 100)$

```
a=[1,1,1]
da=[0.5122,0.643,0.234]
mu=100
eps=1.0e-6
f=f2(mu,eps)
r=nelder(f,a,da,100,1.0e-15,0)
print("r = ",r)
```

```
runfile('E:/okul/SCO1/restricted_nelder.py', wdir='E:/okul/SCO1')
Reloaded modules: f_xj
r =  [-0.9768896182605153, 2.9966568356905317, 0.026395850601849763]
```

8.11 PROBLEMS

PROBLEM 1

In a distillation tank operates with saturated steam benzen and toluen is seperated. In order to obtain purest toluen in liquid phase (maximize the toluen in the mixture) what would be the ideal temperature?

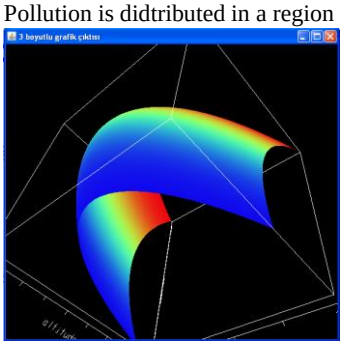
$x_{tolüen} P_{doyma_tolüen} + x_{benzen} P_{doyma_benzen} = P = 760$ mmHg

$\log_{10}(P_{doyma_benzen}) = 6.905 - 1211/(T+221)$
$\log_{10}(P_{doyma_tolüen}) = 6.953 - 1344/(T+219)$

temperature degree C, pressure mmHg

PROBLEM 2

Two dimensional distribution of pollution in a channel can be given by the equation
$C(x,y)=7.9+0.13x+0.21y-0.05x^2-0.016y^2-0.007xy$
Pollution is didtributed in a region -10 <= x <= 10 and 0 <= y <= 20 , find the place where the pollution is maximum



PROBLEM 3

A petrochemical company developing a new fuel additive for commercial airplanes. Additive is consist of three components X,Y and Z. Fort he best performance total additive in the fuel should be at least 6 mL/L. Due o security reasons the total of most flammable X and Y components should be less than 3 mL/L. Componnt X should always be equal or more than the component Y, and componet Z should be more than half of the component Y. The cost of X,Y and Z components are 0.15,0.025 ve 0.05 TL respectively. Find the mixing component tha will give the minimum cost.

PROBLEM 4

A food company is trying to design a new container box for preserved food that will minimise the cost of material. Total 320 cm^3 volume is requested. Estetic concers limits diameter of the box in between 3 to 10 cm. and the height in between 2 to 10 cm. Box will be made of steel sheet of 0.4 mm thich with density of 7800 kg/m^3. The cost of box material is 37 TL. Assume that box including top and bottom covers are made of the same material, find the box size that will be given the minimum cost. And the cost per box.

PROBLEM 5

Air enters at 1 bar 300K to a 3 stage compressor system. After each stage air is cooled down again to 300 K before entering to the next stage. Calculate internal pressure for each stage to minimize work input to the compressors.

The work requirement for eah stage of the compressor:

$W=C_p(T_i - T_e)=C_pT_i(1-T_e/T_i)=kRT_i/(k-1)*[1-(P_e/P_i)^{(k-1)/k}]$

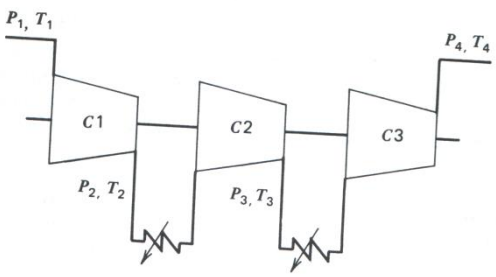
Total work fort he thre stages.

$W=kRT_i/(k-1)*[1-(P_2/P_1)^{(k-1)/k}]+ kRT_i/(k-1)*[1-(P_3/P_2)^{(k-1)/k}]+ kRT_i/(k-1)*[1-(P_4/P_3)^{(k-1)/k}]$

$W= kRT_i/(k-1)*[3-(P_2/P_1)^{(k-1)/k}-(P_3/P_2)^{(k-1)/k}-(P_4/P_3)^{(k-1)/k}]$

$(P_2/P_1) *(P_3/P_2)*(P_4/P_3)= (P_4/P_1)=27$

$k=1.4$ $C_p=1.005$ KJ/kg K



Work written as java function :

```
class f1 extends f_xj
{
public double func(double x[])
{
 double ff=1.4*8.3145*300/(1.4-1)*(1-x[0]*(1.4-1)/1.4)+ 1.4*8.3145*300/(1.4-1)*(1-(x[1]/x[0])*(1.4-1)/1.4)
 +1.4*8.3145*300/(1.4-1)*(1-(27/x[1])*(1.4-1)/1.4);
return ff; //minimum değil maxsimum istendiğinden - ile çarptik
}}
```

answer : 3 bar and 9 bar

PROBLEM 6

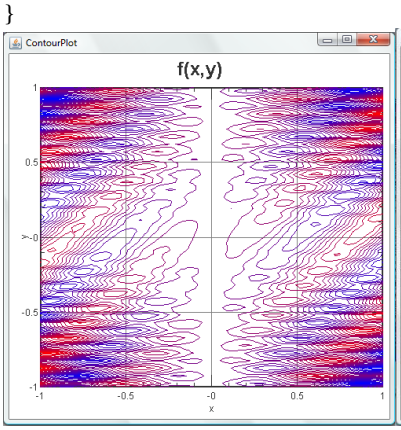
By using the data below find the aximum point. You can use an interpolation polynomial then mximise it.

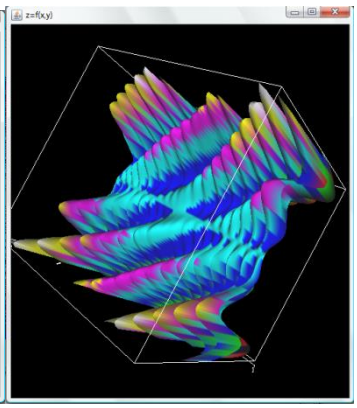
X	Y
0	2
1	6
2	8
3	12
4	14
5	13
6	11
7	9
8	13
9	14.5
10	7

PROBLEM 7

Function below is given, find the maximum. -1<x [0]<1 -1<x [1]<1

```
double func(double x[])
 { double ff= 34+x[0]*(Math.sin(3.0*Math.PI*(x[0]-x[1]))+x[1]*Math.sin(20.0*Math.PI*x[1]));
   return ff;
}
```





PROBLEM 8

An antibiotic producing microorganism has a specific growth rate(g) is function of food concentration (c)

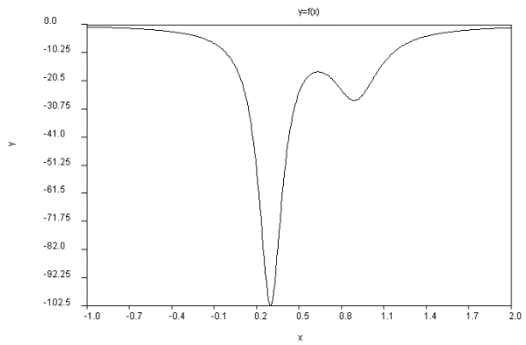
If $g = 2*c/(4+0.8c+c^2+0.2c^3)$

Find the c that will give te maximum growth rate. In low food concentrations growth rate drops to zero, similarly in very high food concentrations growth rate will drop to zero due to food poisining effect. Because of this reason, values bigger than c = 10 mg/L should not be investigated.

PROBLEM 9

Find minimum of function

f(x)=-(1.0/((x-0.3)*(x-0.3)+0.01)+1.0/((x-0.9)*(x-0.9)+0.04))



PROBLEM 10

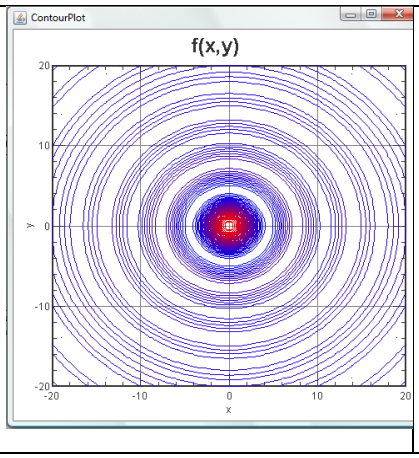
Function

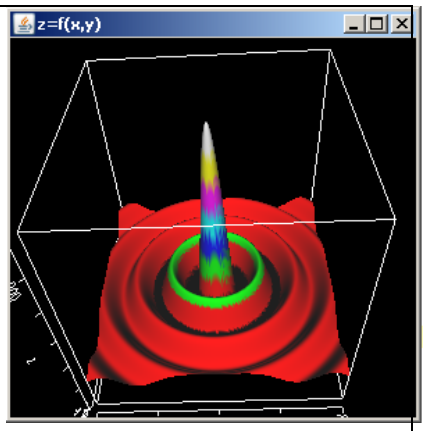
$$z(x,y) = \frac{\sin(\sqrt{x^2+y^2+\varepsilon})}{\sqrt{x^2+y^2+\varepsilon}}$$

$-20 \le x \le 20$ ve

$-20 \le y \le 20$

is given. Find the maximum of the function for ε=1e-2 starting from P(18,18)





PROBLEM 11

Function

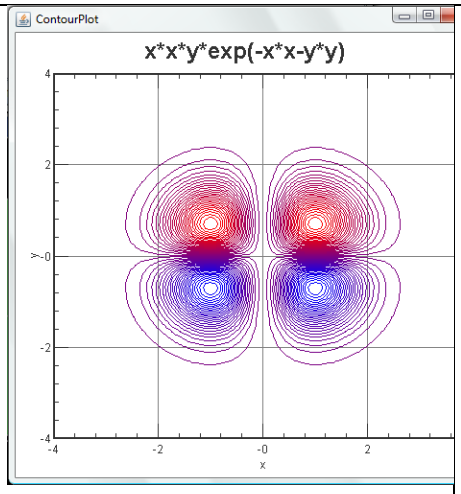
$$z(x,y) = x^2 y\, e^{(-x^2-y^2)}$$

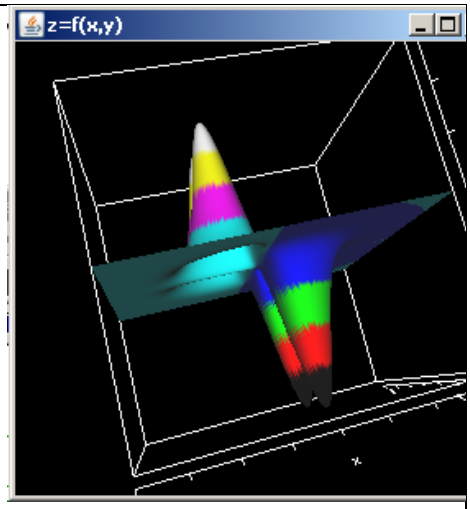
With boundaries

$-4 \le x \le 4$ ve

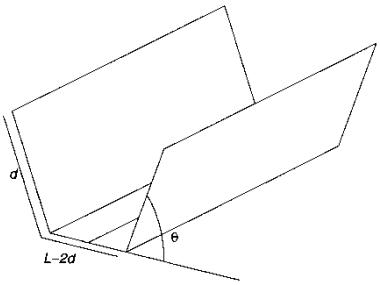
$-4 \le y \le 4$

is given. Find the minimum of the function starting from P(3,3)





PROBLEM 12



In order to construct water channel in a house , a flat sheet will be banded as shown in the figure. Water carrying capacity of the channel can be written as

$V(d,\theta) = Z(d^2\sin\theta\cos\theta + (L-2d)d\sin\theta)$

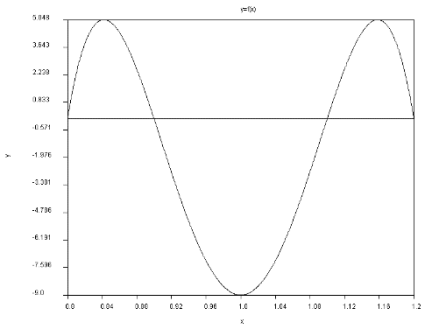
In this equation z is the length of the sheet (sheet area is zL). Find the d and □□to maximize water holding capacity of the channel.

PROBLEM 13

Function below is given

$f(x) = 2500x^6-15000x^5+14875x^4+40500x^3-97124x^2+73248x-19008$

find the minimum between x_0=0.8, x_1=1.2 values.



PROBLEM 14

Cities, A,B,C, and D located at corrdinates(0,0),(1,5),(5,2) and (7,3) respectively. The roads from all cities are merged at point P. Find point P that makes the total distance of the roads minimum.

PROBLEM 15

Function below is given

$f(x_0, x_1, x_2, x_3) = (x_0+10\,x_1)^2 + 5(x_2-10\,x_3)^2+(x_1-2\,x_2)^4+10(x_1-x_4)^4$

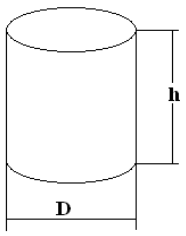
find the minimum starting from P(1,-1,0,1)

PROBLEM 16
Function below is given
$f(x_0, x_1) = 3(x_0 - 1)^2 + 2(x_1-1)(x_0-1)(x_1-2)+(x_1-2)^4$
Minimum of the function is given at point (1,2). Located the minimum at 3D plot, then by using any 3D nonlinear optimization method find the minimum points starting at P (0,0)

PROBLEM 17
Find theminimum of function

$$f(x) = \frac{15x}{(4x^2 - 3x + 4)}$$ fin the range of 0 to10.

PROBLEM 18



One of the very basic optimization problem is the minimum cost of container problem. The cost of a box usually is a function of the surface area. Therefore we should minimize the area for a given volume For example if the volume of the container V=0.5 liter=$0.5x10^{-3}$ m^3 :

Volume $V = \frac{\pi D^2}{4} h$ or from this equation h, height is obtained as $h = \frac{4V}{\pi D^2}$.

Surface area of the cylinder : $A = 2\frac{\pi D^2}{4} + \pi D h = \frac{\pi D^2}{2} + \frac{4V}{D}$.

Analytical solution of the minimization problem

$$\frac{dA}{dD} = \pi D - \frac{4V}{D^2} = 0$$

$D = \sqrt[3]{\frac{4V}{\pi}}$. From here solution is $D = \sqrt[3]{\frac{4*0.5x10^{-3}}{\pi}}$ =0.086025401 m and .

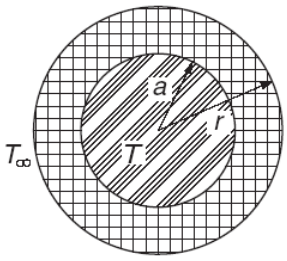
$h = \frac{4*0.5x10^{-3}}{\pi D^2}$ = 0.086025401 m.

Now obtain this results by using nuerical optimization methods.
For the range of 0.01<=D<=0.2

a) Graphic method
b) Golden ratio (Fibonnachi)
c) Newton-Raphson root finding
d) Bisection method
e) Newton-Raphson method
f) Secant method
g) Brent method
h) Second degree polynomials
i) Third degree polynomials

Note: Some of these methods are defined only as root finding method, is not defined as optimization methods.

PROBLEM 19

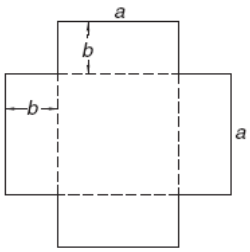


An isolation covered a wire carrying electrical current. The outlet diameter of electrical insulation is r. The electrical resistance on the wire caused heat. Thi heat crried through electrical insulation to the air around the wire with a temperature of $T_\propto$. The temperature of the wire can be calculated as

$$T = \frac{q}{2\pi}\left(\frac{\ln(r/a)}{k} + \frac{1}{hr}\right) + T_\propto$$ In this equation

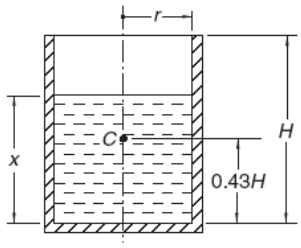
q = heat generation in the wire = 50 W/m
a = diameter of the wire = 5 mm = $5x10^{-3}$ m
k = coeeficient of thermal conductivity of insulation = 0.2 W/(mK)
h=thermal convetivity coefficient for air = 30 W/(m^2K)
$T_\propto$ = environmental temperature = 300 K.
Find radius r that wil make temperature T minimum.

PROBLEM 20



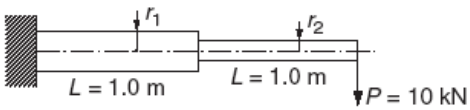
In order to make a cartoon box, the shape above should be used.After bending from the dotted linet he volume of the box will be 1 m^3. In order to spent the minimum amount of the cartoon, what a and b dimensions should be?

PROBLEM 21



A cylindrical container has a mass of M and has an empty center of gravity of C=0.43H. If the height of water in the cup is x, In order to have the common center of gravity of water and the cup as low as possible, what sholud be the value of x ?

PROBLEM 22



Two cylindrical beams with two different diameters is placed as in the figure. It is desired beam to have the minimum cross sectional area. Operational condions are:

$\sigma_1 \leq$ 180 MPa, $\sigma_2 \leq$ 180 MPa, and bending $\delta \leq$ 25 mm dir

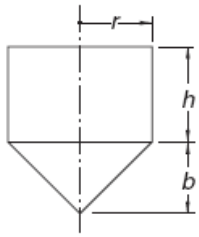
$\sigma_1 = \dfrac{8PL}{\pi\ r_1^3}$ =maximum stress in the left beam

$\sigma_2 = \dfrac{8PL}{\pi\ r_2^3}$ = maximum stress in the right beam

$\delta = \dfrac{4PL^3}{3\pi\ E}\left(\dfrac{7}{r_1^4} + \dfrac{1}{r_2^4}\right)$ =bending at the end of the beam

ve E =Young modüulus=200 GPa. Calculate r_1 and r_2.

PROBLEM 23



It is desired to have volume of the cone base shape shown in the figure as 1 m^3. Calculate r,h and b dimensions to minimise the surface area.

$$V = \pi r^2\left(\frac{b}{3} + h\right)$$

$$S = \pi r\left(2h + \sqrt{b^2 + r^2}\right)$$

PROBLEM 24

Lennard-Jones potential in between two molecules can be expressed as

$V = 4\varepsilon\left[\left(\dfrac{\sigma}{r}\right)^{12} - \left(\dfrac{\sigma}{r}\right)^{6}\right]$ In this equation □ and □ values are constants. Find $\left(\dfrac{\sigma}{r}\right)$ values that makes. V minimum

PROBLEM 25

Wave function for hydrogen atoms

$\psi = C(27 - 18\sigma + 2\sigma^2)e^{-\sigma/3}$ In this equation

$$\sigma = zr / a_0$$

$$C = \frac{1}{81\sqrt{3\pi}}\left(\frac{z}{a_0}\right)^{2/3}$$

z = nucleus electrical load

a_0 = Bohr diameter

r =radial distance.

Find σ value that make ψ wave function minimum.

PROBLEM 26

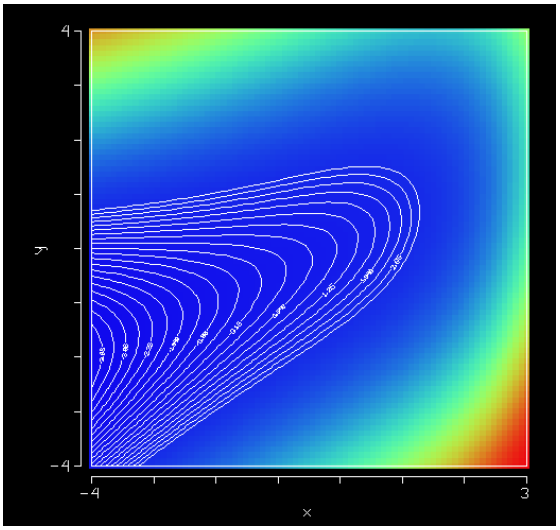
Find optimum of function

$$f = e^{x_1} - x_1 x_2 + x_2^2$$

$$g = 2x_1 + x_2 - 2 \le 0$$

$$h = x_1^2 + x_2^2 - 4 = 0$$

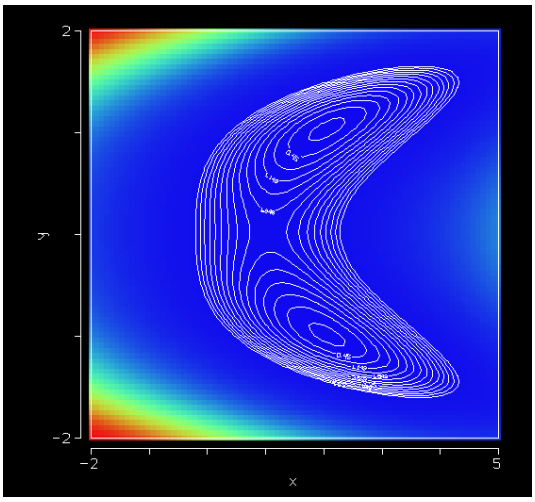
With -4<=x<=3 and -4<=y<=4 boundaries.



PROBLEM 27

Find the minimum of

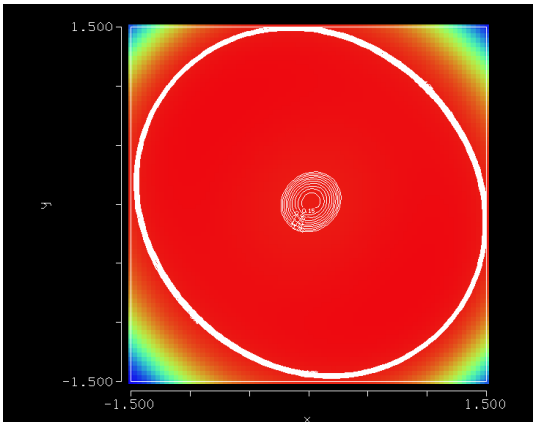
$$f = (x-2)^2 + (x-2y^2)^2 \quad .$$



PROBLEM 28

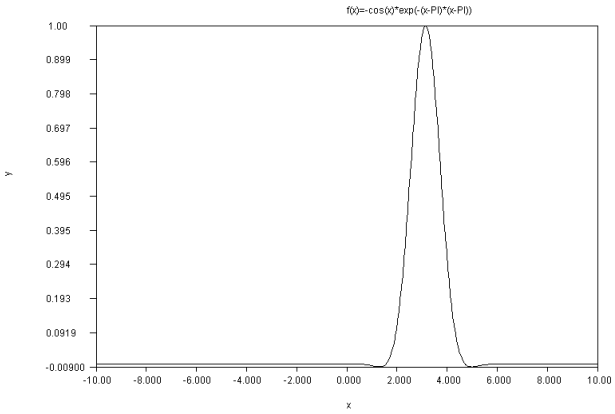
Find the minimum of

$$f = 25x^2 - 12x^4 + 6xy + 25y^2 - 24x^2y^2 - 12y^4 \quad .$$



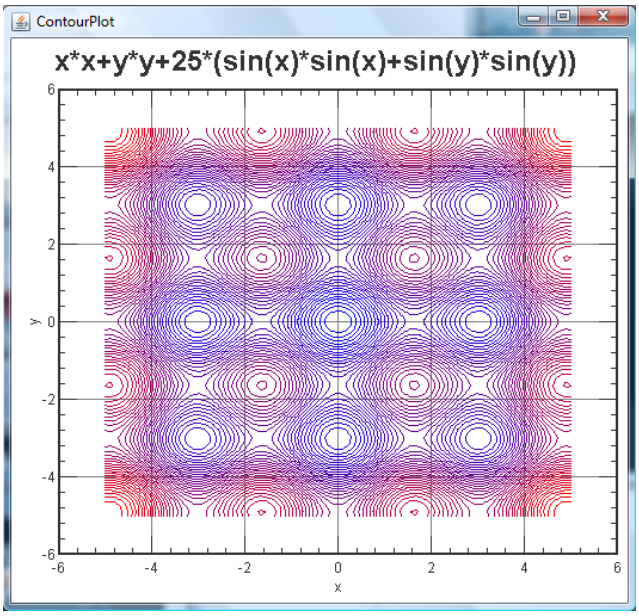
PROBLEM 29 Find maximum of Easoms's function

$f = -\cos(x)e^{-(x-\pi)^2}$ between limits [-10,10] by using Genetic algorithms



PROBLEM 30 Egg rate function

$f = x^2 + y^2 + 25[\sin^2(x) + \sin^2(y)]$ has the global minimum f=0 at (0,0) in the domain [-5,5]

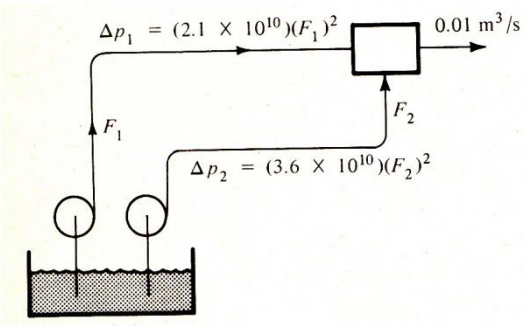


PROBLEM 31 : Two parallel pump-pipe assemblies, shown in the figure, deliver water from a common source to a common destination. The total flow rate required at the destination is $F=F_1+F_2=0.01\ m^3/s$. The drops in pressure in two lines are functions of the square of flow rates.

$\Delta p_1 = 2.1x10^{10} F_1^2$ Pa and $\Delta p_2 = 3.6x10^{10} F_2^2$ Pa where F_1 and F_2 are respective flow rates in Cubic meters per second. pumping power for a pump is given by the equation

$Power = \dfrac{F\Delta p}{\rho\eta}$ where F is flow rate(m^3/s), Δp is pressure drop (Pa), ρ=1000 kg/m^3 is density and is pump efficiency.

$\eta_1=0.81$ and $\eta_2=0.86$

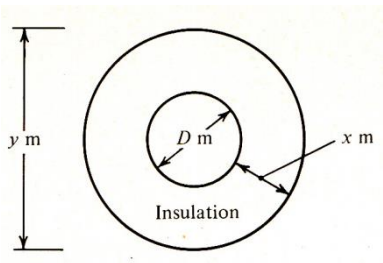


So pumping power is equal to: $Power = \dfrac{F_1\Delta p_1}{\rho\eta_1} + \dfrac{F_2\Delta p_2}{\rho\eta_2}$

$$Power = \frac{F_1\Delta p_1}{\rho\eta_1} + \frac{F_2\Delta p_2}{\rho\eta_2} = \frac{C_1F_1^3}{\rho\eta_1} + \frac{C_2F_2^3}{\rho\eta_2} = \frac{C_1F_1^3}{\rho\eta_1} + \frac{C_2}{\rho\eta_2}(F-F_1)^3$$

Solve the values of F_1 and F_2 for minimum power requirements.

PROBLEM 32: A pipe carrying high temperature water is to be insulated and then mounted in a restricted space, as shown in the figure. The choice of the pipe diameter D m and the insulation thickness x m are to be such that the OD of the insulation



Y is a minimum, but the total annual operation cost of the insulation is limited to 40000 TL. This annual operation cost has two components, the water pumping cost and the cost of the heat loss:

Pumping cost = $\dfrac{8}{D^5}$ and Heat cost= $\dfrac{1500}{x}$

If an objective function is created for this problem:

$y = D + 2x$

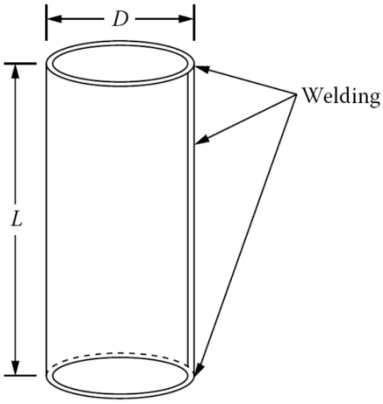
Subject to:

$\dfrac{8}{D^5} + \dfrac{1500}{x} = 40000$ Equation can be arranged as a single equation as:

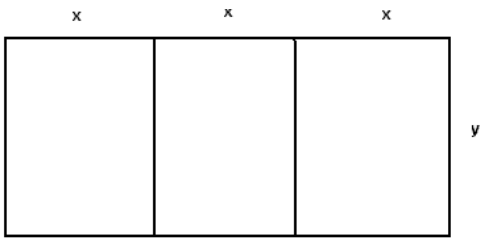
$y = D + 3000\left[\dfrac{1}{40000 - \dfrac{8}{D^5}}\right]$ Find the minimum y and corresponding x and D values.

PROBLEM 33: A manufacturer of steel cans wants to minimize costs. As a first approximation, the cost of making a can consists of the cost of the metalplus the cost of welding the longitudinal seam and the top and bottom. The can may have any diameter *D* and length *L*, for a given volume *V*. The wall thickness *d* is 1 mm. The cost of the material is \$0.50/kg and the cost of welding is \$0.1/m of the weld. The density of the material is

10^4 kg/m^3 Find the dimensions of the can that will minimize cost.



PROBLEM 33: A farmer needs 150 meters of fencing to fence three adjacent gardens. What would be maximum area of each garden, and what would be values of x and y



$$4y + 6x = 150$$
$$y = 37.5 - 1.5x$$
$$A(x) = 3(37.5 - 1.5x)x = 112.5x - 4.5x^2$$
$$\frac{dA(x)}{dx} = 112.5 - 9x = 0$$
$x = 12.5$ y=18.75

$A = 703.125$ total of 3 garden

Calculate by using a numerical method

9.0 NON-LINEAR SYSTEM OF EQUATIONS

9.1 Newton-Raphson Method

One dimensional Newton-Raphson Formula was:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \qquad (4.1.1)$$

If the equation format changed a little, it can be written in the form of

$$\Delta x_n = x_{n+1} - x_n = -\frac{f(x_n)}{f'(x_n)} \qquad (4.1.2)$$

$$f'(x_n)\Delta x_n = -f(x_n) \qquad (4.1.3)$$

Now the same equation can be considered for a system of non-linear equation

$$[\nabla f] = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix} \quad (4.1.4) \qquad \{\Delta x\} = \begin{Bmatrix} \Delta x_1 \\ \Delta x_2 \\ \cdots \\ \Delta x_n \end{Bmatrix} \quad \{f\} = \begin{Bmatrix} f_1 \\ f_2 \\ \cdots \\ f_n \end{Bmatrix} \quad (4.1.5)$$

So multidimensional Newton-Raphson equation becomes

$$[\nabla f]\{\Delta x\} = -\{f\} \qquad (4.1.6)$$

The equation can be solved by using tecqniques such as gauss elimination. An initial estimate for all the x values are required to start iterative solution

First example function

$f_0(x_0,x_1)=x_0^2+ x_0\, x_1-10$

$f_1(x_0,x_1)=x_1+3\, x_0\, x_1^2-57$

$$[\nabla f] = \begin{bmatrix} 2x_0 + x_1 & x_0 \\ 3x_1^2 & 1 + 6x_0x_1 \end{bmatrix}$$

newton_NLE.py

```
 #newton_NLE
from math import *
from fi_xi import *;
from gauss import *;

def newton(f,x):
  # Newton-Raphson root finding method
  k=len(x);
  nmax=100;
  tolerance=1.0e-10
  for i in range(nmax):
    fx= f.func(x);
    dfx=f.dfunc(x,1);
    dx=gauss(dfx,fx);
    for j in range(k):
        x[j]-=dx[j];
    total=0;
    for j in range(k):
        total+=dx[j];
    if abs(total)<tolerance: return x
  return x;
```

```
#newton_NLEtest
from math import *
from newton_NLE import *
from fi_xi import *;

class f1(fi_xi):
  def func(self,x):
      ff=[0.0 for i in range(2)]
      ff[0]=x[0]*x[0]+x[0]*x[1]-10.0
      ff[1]=x[1]+3.0*x[0]*x[1]*x[1]-57.0
      return ff
```

```
f=f1();
x=[1.0,1.0]
a=newton(f,x)
print("a=\n",a)
```

```
runfile('E:/okul/SCO1/newton_NLEtest.py', wdir='E:/okul/SCO1')
Reloaded modules: fi_xi, gauss
a=
 [2.0, 3.0]
```

If the same problem is solved by hand starting from the initial guess x={1,2}

Newton-Raphson Non-linear equation solution

Initial guess

x0	1
x1	2

	4	1	dx0	7
	12	13	dx1	43

3	4	1	dx0	7
	0	10	dx1	22

dx0	1.2
dx1	2.2

First iteration result

x0	2.2
x1	4.2

	8.6	2.2	dx0	-4.08
	52.92	56.44	dx1	-63.624

6.153488	8.6	2.2	dx0	-4.08
	0	42.90233	dx1	-38.5178

dx0	-0.24475
dx1	-0.8978

Second iteration results

x0	1.955252
x1	3.302199

	7.212702	1.955252	dx0	-0.27964
	32.71355	39.73977	dx1	-10.2654

4.535547	7.212702	1.955252	dx0	-0.27964
	0	27.70816	dx1	-8.54467

dx0	0.044827
dx1	-0.30838

Third iteration results

x0	2.000079
x1	2.993818

	6.993975	2.000079	dx0	0.011814
	26.88883	36.92723	dx1	0.2264

3.844571	6.993975	2.000079	dx0	0.011814
	0	29.23778	dx1	0.180979

dx0	-8.1E-05
dx1	0.00619

x0	1.999998
x1	3.000008

	7.000003	1.999998	dx0	4.94E-07
	27.00014	37.00005	dx1	-0.00022

3.857161	7.000003	1.999998	dx0	4.94E-07
	0	29.28574	dx1	-0.00022

dx0	2.24E-06
dx1	-7.6E-06

Fourth iteration results

x0	2
x1	3

	7	2	dx0	1.2E-11
	27	37	dx1	-4E-11
3.857143	7	2	dx0	1.2E-11
	0	29.28571	dx1	-8.6E-11

dx0	2.56E-12
dx1	-2.9E-12

Fifth iteration results

x0	2
x1	3

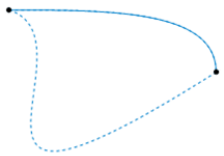
Another example:

```
#newton_NLEtest1
from math import *
from newton_NLE import *
from fi_xi import *;

class f1(fi_xi):
  def func(self,x):
      ff=[0.0 for i in range(3)]
      ff[0]=3.0*x[0]-cos(x[1]*x[2])-0.5;
      ff[1]=x[0]*x[0]-81.0*(x[1]+0.1)*(x[1]+0.1)+sin(x[2])+1.06;
      ff[2]=exp(-x[0]*x[1])+20.0*x[2]+(10.0*pi-3.0)/3.0;
      return ff

f=f1();
x=[1.0,1.0,1.0]
a=newton(f,x)
print("a=\n",a)
```

```
runfile('E:/okul/SCO1/newton_NLEtest1.py', wdir='E:/okul/SCO1')
Reloaded modules: fi_xi, gauss, newton_NLE
a=
 [0.49999999999999994, -1.3277223964327195e-17, -0.5235987755982988]
```

9.2 HOMOTOPY OR CONTINUATION METHOD

In topology, two continuous functions from one topological space to another are called **homotopic**(from Greek ὁμός homós "same, similar" and τόπος tópos "place") if one can be "continuously deformed" into the other, such a deformation being called a **homotopy** between the two functions.



When a problem of system of nonlinear equations of the form F(x)=0 desired to be solved, another interesting solution method is Homotopy or Continuation method. Assume that solution set to be found is x*. Consider a parametric function G(λ,x) in the form of

$G(\lambda,x) = \lambda F(x) + (1-\lambda)[\ F(x) - F(x(0))\ ]$ (9.2.1)

Where λ=0 coresponds to initial guess of the solution , x(0) ,and where λ=1 value correspods the actual solution set x(1)= x* (9.2.2)

It is desired to be found G(λ,x) = 0 therefore for λ=0 equation becomes

$G(\lambda,x) = G(0,x) = F(x) - F(x(0))$ and for λ=1 (9.2.3.4)

$0=G(1,x) = F(x)$ (9.2..4)

Therefore x(1)=x* solution set will be obtained. If a function G(λ,x) satisfies the above equation can be found, it will also give us the solution. Function G is called a homotopy between the function G(0,x) and G(1,x)=F(x) (9.2.5)

In order to find such a function, it is assumed to have a function

G(λ,x)=0 is existed and partial derivative of this function with respect to λ and x will also be zero

$\frac{\partial G(\lambda,x)}{\partial \lambda} + \frac{\partial G(\lambda,x)}{\partial x} x'(\lambda) = 0$ (9.2.6)

if x'(λ) is isolated form this equation, it becomes:

$x'(\lambda) = -\left[\frac{\partial G(\lambda,x)}{\partial x}\right]^{-1} \left[\frac{\partial G(\lambda,x)}{\partial \lambda}\right]$ (9.2.7)

If $G(\lambda,x)=\lambda F(x)+(1-\lambda)[F(x)-F(x(0))]$ the equation is substituted into the differential equation

$$\left[\frac{\partial G(\lambda,x)}{\partial \lambda}\right] = \begin{bmatrix} \frac{\partial f_1(x(\lambda))}{\partial x_1} & \frac{\partial f_1(x(\lambda))}{\partial x_2} & \frac{\partial f_1(x(\lambda))}{\partial x_{33}} \\ \frac{\partial f_2(x(\lambda))}{\partial x_1} & \frac{\partial f_2(x(\lambda))}{\partial x_2} & \frac{\partial f_2(x(\lambda))}{\partial x_3} \\ \frac{\partial f_3(x(\lambda))}{\partial x_1} & \frac{\partial f_3(x(\lambda))}{\partial x_2} & \frac{\partial f_3(x(\lambda))}{\partial x_3} \end{bmatrix} = J(x(\lambda)) \quad (9.2.8)$$

Forms a Jacobian matrix. and

$\left[\frac{\partial G(\lambda,x)}{\partial x}\right] = F(x(0))$ (9.2.9)

Differential equation becomes

$x'(\lambda) = \frac{dx(\lambda)}{d\lambda} = -[J(x(\lambda)]^{-1} F\big(x(0)\big) \quad 0 \le \lambda \le 1$ (9.2.10)

It is possible to solve such a differential equation by using initial value problem approaches, solution at x(1) will be given us the roots of the system of equation. Solutions of initial value problems will be given latter chapters in details, but A sixth order Runge-Kutta solution will be defined here to solve our homotopy problem. If equation is in the form of $\frac{dx(\lambda)}{d\lambda} = f(\lambda,x(\lambda))$ (9.2.11) the 6th order Runge-Kutta method to numerically solve this differential equation. It is defined as:

Runge-Kutta with sixth degree polynomial solution RK6:

$y_{i+1} = y_i + (1/90)*(7k_1 + 32k_3 +12k_4+32k_5+7k_6)h$

$k_1=f(x_i,y_i)$

$k_2=f(x_i+0.25h , y_i+0.25k_1h)$

$k_3=f(x_i+0.25h , y_i+0.125k_1h+0.125k_2h)$

$k_4=f(x_i+0.5h , y_i – 0.5k_2h+k_3h)$

$k_5=f(x_i+0.75h , y_i + (3/16)k_1h+(9/16)k_4h)$

$k_6=f(x_i+h , y_i - (3/7)k_1h+(2/7)k_2h+(12/7)k_3h - (12/7)k_4h+(8/7)k_5h)$

This equation can be given as Buthcher tableu as:

$$\begin{array}{c|ccccc} 0 & 0 & 0 & 0 & 0 & 0 \\ 1/4 & 1/4 & 0 & 0 & 0 & 0 \\ 1/4 & 1/8 & 1/8 & 0 & 0 & 0 \\ 2/4 & 1/4 & -1/4 & 1 & 0 & 0 \\ 3/4 & 3/16 & 0 & 0 & 9/16 & 0 \\ 1 & -3/7 & 2/7 & 12/7 & -12/7 & 8/7 \\ \hline & 1/90 & 7/90 & 32/90 & 12/90 & 7/90 \end{array}$$

In these equations h is finite difference step size. Solution starts by using the initial value λ=0 , x0(0) and adds h into λ in each iteration step. The example program uses 6th degree Runge-Kutta method to solve homotopy(Continuation problem). It should be note that Homotophy method is less dependent to initial value compare to methods such as Newton-Raphson therefore one possibility is to approach solution with a relatively rough estimate with homotophy following with a Newton-Raphson type of method. In the following code methods of Homotophy with 6th order Runge-Kutta methods is given.

```
#continuation_NLE
from math import *
from fi_xi import *
from gauss import *
```

```
def newton(f,x):
  # Newton-Raphson root finding method
  k=len(x);
  nmax=100;
  tolerance=1.0e-10
  for i in range(nmax):
    fx= f.func(x);
    dfx=f.dfunc(x,1);
    dx=gauss(dfx,fx);
    for j in range(k):
        x[j]-=dx[j];
    total=0;
    for j in range(k):
        total+=dx[j];
    if abs(total)<tolerance: return x
  return x

def mult_c_v(left,right):
   #multiplying a vector with a constant
   i=0
   n=len(right)
   b=[0 for x in range(n)]
   for i in range(n):
     b[i]=left*right[i]
   return b;

def mult_m_m(left,right):
   #multiplying two matrisis
   n1=len(left);
   m1=len(left[0]);
   n2=len(right);
   m2=len(right[0]);
   b = [[0 for x in range(n1)] for y in range(m2)];
   if m1 != n2:
     print("warning : matrix sizes should be same");
     return b;
   for i in range(0,n1):
     for j in range(0,m2):
       for k in range(0,m1):
          b[i][j]+=left[i][k]*right[k][j];
   return b
def mult_m_v(left,right):
   #multiplication of one matrix with one vector
   #int ii,jj,i,j,k;
   m1=len(left[0])
   n1=len(left)
   m2=len(right)
   b = [0 for x in range(m2)]
   if n1 != m2:
      print("inner matrix dimensions must agree")
      for ii in range(n1):
         b[ii]=0
      return b;
   for i in range(m1):
      b[i]=0
      for k in range(n1):
         b[i]=b[i]+left[i][k]*right[k]
   return b;
#end of multiply of a matrix and a vector

def mult_v_m(left,right):
   #multiplication of one vector with one matrix
   #ii,jj,i,j,k;
   m2=len(right[0])
   n2=len(right)
   m1=len(left)
   b=[0.0 for i in range(m1)]
   if n2 != m1:
      print("inner matrix dimensions must agree")
      for ii in range(n2):
         b[ii]=0
         return b
      for i in range(m2):
         b[i]=0;
         for k in rage(m1):
```

```
            b[i]+=right[i][k]*left[k]
    return b

def mult_c_m(left,right):
    #multiplying a matrix with a constant
    n=len(right);
    m=len(right[0])
    b = [[0 for x in range(m)] for y in range(n)];
    for i in range(0,n):
      for j in range(0,m):
        b[i][j]=right[i][j]*left;
    return b
    #end of multiplying a matrix with a constant double

def add_m_m(left,right):
    #adding two matrisis
    n1=len(left);
    m1=len(left[0]);
    n2=len(right);
    m2=len(right[0]);
    b = [[0 for x in range(n1)] for y in range(m2)];
    if m1 != n2:
      print("warning : matrix sizes should be same");
      return b;
    for i in range(0,n1):
      for j in range(0,m2):
          b[i][j]=left[i][j]+right[i][j];
    return b

def add_v_v(left,right):
    #addition of two vectors
    n1=len(left)
    n2=len(right)
    nMax=0
    i=0
    if n1>=n2:
       nMax=n1
    else:
       nMax=n2
    b=[0 for x in range(nMax)]
    for i in range(n1):
       b[i]=b[i]+left[i]
    for i in range(n2):
       b[i]=b[i]+right[i]
    return b
#end of vector addition method

def continuationRK6(f,x,N):
    #=====================================================
    # Roots of nonlinear system of equations Homotophy RK6
    # yi+1 = yi + (1/90)*( 7k1 + 32k3 +12k4+32k5+7k6)h
    # k1=f(xi,yi)
    # k2=f(xi+0.25h , yi+0.25k1h)
    # k3=f(xi+0.25h , yi+0.125k1h+0.125k2h)
    # k4=f(xi+0.5h , yi - 0.5k2h+k3h)
    # k5=f(xi+0.75h , yi + (3/16)k1h+(9/16)k4h)
    # k6=f(xi+h , yi - (3/7)k1h+(2/7)k2h+(12/7)k3h - (12/7)k4h+(8/7)k5h)
    #=====================================================
    #x vector of independent variables
    #y vector of dependent variables
    #dy derivative vector of dependent variables
    i=0
    nmax=400
    tolerance=1.0e-20
    n=len(x)
    h=1.0/float(N)
    b=[0.0 for i in range(n)]
    x1=[0.0 for i in range(n)]
    x0=[0.0 for i in range(n)]
    b0=[0.0 for i in range(n)]
    for i in range(n):
       x0[i]=x[i]
    k = [[0 for x in range(n)] for y in range(6)]
    b=mult_c_v(-h,f.func(x))
    for i in range(N):
       A=f.dfunc(x,1)
```

```
        k[0]=gauss(A,b)
        b0=mult_c_v(-h,f.func(x))
        b=b0
        x1=add_v_v(x,mult_c_v(0.25,k[0]));
        A=f.dfunc(x1,1)
        #k2=f(xi+0.25h , yi+0.25k1h)
        k[1]=gauss(A,b)
        b=mult_c_v(-h,f.func(x0))
        x1=add_v_v(x,add_v_v(mult_c_v(0.125,k[0]),mult_c_v(0.125,k[1])))
        A=f.dfunc(x1,1)
        #k3=f(xi+0.25h , yi+0.125k1h+0.125k2h)
        k[2]=gauss(A,b)
        b=mult_c_v(-h,f.func(x0))
        x1=add_v_v(x,add_v_v(mult_c_v(-0.5,k[1]),k[2]));
        A=f.dfunc(x1,1)
        #k4=f(xi+0.5h , yi - 0.5k2h+k3h)
        k[3]=gauss(A,b)
        b=mult_c_v(-h,f.func(x0))
        x1=add_v_v(x,add_v_v(mult_c_v((3.0/16.0),k[0]),mult_c_v((9.0/16.0),k[3])));
        A=f.dfunc(x1,1)
        #k5=f(xi+0.75h , yi + (3/16)k1h+(9/16)k4h)
        k[4]=gauss(A,b)
        b=mult_c_v(-h,f.func(x0))
        x1=add_v_v(x,add_v_v(mult_c_v((-
3.0/7.0),k[0]),add_v_v(mult_c_v((2.0/7.0),k[1]),add_v_v(mult_c_v((12.0/7.0),k[2]),add_v_v(mult_c_v((-
12.0/7.0),k[3]),mult_c_v((8.0/7.0),k[4])))))));
        A=f.dfunc(x1,1)
        #k6=f(xi+h , yi - (3/7)k1h+(2/7)k2h+(12/7)k3h - (12/7)k4h+(8/7)k5h)
        k[5]=gauss(A,b)
        b=mult_c_v(-h,f.func(x0))
        #yi+1 = yi + (1/90)*( 7k1 + 32k3 +12k4+32k5+7k6)h
        for j in range(n):
            x[j]=x[j]+1.0/90.0*(7.0*k[0][j]+32.0*k[2][j]+12.0*k[3][j]+32.0*k[4][j]+7.0*k[5][j])
    return x
```

```
#continuation_NLEtest
from math import *
from continuation_NLE import *
from fi_xi import *;

class f1(fi_xi):
  def func(self,x):
      ff=[0.0 for i in range(2)]
      ff[0]=x[0]*x[0]+x[0]*x[1]-10.0
      ff[1]=x[1]+3.0*x[0]*x[1]*x[1]-57.0
      return ff

f=f1();
x0=[0.5,1.0]
a=continuationRK6(f,x0,10)
print("a=\n",a)
```

```
runfile('E:/okul/SCO1/continuation_NLEtest.py', wdir='E:/okul/SCO1')
Reloaded modules: fi_xi, gauss
a=
 [2.000049169278479, 3.0001230115694364]
```

```
#continuation_NLEtest
from math import *
from continuation_NLE import *
from fi_xi import *;

class f1(fi_xi):
  def func(self,x):
      ff=[0.0 for i in range(2)]
      ff[0]=x[0]*x[0]+x[0]*x[1]-10.0
      ff[1]=x[1]+3.0*x[0]*x[1]*x[1]-57.0
      return ff


f=f1();
x0=[0.5,1.0]
a=continuationRK6(f,x0,1000)
print("a=\n",a)
```

```
runfile('D:/okul/SCO1/continuation_NLEtest.py', wdir='D:/okul/SCO1')
Reloaded modules: continuation_NLE, fi_xi, gauss
```

```
a=
 [2.000000005727764, 3.0000000131203]
```

By combining different methods, result accuracy may be improved

```
#continuation_NLEtest
from math import *
import continuation_NLE as co
from fi_xi import *;

class f1(fi_xi):
  def func(self,x):
      ff=[0.0 for i in range(2)]
      ff[0]=x[0]*x[0]+x[0]*x[1]-10.0
      ff[1]=x[1]+3.0*x[0]*x[1]*x[1]-57.0
      return ff


f=f1();
x0=[0.5,1.0]
a=co.continuationRK6(f,x0,100)
b=co.newton(f,a)
print("b=\n",b)
```

```
runfile('D:/okul/SCO1/continuation_NLEtest.py', wdir='D:/okul/SCO1')
Reloaded modules: continuation_NLE, fi_xi, gauss, newton_NLE
b=
 [2.0, 3.0]
```

9.3 SOLUTION OF THE NONLINEAR SYSTEM OF EQUATIONS BY USING OPTIMIZATION ALGORITHMS

Non-Linear system of equations can also be solved by using optimization methods through an adaptation function.

To solve function $f_i(x_j)=0$

$$g(x_j) = \sum_{i=0}^{n_equation} f_i(x_j) * f_i(x_j)$$

The derivative of this equation is the root of non-linear system of equation again became $f_i(x_j)=0$, therefore optimum of $g(x_j)$ is the same problem as solution of $f_i(x_j)=0$.

As the first adaptation algorithm Nelder-Mead Method will be given.

$$f(0) = 3x_0 - \cos(x_1 x_2) - 0.5$$
$$f(1) = x_0^2 - 81(x_1 + 0.1)^2 + \sin(x_2) + 1.06$$
$$f(2) = \exp(-x_0 x_1) + 20x_2 + \frac{(10\pi - 3.0)}{3}$$

```
from gauss import *;
from math import *;
from f_xj import *;
from fi_xi import *;
import numpy as np;
import matplotlib.pyplot as plt;
from mpl_toolkits.mplot3d import Axes3D
from random import *;

class f1(f_xj):
   def __init__(self,f):
      self.ff1=f
   def func(self,x):
      ffx=0.0
      fa=self.ff1.func(x)
      n=len(fa)
      for i in range(n):
         ffx=ffx+fa[i]*fa[i]
      return ffx
```

```
class fa(fi_xi):
    def __init__(self):
        self.name = 'fa'
    def func(self,x):
        ff=[0.0 for j in range(3)]
        ff[0]=3.0*x[0]-cos(x[1]*x[2])-0.5
        ff[1]=x[0]*x[0]-81.0*(x[1]+0.1)*(x[1]+0.1)+sin(x[2])+1.06
        ff[2]=exp(-x[0]*x[1])+20.0*x[2]+(10.0*pi-3.0)/3.0
        return ff


def  nelder(fnelder,a,da,maxiteration,tolerance,printlist):
    #print("a",a)
    #print("da",da)
    n=len(a)
    m=n+1
    x=[[0.0 for j in range(n)] for i in range(n+1)]
    p=[[0.0 for j in range(m)] for i in range(m)]
    s=""
    NDIMS = len(x)-1
    NPTS = len(x)
    FUNC = NDIMS
    ncalls = 0;
    for i in range(m):
        for j in range(n):
            if i==j:
                x[i][j]=a[j]+da[j]
                p[i][j]=x[i][j]
            else:
                x[i][j]=a[j]
                p[i][j]=x[i][j]
        p[i][FUNC] = fnelder.func(p[i])
        #print("x[",i,"]=",x[i])
        #print("p[",i,"]=",p[i])
    #print("p",p)
    #Inlet variable definitions
    #fnelder : abstract multivariable function f(x)
    #x : independent variable set of n+1 simplex elements
    #maxiteration : maximum iteration number
    #tolerance :

    #///// construct the starting simplex ////////////////////
    z=[0.0 for i in range(NDIMS)]
    best = 1.0E99;
    #/////////////// calculate the first function values for the simplex ////////////////
    iter=0;
    for iter in range(1,maxiteration):
        #////////  define lo, nhi, hi (low high next_to_high //////////////
        ilo=0
        ihi=0
        inhi = -1
        # -1 means missing
        flo = p[0][FUNC];
        fhi = flo
        #print("flo=",flo,"fhi=",fhi)
        #double pavg,sterr;
        for i in range(1,NPTS):
            if p[i][FUNC] < flo:
                flo=p[i][FUNC]
                ilo=i
            if  p[i][FUNC] > fhi:
                fhi=p[i][FUNC]
                ihi=i
            #print("i=",i,"flo=",flo,"fhi=",fhi,"ilo=",ilo,"ihi=",ihi,"pifunc=",p[i][FUNC])
        fnhi = flo
        inhi = ilo
        #print("fnhi=",fnhi,"inhi=",inhi)
        for i in range(NPTS):
            if (i != ihi) and (p[i][FUNC] > fnhi):
                fnhi=p[i][FUNC]
                inhi=i
        #///////// exit criteria //////////////
        if (iter % 4*NDIMS) == 0:
            #calculate the avarage (including maximum value)
            pavg=0
```

```
        for i in range(NPTS):
            pavg=pavg+p[i][FUNC]
        pavg=pavg/NPTS
        tot=0.0
        if printlist!=0:
            print(iter)
            for j in range(NDIMS):
                print(p[ilo][j]," ")
        for i in range(NPTS):
            tot=(p[i][FUNC]-pavg)*(p[i][FUNC]-pavg)
        sterr=sqrt(tot/NPTS)
        for j in range(NDIMS):
            z[j]=p[ilo][j]
        best = p[ilo][FUNC];
    # end of if iter%4*NDMS)==0
    #//// calculate avarage without maximum value //////
    ave=[0.0 for i in range(NDIMS)]
    for j in range(NDIMS):
        ave[j] = 0
    for i in range(NPTS):
        if i != ihi:
            for j in range(NDIMS):
                ave[j] = ave[j]+p[i][j];
    for j in range(NDIMS):
        ave[j]=ave[j]/(NPTS-1)
    #print("ave[",j,"]=",ave[j])
    #//////// reflect ///////////////
    #print("reflect")
    r=[0.0 for i in range(NDIMS)]
    for j in range(NDIMS):
        r[j] = 2.0*ave[j] - p[ihi][j]
    fr = fnelder.func(r)
    if (flo <= fr) and (fr < fnhi): #in zone: accept
        for j in range(NDIMS):
            p[ihi][j] = r[j]
        p[ihi][FUNC] = fr
        continue
    #///(//////////// expand
    if fr < flo:
        #print("expand")
        e=[0.0 for i in range(NDIMS)]
        for j in range(NDIMS):
            e[j] = 3.0*ave[j] - 2.0*p[ihi][j];
        fe = fnelder.func(e);
        if fe < fr:
            for j in range(NDIMS):
                p[ihi][j] = e[j]
            p[ihi][FUNC] = fe
            continue
        else:
            for j in range(NDIMS):
                p[ihi][j] = r[j]
            p[ihi][FUNC] = fr
            continue
    #//////////// shrink:
    if fr < fhi:
        #print("srink")
        c=[0.0 for i in range(NDIMS)]
        for j in range(NDIMS):
            c[j] = 1.5*ave[j] - 0.5*p[ihi][j]
        fc = fnelder.func(c);
        if fc <= fr:
            for j in range(NDIMS):
                p[ihi][j] = c[j];
            p[ihi][FUNC] = fc
            continue
        else:   #////// shrink
            for i in range(NPTS):
                if i != ilo:
                    for j in range(NDIMS):
                        p[i][j] = 0.5*p[ilo][j] + 0.5*p[i][j]
                    p[i][FUNC] = fnelder.func(p[i])
            continue
    #/////////////// contract
    if fr >= fhi:
        #print("contract")
```

```
            cc=[0.0 for i in range(NDIMS)]
            for j in range(NDIMS):
                cc[j] = 0.5*ave[j] + 0.5*p[ihi][j];
            fcc = fnelder.func(cc)
            if fcc < fhi:
                for j in range(NDIMS):
                    p[ihi][j] = cc[j];
                p[ihi][FUNC] = fcc
                continue
            else:
                for i in range(NPTS):
                    if i != ilo:
                        for j in range(NDIMS):
                            p[i][j] = 0.5*p[ilo][j] + 0.5*p[i][j]
                        p[i][FUNC] = fnelder.func(p[i]);
    return z

a=[0.5,0.5,0.5]
da=[0.1122,0.243,0.25]
ff=fa()
f=f1(ff)
r=nelder(f,a,da,100,1.0e-15,0)
print("r = ",r)
```

```
runfile('E:/okul/SCO1/nelder_NLE.py', wdir='E:/okul/SCO1')
Reloaded modules: gauss, f_xj, fi_xi
r =  [0.5000562603286207, 1.0137773003008005e-05, -0.5235979821174814]
```

As a second example we will use steepest descent method

```
from math import *
from f_x import *
from matplotlib import pyplot as plt
import numpy as np
from f_xj import *;
from fi_xi import *;
from mpl_toolkits.mplot3d import Axes3D
from random import *;

class f1(f_xj):
    def __init__(self,f):
        self.ff1=f
    def func(self,x):
        ffx=0.0
        fa=self.ff1.func(x)
        n=len(fa)
        for i in range(n):
            ffx=ffx+fa[i]*fa[i]
        return ffx

class fa(fi_xi):
    def __init__(self):
        self.name = 'fa'
    def func(self,x):
        ff=[0.0 for j in range(3)]
        ff[0]=3.0*x[0]-cos(x[1]*x[2])-0.5
        ff[1]=x[0]*x[0]-81.0*(x[1]+0.1)*(x[1]+0.1)+sin(x[2])+1.06
        ff[2]=exp(-x[0]*x[1])+20.0*x[2]+(31.415926-3.0)/3.0
        return ff

class f1dim(f_x):
    def __init__(self,ff,pi,xi):
      self.name = 'f1dim';
      self.ff1=ff;
      self.pc=pi;
      self.xc=xi;
    def func(self,x):
        n=len(self.pc);
        xt = [0.0 for ii in range(n)];
        for j in range(n):
            xt[j]=self.pc[j]+x*self.xc[j];
        return self.ff1.func(xt);

def linmin(f,p,xi):
    f1=f1dim(f,p,xi);
```

```
    xi=1.123554162;
    xmin=newton(f1,xi);
    return xmin;

def multiply_c_v(left,right):
    #multiplying a vector with a constant
    n=len(right);
    b = [0.0 for i in range(n)];
    for i in range(n):
        b[i]=left*right[i];
    return b;

def add(left,right):
    #addition of two vectors
    n1=len(left);
    n2=len(right);
    if n1>=n2: nMax=n1;
    else     : nMax=n2;
    b = [0.0 for i in range(nMax)];
    for i in range(n1):
        b[i]=b[i]+left[i];
    for i in range(n2):
        b[i]=b[i]+right[i];
    return b;

def steepestdescent(f,x):
    # steepestdescent method
    ti=1.0;
    nmax=50;
    tolerance=1.0e-6;
    n=len(x);
    b = [0.0 for ii in range(n)];
    dy = [0.0 for ii in range(n)];
    i=0;
    total=0;
    for i in range(n):
        b[i]=1.0;
        total+=abs(b[i]);
    i=0;
    while i<nmax and total>tolerance:
        dy=f.dfunc(x);
        lm=linmin(f,x,dy);
        b= multiply_c_v(lm,dy);
        x=add(x,b);
        total=0;
        for j in range(n):
            total+=abs(b[j]);
        i=i+1;
    return x;

def newton(f,x):
   # Newton-Raphson optimisation method
   # single variable
  nmax=100
  tolerance=1.0e-6
  for i in range(0,nmax):
      fx= f.dfunc(x);
      dfx=f.d2func(x);
      x=x-fx/dfx
      if abs(fx)<tolerance: return x
  return x;


a=[0.5,0.0,-0.5]
ff=fa()
f=f1(ff)
r=steepestdescent(f,a)
print("r = ",r)
```

```
runfile('D:/okul/SCO1/steepest_decent_root.py', wdir='D:/okul/SCO1')
Reloaded modules: f_x, f_xj, fi_xi
r =  [0.5000105547521496, 6.473394258141787e-07, -0.523598580022288]
```

As a third method particle swarm algorithm

```
 from gauss import *;
from math import *;
from f_xj import *;
from fi_xi import *;
import numpy as np;
import matplotlib.pyplot as plt;
from mpl_toolkits.mplot3d import Axes3D
from random import *;

class f1(f_xj):
    def __init__(self,f):
        self.ff1=f
    def func(self,x):
        ffx=0.0
        fa=self.ff1.func(x)
        n=len(fa)
        for i in range(n):
            ffx=ffx+fa[i]*fa[i]
        return ffx

class fa(fi_xi):
    def __init__(self):
        self.name = 'fa'
    def func(self,x):
        ff=[0.0 for j in range(3)]
        ff[0]=3.0*x[0]-cos(x[1]*x[2])-0.5
        ff[1]=x[0]*x[0]-81.0*(x[1]+0.1)*(x[1]+0.1)+sin(x[2])+1.06
        ff[2]=exp(-x[0]*x[1])+20.0*x[2]+(10.0*pi-3.0)/3.0
        return ff

class particle_swarm1i():

#class variables
#int m,n
#double alpha,beta,beta0,teta,gamma
#m number of population
#n number of variables for each individual
#double x[][];  location for each population member
#double y[];    function value of each population member
#double yxmin;  current best minimum
#double ygmin;  global best minimum
#double xmin[]; current best x
#double gmin[]; global best x
#double v[][];  velocity vector
#int ngeneration;  number of generations in iteration


    def __init__(self,ff,mi, xxmin,xxmax,alphai,betai,ngenerationi):
        #initial population set
        self.teta=0.5
        self.gamma=0.5
        #initial population set
        self.n=len(xxmin);
        self.m=mi;
        self.alpha=alphai
        self.beta0=betai
        self.beta=self.beta0
        self.ngeneration=ngenerationi
        self.x= [[0 for j in range(self.n)] for i in range(self.m)]
        self.v= [[0 for j in range(self.n)] for i in range(self.m)]
        self.y=[0 for i in range(self.m)]
        self.xmin=[0 for i in range(self.n)]
        self.gmin=[0 for i in range(self.n)]
        self.ngeneration=ngenerationi
        yxmin=1.0e50
        for i in range(self.m):
            for j in range(self.n):
                self.x[i][j]=xxmin[j]+random()*(xxmax[j]-xxmin[j])
                self.v[i][j]=random()
            self.y[i]=ff.func(self.x[i]);
            if self.y[i]<yxmin:
                yxmin=self.y[i]
                ygmin=self.y[i]
                for k in range(self.n):
                    self.xmin[k]=self.x[i][k]
                    self.gmin[k]=self.x[i][k]
```

```
        t=0
        while t<self.ngeneration:
            yxmin=1.0e50
            for i in range(self.m):
                for j in range(self.n):
                    self.x[i][j]=self.x[i][j]+self.v[i][j]
                    if self.x[i][j]<xxmin[j]:
                        self.x[i][j]=xxmin[j]
                    if self.x[i][j]>xxmax[j]:
                        self.x[i][j]=xxmax[j]
                    self.y[i]=ff.func(self.x[i])
                    if self.y[i]<yxmin:
                        yxmin=self.y[i]
                        for k in range(self.n):
                            self.xmin[k]=self.x[i][k]
            for i in range(self.m):
                if self.y[i]<ygmin:
                    ygmin=self.y[i]
                    for k in range(self.n):
                        self.gmin[k]=self.x[i][k]
            for i in range(self.m):
                for j in range(self.n):
                    self.v[i][j]=self.teta*self.v[i][j]+self.alpha*random()*(self.gmin[j]-self.xmin[j])+self.beta*random()*(self.xmin[j]-
self.x[i][j])
            t=t+1
            self.beta=self.beta0*pow(self.gamma,0.01*t)

ff=fa()
f=f1(ff)
xmin=[-1.0,-1.0,-1.0];
xmax=[1.0,1.0,1.0];
m=500
alpha=2.1
beta=0.7123
ngeneration=500
r1= particle_swarm1i(f,m,xmin,xmax,alpha,beta,ngeneration)
z=r1.xmin;
print("xmin = ",z,"fmin=",f.func(z));
```

```
runfile('D:/okul/SCO1/particle_swarm_NLE.py', wdir='D:/okul/SCO1')
Reloaded modules: f_x, f_xj, fi_xi
xmin =  [0.5, 4.9334269649687666e-18, -0.5235987755982988] fmin= 3.1554436208840472e-30
```

Another non-linear equation

$$f(0) = x_0^2 + x_0 x_1 - 10$$

$$f(1) = x_1 + 3x_0 x_1^2 - 57$$

```
#nelder_NLEtest
from math import *
import continuation_NLE as co
import nelder_NLE as nel
from fi_xi import *;

class fa(fi_xi):
  def func(self,x):
      ff=[0.0 for i in range(2)]
      ff[0]=x[0]*x[0]+x[0]*x[1]-10.0
      ff[1]=x[1]+3.0*x[0]*x[1]*x[1]-57.0
      return ff

# nonlinear system of equation
ff=fa()
#conversion to optimisation function
f=nel.f1(ff)
x0=[0.5,1.0]
dx=[0.1122,0.143]
a=co.continuationRK6(ff,x0,100)
print("a=\n",a)
b=nel.nelder(f,x0,dx,100,1.0e-15,0)
print("b=\n",b)
```

```
runfile('D:/okul/SCO1/nelder_NLE_test.py', wdir='D:/okul/SCO1')
Reloaded modules: gauss, f_xj, fi_xi
```

```
a=
 [2.000000556357058, 3.0000013131609307]
b=
 [2.0000000000000098, 3.0000000000000044]
```

```
#steepest_descent_root test
from math import *
import continuation_NLE as co
import steepest_decent_root as stp
from fi_xi import *;

class fa(fi_xi):
  def func(self,x):
      ff=[0.0 for i in range(2)]
      ff[0]=x[0]*x[0]+x[0]*x[1]-10.0
      ff[1]=x[1]+3.0*x[0]*x[1]*x[1]-57.0
      return ff

# nonlinear system of equation
ff=fa()
#conversion to optimisation function
f=stp.f1(ff)
x0=[0.5,1.0]
a=co.continuationRK6(ff,x0,100)
print("a=\n",a)
b=stp.steepestdescent(f,a)
print("b=\n",b)
```

```
runfile('D:/okul/SCO1/steepest_descent_NLE_test.py', wdir='D:/okul/SCO1')
Reloaded modules: continuation_NLE, fi_xi, gauss, steepest_decent_root, f_x, f_xj
a=
 [2.000000556357058, 3.0000013131609307]
b=
 [1.9999998541451125, 3.0000001096152387]
```

```
#steepest_descent_root test
from math import *
import steepest_decent_root as stp
from fi_xi import *;

class fa(fi_xi):
  def func(self,x):
      ff=[0.0 for i in range(2)]
      ff[0]=x[0]*x[0]+x[0]*x[1]-10.0
      ff[1]=x[1]+3.0*x[0]*x[1]*x[1]-57.0
      return ff

# nonlinear system of equation
ff=fa()
#conversion to optimisation function
f=stp.f1(ff)
x0=[1.0,1.0]
b=stp.steepestdescent(f,x0)
print("b=\n",b)
```

```
runfile('D:/okul/SCO1/steepest_descent_NLE_test1.py', wdir='D:/okul/SCO1')
b=
 [1.9999996045274542, 3.000000296080793]
```

```
# test
from math import *
import particle_swarm_NLE as ps
from fi_xi import *;

class fa(fi_xi):
  def func(self,x):
      ff=[0.0 for i in range(2)]
      ff[0]=x[0]*x[0]+x[0]*x[1]-10.0
      ff[1]=x[1]+3.0*x[0]*x[1]*x[1]-57.0
      return ff

# nonlinear system of equation
ff=fa()
```

```
f=ps.f1(ff)
xmin=[-1.0,-1.0];
xmax=[4.0,4.0];
m=500
alpha=2.1
beta=0.7123
ngeneration=500
r1= ps.particle_swarm1i(f,m,xmin,xmax,alpha,beta,ngeneration)
z=r1.xmin;
print("xmin = ",z,"fmin=",f.func(z));
```

```
runfile('D:/okul/SCO1/particle_swarm_NLE_test.py', wdir='D:/okul/SCO1')
Reloaded modules: particle_swarm_NLE, gauss, f_xj, fi_xi
xmin =  [2.0, 3.0] fmin= 0.0
```

9.4 PROBLEMS

PROBLEM 1

Solve the following system of non-linear equations

$4x_1^2-20x_1+0.25x_2^2+8=0$

$0.5x_1x_2^2+2x_1+8=0$

By using

a) Newton raphson method

b) Continuity(Homotopy) method

PROBLEM 2

Solve the following system of non-linear equations

$3x_1^2-\cos(x_2x_2) - 0.5=0$

$4x_1^2-6.25x_2^2+2x_3-1=0$

$e^{x1\ x2} +20x_3+(10\pi-3)/3=0$

By using

a) Newton raphson method

b) Continuity(Homotopy) method

PROBLEM 3

Solve te following system of non-linear equations

$x_1(1- x_1) + 4x_3 – 12 =0$

$(x_1-2)^2+(2x_2-3)^2-25 =0$

By using

a) Newton raphson method

b) Continuity(Homotopy) method

PROBLEM 4

Solve the following system of non-linear equations

$\sin(4\pi x_1x_2)-2x_2-x_1=0$

$\left(\frac{4\pi-1}{4\pi}\right)(e^{2x_1}-e)+4ex_2^2-2ex_1=0$

By using

a) Newton raphson method

b) Continuity(Homotopy) method

PROBLEM 5

Solve the following system of non-linear equations

$10x_1-2x_2^2+x_2-2x_3-5=0$

$8x_2^2+4x_3^2-9=0$

$8x_2x_3+4=0$

By using

a) Newton raphson method

b) Continuity(Homotopy) method

10. CURVE FITTING

Data is obtained by measurements in the scientific world. Scientits are generalized the measurement results by using various formulas and theories. Creating functions by using experimental numerical data is called curve fitting. Formulas are generally formed from the theories of physical laws or freely selected by the researchers. In the fitted equation usually a good fit to the experimental data is a primary requirement. Fitting means assigning values to unknown coefficients in the equations. The coefficients can be linearly or non-linearly dependent the the variables of the function(s). Interpolation is also similar process, but in the interpolation a function is not always required. Interpolation could be just a repeatetive calculation process by following a certain rule. Different curve fitting methods will be investigated in this chapter. At the end of the chapter Approximation of the functions will also be covered.

10.1 LINEAR CURVE FITTING : LEAST SQUARE METHODS

Least square methods are one of the most used curve fitting methods. Polinomial equation version is widely used. Assuming having a linear function f(a_j,x) where a_j are m linear coefficient and x are independent variable and $\phi_j(x)$ are m sub-funtions linearly multiplied with the coefficients

$f(a_i, x) = \sum_{j=0}^{m} a_j^{(m)} \phi_j(x)$ (9.1-1)

It should be noted again that linearity is only for the coefficients, $\phi_j(x)$ functions does not have to be linear.

Sample functions: $f(a_i, x) = a_0 \sin(x) + a_1 \exp(x) + a_2 \ln(x) + a_3 x^2$

It is desired to fit data x_i, f_i, i=0..n into the equation so that the difference between data and fitted function dependent values for all points will be minimum. In order to establish this, the following function H will be minimized with respect to a_j

$$H(a_0^{(m)}, .., a_0^{(m)}) = \sum_{i=1}^{n} w(x_i) \left[f(x_i) - \sum_{j=0}^{m} a_j^{(m)} \phi_j(x) \right]^2 \qquad (10.1-1)$$

Where w(x_i) values in the equation are called weight function. In order to minimiz the function root of the derivative of the function can be calculated.

$$\frac{\partial H(a_0^{(m)}, .., a_0^{(m)})}{\partial a_k^{(m)}} = 2 \sum_{i=1}^{n} w(x_i) \left[f(x_i) - \sum_{j=0}^{m} a_j^{(m)} \phi_j(x) \right] \phi_k(x) = 0 \quad k = 0, \dots, m \qquad (10.1-2)$$

For weight function to be taken equal to unity , $w(x_i)$ =1, equation in the open form can be written in the following form.

$$\begin{bmatrix} \sum_{i=1}^{n} w(x_i)\phi_0(x_i)\phi_0(x_i) & \sum_{i=1}^{n} w(x_i)\phi_0(x_i)\phi_1(x_i) & \sum_{i=1}^{n} w(x_i)\phi_0(x_i)\phi_2(x_i) & \cdots & \sum_{i=1}^{n} w(x_i)\phi_0(x_i)\phi_n(x_i) \\ \sum_{i=1}^{n} w(x_i)\phi_1(x_i)\phi_0(x_i) & \sum_{i=1}^{n} w(x_i)\phi_1(x_i)\phi_1(x_i) & \sum_{i=1}^{n} w(x_i)\phi_1(x_i)\phi_2(x_i) & \cdots & \sum_{i=1}^{n} w(x_i)\phi_2(x_i)\phi_n(x_i) \\ \sum_{i=1}^{n} w(x_i)\phi_2(x_i)\phi_0(x_i) & \sum_{i=1}^{n} w(x_i)\phi_2(x_i)\phi_1(x_i) & \sum_{i=1}^{n} w(x_i)\phi_2(x_i)\phi_2(x_i) & \cdots & \sum_{i=1}^{n} w(x_i)\phi_2(x_i)\phi_n(x_i) \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \sum_{i=1}^{n} w(x_i)\phi_n(x_i)\phi_0(x_i) & \sum_{i=1}^{n} w(x_i)\phi_n(x_i)\phi_1(x_i) & \sum_{i=1}^{n} w(x_i)\phi_n(x_i)\phi_2(x_i) & \cdots & \sum_{i=1}^{n} w(x_i)\phi_n(x_i)\phi_n(x_i) \end{bmatrix} \begin{Bmatrix} a_0 \\ a_1 \\ a_2 \\ \cdots \\ a_n \end{Bmatrix} = \begin{Bmatrix} \sum_{i=1}^{n} w(x_i)\phi_0(x_i) f(x_i) \\ \sum_{i=1}^{n} w(x_i)\phi_1(x_i) f(x_i) \\ \sum_{i=1}^{n} w(x_i)\phi_2(x_i) f(x_i) \\ \cdots \\ \sum_{i=1}^{n} w(x_i)\phi_n(x_i) f(x_i) \end{Bmatrix} \quad (10.1\text{-}3)$$

If w(x_i)=1 matrix will be

$$\begin{bmatrix} \sum_{i=1}^{n} \phi_0(x_i)\phi_0(x_i) & \sum_{i=1}^{n} \phi_0(x_i)\phi_1(x_i) & \sum_{i=1}^{n} \phi_0(x_i)\phi_2(x_i) & \cdots & \sum_{i=1}^{n} \phi_0(x_i)\phi_n(x_i) \\ \sum_{i=1}^{n} \phi_1(x_i)\phi_0(x_i) & \sum_{i=1}^{n} \phi_1(x_i)\phi_1(x_i) & \sum_{i=1}^{n} \phi_1(x_i)\phi_2(x_i) & \cdots & \sum_{i=1}^{n} \phi_2(x_i)\phi_n(x_i) \\ \sum_{i=1}^{n} \phi_2(x_i)\phi_0(x_i) & \sum_{i=1}^{n} \phi_2(x_i)\phi_1(x_i) & \sum_{i=1}^{n} \phi_2(x_i)\phi_2(x_i) & \cdots & \sum_{i=1}^{n} \phi_2(x_i)\phi_n(x_i) \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \sum_{i=1}^{n} \phi_n(x_i)\phi_0(x_i) & \sum_{i=1}^{n} \phi_n(x_i)\phi_1(x_i) & \sum_{i=1}^{n} \phi_n(x_i)\phi_2(x_i) & \cdots & \sum_{i=1}^{n} \phi_n(x_i)\phi_n(x_i) \end{bmatrix} \begin{Bmatrix} a_0 \\ a_1 \\ a_2 \\ \cdots \\ a_n \end{Bmatrix} = \begin{Bmatrix} \sum_{i=1}^{n} \phi_0(x_i) f(x_i) \\ \sum_{i=1}^{n} \phi_1(x_i) f(x_i) \\ \sum_{i=1}^{n} \phi_2(x_i) f(x_i) \\ \cdots \\ \sum_{i=1}^{n} \phi_n(x_i) f(x_i) \end{Bmatrix} \quad (10.1\text{-}4)$$

This equation is an m+1 linear system of equation. It can easily be solved by using a system of equation solving method. A special form of the above equation a form of a polinomial can be assumed. If $\phi_j(x) = x^j$, equation becomes

$$f(a_i, x) = \sum_{j=0}^{m} a_j^{(m)} x^j \qquad (10.1-5)$$

In this case the minimisation equation becomes

$$\frac{\partial H\left(a_0^{(m)}, .., a_0^{(m)}\right)}{\partial a_k^{(m)}} = 2\sum_{i=1}^{n} w(x_i)\left[f(x_i) - \sum_{j=0}^{m} a_j^{(m)} x^j\right] x^k = 0 \quad k = 0, \dots, m \qquad (10.1-6)$$

$$\begin{bmatrix} \sum_{i=1}^{n} w(x_i) & \sum_{i=1}^{n} w(x_i)x_i & \sum_{i=1}^{n} w(x_i)x_i^2 & \cdots & \sum_{i=1}^{n} w(x_i)x_i^n \\ \sum_{i=1}^{n} w(x_i)x_i & \sum_{i=1}^{n} w(x_i)x_i^2 & \sum_{i=1}^{n} w(x_i)x_i^3 & \cdots & \sum_{i=1}^{n} w(x_i)x_i^{n+1} \\ \sum_{i=1}^{n} w(x_i)x_i^2 & \sum_{i=1}^{n} w(x_i)x_i^3 & \sum_{i=1}^{n} w(x_i)x_i^4 & \cdots & \sum_{i=1}^{n} w(x_i)x_i^{n+2} \\ \dots & \dots & \dots & \dots & \dots \\ \sum_{i=1}^{n} w(x_i)x_i^n & \sum_{i=1}^{n} w(x_i)x_i^{n+1} & \sum_{i=1}^{n} w(x_i)x_i^{n+2} & \cdots & \sum_{i=1}^{n} w(x_i)x_i^{2n} \end{bmatrix} \begin{Bmatrix} a_0 \\ a_1 \\ a_2 \\ \cdots \\ a_n \end{Bmatrix} = \begin{Bmatrix} \sum_{i=1}^{n} w(x_i)f(x_i) \\ \sum_{i=1}^{n} w(x_i)x_i f(x_i) \\ \sum_{i=1}^{n} w(x_i)x_i^2 f(x_i) \\ \dots \\ \sum_{i=1}^{n} w(x_i)x_i^n f(x_i) \end{Bmatrix} \quad (10.1-7)$$

Or if $w(x_i) = 1$

$$\begin{bmatrix} n & \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 & \cdots & \sum_{i=1}^{n} x_i^n \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i^3 & \cdots & \sum_{i=1}^{n} x_i^{n+1} \\ \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i^3 & \sum_{i=1}^{n} x_i^4 & \cdots & \sum_{i=1}^{n} x_i^{n+2} \\ \dots & \dots & \dots & \dots & \dots \\ \sum_{i=1}^{n} x_i^n & \sum_{i=1}^{n} x_i^{n+1} & \sum_{i=1}^{n} x_i^{n+2} & \cdots & \sum_{i=1}^{n} x_i^{2n} \end{bmatrix} \begin{Bmatrix} a_0 \\ a_1 \\ a_2 \\ \cdots \\ a_n \end{Bmatrix} = \begin{Bmatrix} \sum_{i=1}^{n} f(x_i) \\ \sum_{i=1}^{n} x_i f(x_i) \\ \sum_{i=1}^{n} x_i^2 f(x_i) \\ \dots \\ \sum_{i=1}^{n} x_i^n f(x_i) \end{Bmatrix} \quad (10.1-8)$$

Linear polynomial regression

Consider the general polynomial curve fitting equation and take only first two column and row, equation will be for the linear polynomial $f(x)=a_0+a_1x$

$$\begin{bmatrix} n & \sum_{i=1}^{n} x_i \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 \end{bmatrix} \begin{Bmatrix} a_0 \\ a_1 \end{Bmatrix} = \begin{Bmatrix} \sum_{i=1}^{n} f(x_i) \\ \sum_{i=1}^{n} x_i f(x_i) \end{Bmatrix} \quad (10.1-9)$$

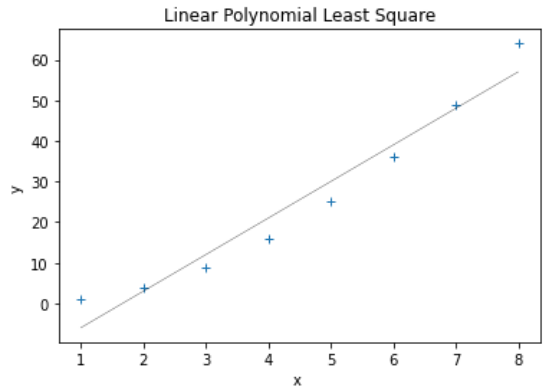
Sample data:
1 1
2 4
3 9
4 16
5 25
6 36
7 49
8 64

```
 # Linear least curve fitting (regression)  y=a0+a1x
from math import *
import numpy as np
import matplotlib.pyplot as plt
import gauss
def poly_1(x,y):
    n=len(x)
    x1=0
    x2=0
    y1=0
    y2=0
    for i in range(n):
        x1+=x[i]
```

```
        x2+=x[i]*x[i]
        y1+=y[i]
        y2+=x[i]*y[i]

    a= [[0 for i in range(2)] for j in range(2)];
    a[0][0]=n
    a[0][1]=x1
    a[1][0]=x1
    a[1][1]=x2;
    b=[0 for i in range(2)]
    b[0]=y1
    b[1]=y2
    x=gauss.gauss(a,b)
    return x

def func(e,x):
    # this function calculates the value of
    # least square curve fitting function
    ff=e[0]+e[1]*x
    return ff

def read_array2D(d):
    path = d
    file1 = open(path,'r')
    file1.seek(0,2)  #jumps to the end
    endLocation=file1.tell()
    file1.seek(0) #jumps to the start
    nn=0
    m=0
    while True:
        line=file1.readline()
        ee=line.split()
        nn+=1
        if  file1.tell()==endLocation:
            break
    m=len(ee)
    print("m=",m)
    a=[[0.0 for i in range(nn)] for j in range(m)]
    file1.seek(0) #jumps to the start
    for i in range(nn):
        line=file1.readline()
        ee=line.split()
        for j in range(m):
            a[j][i]=float(ee[j])
    file1.close()
    return a

na=input("file name : ")
a=read_array2D(na)
n=len(a[0])
x =a[0]
print("x=",x)
y =a[1]
print("y=",y)
e=poly_1(x,y)
n1=100
x1=[0 for x in range(n1)]
y1=[0 for x in range(n1)]
xmax=max(x)
xmin=min(x)
dx=(xmax-xmin)/(n1-1)
for i in range(n1):
    x1[i]=xmin+i*dx
    y1[i]=func(e,x1[i])
print("x1=",x1)
print("y1=",y1)
plt.title('Linear Polynomial Least Square')
plt.xlabel('x ')
plt.ylabel('y ')
plt.plot(x,y,'+',x1,y1,'k',linewidth=0.3)
```

```
runfile('E:/okul/SCO1/d_regression1.py', wdir='E:/okul/SCO1')
Reloaded modules: gauss

file name : a.txt
```

```
m= 2
x= [1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0]
y= [1.0, 4.0, 9.0, 16.0, 25.0, 36.0, 49.0, 64.0]
x1= [1.0, 1.0707070707070707, 1.1414141414141414, 1.2121212121212122, 1.2828282828282829, 1.3535353535353536,
1.4242424242424243, 1.4949494949494948, 1.5656565656565657, 1.6363636363636362, 1.7070707070707072, 1.7777777777777777,
1.8484848484848484, 1.9191919191919191, 1.9898989898989898, 2.0606060606060606, 2.1313131313131315, 2.202020202020202,
2.2727272727272725, 2.3434343434343434, 2.4141414141414144, 2.484848484848485, 2.5555555555555554, 2.6262626262626263,
2.696969696969697, 2.7676767676767673, 2.8383838383838382, 2.909090909090909, 2.9797979797979797, 3.0505050505050506,
3.121212121212121, 3.191919191919192, 3.2626262626262625, 3.333333333333333, 3.404040404040404, 3.4747474747474745,
3.5454545454545454, 3.616161616161616, 3.686868686868687, 3.7575757575757573, 3.8282828282828283, 3.898989898989899,
3.9696969696969697, 4.04040404040404, 4.111111111111111, 4.181818181818182, 4.252525252525253, 4.3232323232323235,
4.393939393939394, 4.4646464646464645, 4.535353535353535, 4.6060606060606055, 4.6767676767676765, 4.747474747474747,
4.818181818181818, 4.888888888888889, 4.959595959595959, 5.03030303030303, 5.101010101010101, 5.171717171717171,
5.242424242424242, 5.313131313131313, 5.383838383838384, 5.454545454545454, 5.525252525252525, 5.595959595959596,
5.666666666666666, 5.737373737373737, 5.808080808080808, 5.878787878787879, 5.949494949494949, 6.02020202020202,
6.090909090909091, 6.161616161616162, 6.232323232323232, 6.303030303030303, 6.373737373737374, 6.444444444444445,
6.515151515151515, 6.585858585858586, 6.656565656565657, 6.727272727272727, 6.797979797979798, 6.8686868686868685,
6.9393939393939394, 7.0101010101010095, 7.08080808080808, 7.151515151515151, 7.222222222222222, 7.292929292929292,
7.363636363636363, 7.434343434343434, 7.505050505050505, 7.575757575757575, 7.646464646464646, 7.717171717171717,
7.787878787878787, 7.858585858585858, 7.929292929292929, 8.0]
y1= [-6.000000000000027, -5.36363636363639, -4.727272727272753, -4.090909090909117, -3.45454545454548, -
2.818181818181843, -2.1818181818182047, -1.5454545454545716, -0.9090909090909314, -0.27272727272729647,
0.363636363636342, 0.9999999999999751, 1.6363636363636118, 2.272727272727252, 2.9090909090908887, 3.5454545454545254,
4.181818181818162, 4.818181818181799, 5.454545454545432, 6.090909090909072, 6.727272727272709, 7.3636363636363455,
7.999999999999979, 8.636363636363619, 9.272727272727252, 9.909090909090889, 10.545454545454529, 11.181818181818166,
11.818181818181802, 12.454545454545439, 13.090909090909076, 13.727272727272712, 14.363636363636349, 14.999999999999982,
15.636363636363622, 16.272727272727256, 16.909090909090896, 17.54545454545453, 18.18181818181817, 18.818181818181802,
19.454545454545443, 20.090909090909076, 20.727272727272716, 21.36363636363635, 21.99999999999999, 22.636363636363622,
23.272727272727263, 23.909090909090903, 24.545454545454536, 25.18181818181817, 25.818181818181802, 26.454545454545443,
27.090909090909083, 27.727272727272723, 28.363636363636356, 28.999999999999996, 29.63636363636363, 30.27272727272727,
30.909090909090903, 31.545454545454536, 32.18181818181817, 32.81818181818181, 33.45454545454545, 34.09090909090908,
34.72727272727272, 35.36363636363636, 35.999999999999986, 36.636363636363626, 37.272727272727266, 37.90909090909091,
38.54545454545455, 39.18181818181817, 39.81818181818181, 40.45454545454545, 41.09090909090909, 41.72727272727272,
42.36363636363636, 43.0, 43.63636363636364, 44.27272727272728, 44.90909090909091, 45.54545454545455, 46.18181818181819,
46.81818181818183, 47.45454545454545, 48.09090909090909, 48.727272727272734, 49.363636363636374, 50.000000000000014,
50.63636363636364, 51.27272727272728, 51.90909090909092, 52.54545454545456, 53.18181818181819, 53.81818181818183,
54.45454545454547, 55.09090909090909, 55.727272727272734, 56.363636363636374, 57.000000000000014]
```



Regression with quadratic polynomial

Consider the general polynomial curve fitting equation and take only first 3 column and row, equation will be for the quadratic polynomial $f(x)=a_0+a_1x+a_2x^2$

$$\begin{bmatrix} n & \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i^3 \\ \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i^3 & \sum_{i=1}^{n} x_i^4 \end{bmatrix} \begin{Bmatrix} a_0 \\ a_1 \\ a_2 \end{Bmatrix} = \begin{Bmatrix} \sum_{i=1}^{n} f(x_i) \\ \sum_{i=1}^{n} x_i f(x_i) \\ \sum_{i=1}^{n} x_i^2 f(x_i) \end{Bmatrix} \qquad (10.1-10)$$

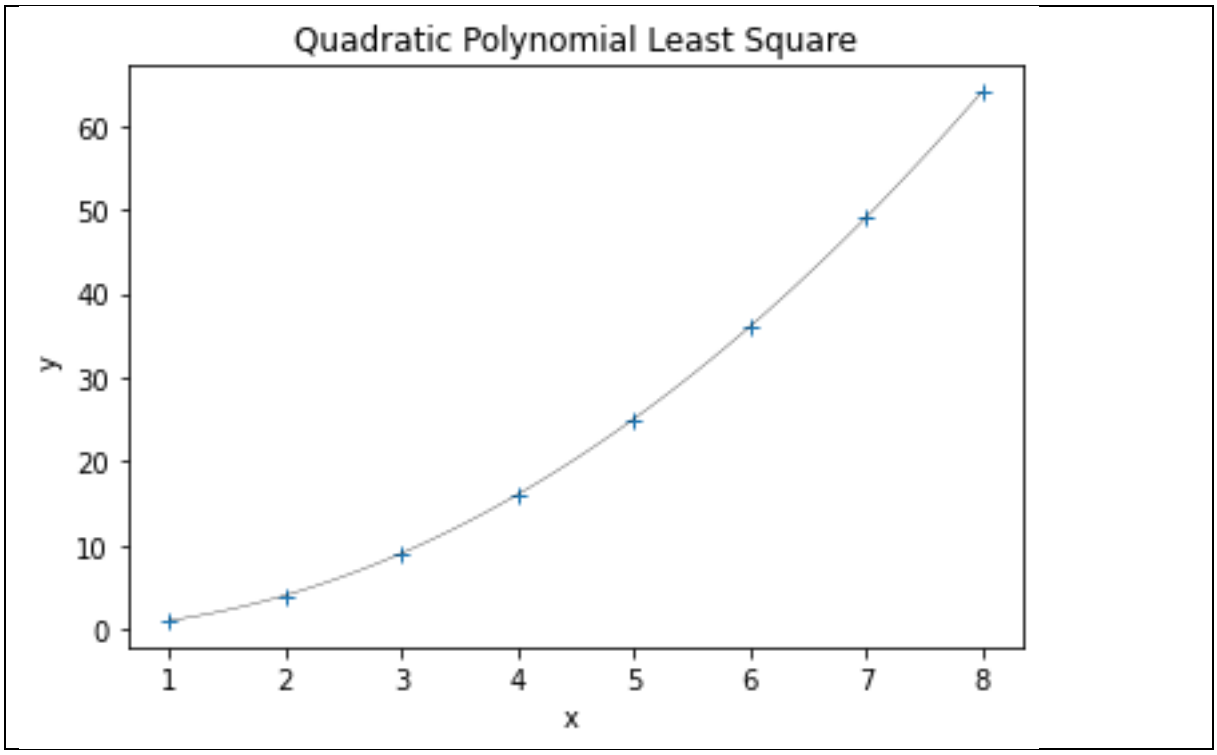
```
Sample data:
1 1
```

```
2 4
3 9
4 16
5 25
6 36
7 49
8 64
```

```
 # quadratic polynomial least square curve fitting
from math import *
import numpy as np
import matplotlib.pyplot as plt
import gauss

def poly_2(x,y):
    n=len(x)
    x1=0
    x2=0
    x3=0
    x4=0
    y1=0
    y2=0
    y3=0
    for i in range(n):
        x1+=x[i]
        x2+=x[i]*x[i]
        x3+=x[i]*x[i]*x[i]
        x4+=x[i]*x[i]*x[i]*x[i]
        y1+=y[i]
        y2+=x[i]*y[i]
        y3+=x[i]*x[i]*y[i]
    a= [[0 for i in range(3)] for j in range(3)];
    a[0][0]=n
    a[0][1]=x1
    a[0][2]=x2
    a[1][0]=x1
    a[1][1]=x2
    a[1][2]=x3
    a[2][0]=x2
    a[2][1]=x3
    a[2][2]=x4
    b=[0 for i in range(3)]
    b[0]=y1
    b[1]=y2
    b[2]=y3
    x=gauss.gauss(a,b)
    return x

def func(e,x):
    # this function calculates the value of
    # least square curve fitting function
    ff=e[0]+e[1]*x+e[2]*x*x
    return ff
def read_array2D(d):
    path = d
    file1 = open(path,'r')
    file1.seek(0,2)  #jumps to the end
    endLocation=file1.tell()
    file1.seek(0) #jumps to the start
    nn=0
    m=0
    while True:
        line=file1.readline()
        ee=line.split()
        nn+=1
        if  file1.tell()==endLocation:
            break
    m=len(ee)
    print("m=",m)
    a=[[0.0 for i in range(nn)] for j in range(m)]
    file1.seek(0) #jumps to the start
    for i in range(nn):
        line=file1.readline()
        ee=line.split()
        for j in range(m):
            a[j][i]=float(ee[j])
```

```
    file1.close()
    return a
a=read_array2D('a.txt')
n=len(a[0])
x =a[0]
print("x=",x)
y =a[1]
print("y=",y)
e=poly_2(x,y)
n1=100
x1=[0 for x in range(n1)]
y1=[0 for x in range(n1)]
xmax=max(x)
xmin=min(x)
dx=(xmax-xmin)/(n1-1)
for i in range(n1):
    x1[i]=xmin+i*dx
    y1[i]=func(e,x1[i])
print("x1=",x1)
print("y1=",y1)
plt.title('Quadratic Polynomial Least Square')
plt.xlabel('x ')
plt.ylabel('y ')
plt.plot(x,y,'+',x1,y1,'k',linewidth=0.3)
```

```
runfile('E:/okul/SCO1/d_regression_2.py', wdir='E:/okul/SCO1')
Reloaded modules: gauss
m= 2
x= [1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0]
y= [1.0, 4.0, 9.0, 16.0, 25.0, 36.0, 49.0, 64.0]
x1= [1.0, 1.0707070707070707, 1.1414141414141414, 1.2121212121212122, 1.2828282828282829, 1.3535353535353536,
1.4242424242424243, 1.4949494949494948, 1.5656565656565657, 1.6363636363636362, 1.7070707070707072, 1.7777777777777777,
1.8484848484848484, 1.9191919191919191, 1.9898989898989898, 2.0606060606060606, 2.1313131313131315, 2.202020202020202,
2.2727272727272725, 2.3434343434343434, 2.4141414141414144, 2.484848484848485, 2.5555555555555554, 2.6262626262626263,
2.696969696969697, 2.7676767676767673, 2.8383838383838382, 2.909090909090909, 2.9797979797979797, 3.0505050505050506,
3.121212121212121, 3.191919191919192, 3.2626262626262625, 3.333333333333333, 3.404040404040404, 3.4747474747474745,
3.5454545454545454, 3.616161616161616, 3.686868686868687, 3.7575757575757573, 3.8282828282828283, 3.898989898989899,
3.9696969696969697, 4.04040404040404, 4.111111111111111, 4.181818181818182, 4.252525252525253, 4.3232323232323235,
4.393939393939394, 4.4646464646464645, 4.535353535353535, 4.6060606060606055, 4.6767676767676765, 4.747474747474747,
4.818181818181818, 4.888888888888889, 4.959595959595959, 5.03030303030303, 5.101010101010101, 5.171717171717171,
5.242424242424242, 5.313131313131313, 5.383838383838384, 5.454545454545454, 5.525252525252525, 5.595959595959596,
5.666666666666666, 5.737373737373737, 5.808080808080808, 5.878787878787879, 5.949494949494949, 6.02020202020202,
6.090909090909091, 6.161616161616162, 6.232323232323232, 6.303030303030303, 6.373737373737374, 6.444444444444445,
6.515151515151515, 6.585858585858586, 6.656565656565657, 6.727272727272727, 6.797979797979798, 6.8686868686868685,
6.9393939393939394, 7.0101010101010095, 7.08080808080808, 7.151515151515151, 7.222222222222222, 7.292929292929292,
7.363636363636363, 7.434343434343434, 7.505050505050505, 7.575757575757575, 7.646464646464646, 7.717171717171717,
7.787878787878787, 7.858585858585858, 7.929292929292929, 8.0]
y1= [1.0, 1.146413631262116, 1.3028262422201817, 1.4692378328741966, 1.645648403224161, 1.8320579532700747,
2.028466483011938, 2.2348739924497494, 2.4512804815835123, 2.6776859504132227, 2.914090398938884, 3.1604938271604937,
3.416896235078053, 3.683297622691562, 3.95969799000102, 4.246097337006428, 4.542495663707785, 4.848892970105092,
5.165289256198346, 5.491684521987552, 5.828078767472708, 6.174471992653811, 6.530864197530863, 6.897255382103867,
7.273645546372818, 7.660034690337718, 8.056422813998571, 8.462809917355372, 8.87919600040812, 9.305581063156822,
9.74196510560147, 10.188348127742067, 10.644730129578614, 11.111111111111109, 11.587491072339557, 12.073870013263951,
12.570247933884298, 13.07662483420059, 13.593000714212835, 14.119375573921026, 14.655749413325172, 15.20212223242526,
15.758494031221304, 16.32486480971329, 16.90123456790123, 17.48760330578512, 18.083971023364963, 18.690337720640752,
19.306703397612484, 19.933068054280174, 20.569431690643803, 21.215794306703394, 21.87215590245893, 22.538516477910417,
23.214876033057852, 23.901234567901238, 24.597592082440563, 25.30394857667585, 26.020304050607084, 26.746658504234258,
27.48301193755739, 28.22936435057647, 28.985715743291504, 29.752066115702476, 30.528415467809406, 31.314763799612287,
32.11111111111111, 32.917457402305885, 33.733802673196614, 34.560146923783286, 35.39649015406591, 36.24283236404448,
37.099173553719005, 37.96551372308948, 38.8418528721559, 39.72819100091827, 40.62452810937659, 41.53086419753087,
42.44719926538108, 43.37353331292725, 44.30986634016937, 45.25619834710743, 46.21252933374145, 47.178859300071416,
48.15518824609734, 49.14151617181919, 50.13784307723701, 51.144168962350776, 52.160493827160494, 53.18681767166615,
54.22314049586777, 55.26946229976533, 56.32578308335884, 57.39210284664829, 58.46842158963371, 59.55473931231507,
60.651056014692365, 61.75737169676563, 62.87368635853484, 64.0]

In [192]:
```



Degree n polynomial

Polynomial curve fitting equation can be solved for any degree of polynomials. Restriction of the degree of polynomial is not needed. In practical applications degree of polynomial rarly taken above degree 10

Data (a.txt)

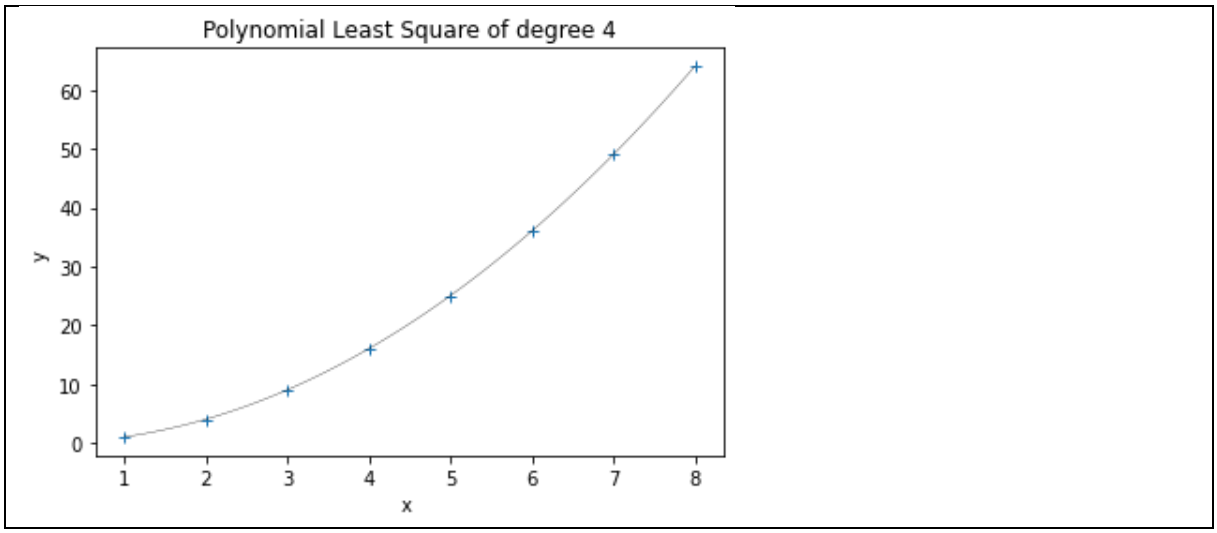
```
1 1
2 4
3 9
4 16
5 25
6 36
7 49
8 64
```

```
 # -*- coding: utf-8 -*-
"""
Created on Sat Aug 25 09:31:06 2018

@author: Mustafa Turhan Çoban
"""
from math import *
import numpy as np
import matplotlib.pyplot as plt
import gauss

def PolynomialLSQ(xi,yi,n):
    #Polynomial least square
    m=len(xi);
    print("l=",m);
    np1=n+1;
    print("np1=",np1);
    A = [[0 for x in range(np1)] for y in range(np1)];
    B =[0 for x in range(np1)];
    X =[0 for x in range(np1)];
    for i in range(np1):
        for j in range(np1):
```

```
            if i==0 and j==0:
                A[i][j]=m;
            else:
                for k in range(m):
                    pp=pow(xi[k],(i+j));
                    A[i][j]= A[i][j]+pow(xi[k],(i+j));
        for k in range(m):
            if i==0 and j==0:
                B[i]=  B[i]+yi[k];
            else:
                B[i]= B[i]+pow(xi[k],i)*yi[k];
    X=gauss.gauss(A,B);
    return X;

def func(e,x):
    # this function calculates the value of
    # least square curve fitting function
    n=len(e);
    if n!=0.0:
        ff=e[n-1];
        for i in range(n-2,-1,-1):
            ff=ff*x+e[i];
    else:
        ff=0;
    return ff;

def listPolynomialLSQ(E,xi,numberofmidpoints):
     #numberofmidpoints:  x--o--o--x--o--o--x
     n=len(xi)
     nn=(n-1)*(numberofmidpoints+1)+1;
     z= [[0 for x in range(nn)] for y in range(2)]
     dx=0.0
     k=0
     for i in range(n-1):
         z[0][k]=xi[i]
         z[1][k]=funcPolynomialLSQ(E,z[0][k])
         k=k+1
         for j in range(numberofmidpoints):
             dx=(xi[i+1]-xi[i])/(numberofmidpoints+1.0)
             z[0][k]=z[0][k-1]+dx
             z[1][k]=funcPolynomialLSQ(E,z[0][k])
             k=k+1
         z[0][k]=xi[i+1];z[1][k]=funcPolynomialLSQ(E,z[0][k])
     return z;

def read_array2D(d):
    path = d
    file1 = open(path,'r')
    file1.seek(0,2)  #jumps to the end
    endLocation=file1.tell()
    file1.seek(0) #jumps to the start
    nn=0
    m=0
    while True:
        line=file1.readline()
        ee=line.split()
        nn+=1
        if  file1.tell()==endLocation:
            break
    m=len(ee)
    print("m=",m)
    a=[[0.0 for i in range(nn)] for j in range(m)]
    file1.seek(0) #jumps to the start
    for i in range(nn):
        line=file1.readline()
        ee=line.split()
        for j in range(m):
            a[j][i]=float(ee[j])
    file1.close()
    return a

a=read_array2D('a.txt')
n=len(a[0])
x =a[0]
print("x=",x)
y =a[1]
```

```
print("y=",y)
m=4
e=PolynomialLSQ(x,y,m)
n1=100
x1=[0 for x in range(n1)]
y1=[0 for x in range(n1)]
xmax=max(x)
xmin=min(x)
dx=(xmax-xmin)/(n1-1)
for i in range(n1):
    x1[i]=xmin+i*dx
    y1[i]=func(e,x1[i])
print("x1=",x1)
print("y1=",y1)
plt.title("Polynomial Least Square of degree "+str(m))
plt.xlabel('x ')
plt.ylabel('y ')
plt.plot(x,y,'+',x1,y1,'k',linewidth=0.3)
```

```
runfile('E:/okul/SCO1/d_regression_n.py', wdir='E:/okul/SCO1')
Reloaded modules: gauss
m= 2
x= [1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0]
y= [1.0, 4.0, 9.0, 16.0, 25.0, 36.0, 49.0, 64.0]
l= 8
np1= 5
x1= [1.0, 1.0707070707070707, 1.1414141414141414, 1.2121212121212122, 1.2828282828282829, 1.3535353535353536,
1.4242424242424243, 1.4949494949494948, 1.5656565656565657, 1.6363636363636362, 1.7070707070707072, 1.7777777777777777,
1.8484848484848484, 1.9191919191919191, 1.9898989898989898, 2.0606060606060606, 2.1313131313131315, 2.202020202020202,
2.2727272727272725, 2.3434343434343434, 2.4141414141414144, 2.484848484848485, 2.5555555555555554, 2.6262626262626263,
2.696969696969697, 2.7676767676767673, 2.8383838383838382, 2.909090909090909, 2.9797979797979797, 3.0505050505050506,
3.121212121212121, 3.191919191919192, 3.2626262626262625, 3.333333333333333, 3.404040404040404, 3.4747474747474745,
3.5454545454545454, 3.616161616161616, 3.686868686868687, 3.7575757575757573, 3.8282828282828283, 3.898989898989899,
3.9696969696969697, 4.04040404040404, 4.111111111111111, 4.181818181818182, 4.252525252525253, 4.3232323232323235,
4.393939393939394, 4.4646464646464645, 4.535353535353535, 4.6060606060606055, 4.6767676767676765, 4.747474747474747,
4.818181818181818, 4.888888888888889, 4.959595959595959, 5.03030303030303, 5.101010101010101, 5.171717171717171,
5.242424242424242, 5.313131313131313, 5.383838383838384, 5.454545454545454, 5.525252525252525, 5.595959595959596,
5.666666666666666, 5.737373737373737, 5.808080808080808, 5.878787878787879, 5.949494949494949, 6.02020202020202,
6.090909090909091, 6.161616161616162, 6.232323232323232, 6.303030303030303, 6.373737373737374, 6.444444444444445,
6.515151515151515, 6.585858585858586, 6.656565656565657, 6.727272727272727, 6.797979797979798, 6.8686868686868685,
6.9393939393939394, 7.0101010101010095, 7.08080808080808, 7.151515151515151, 7.222222222222222, 7.292929292929292,
7.363636363636363, 7.434343434343434, 7.505050505050505, 7.575757575757575, 7.646464646464646, 7.717171717171717,
7.787878787878787, 7.858585858585858, 7.929292929292929, 8.0]
y1= [1.0, 1.146413631262116, 1.3028262422201817, 1.4692378328741966, 1.645648403224161, 1.8320579532700747,
2.028466483011938, 2.2348739924497494, 2.4512804815835123, 2.6776859504132227, 2.914090398938884, 3.1604938271604937,
3.416896235078053, 3.683297622691562, 3.95969799000102, 4.246097337006428, 4.542495663707785, 4.848892970105092,
5.165289256198346, 5.491684521987552, 5.828078767472708, 6.174471992653811, 6.530864197530863, 6.897255382103867,
7.273645546372818, 7.660034690337718, 8.056422813998571, 8.462809917355372, 8.87919600040812, 9.305581063156822,
9.74196510560147, 10.188348127742067, 10.644730129578614, 11.111111111111109, 11.587491072339557, 12.073870013263951,
12.570247933884298, 13.07662483420059, 13.593000714212835, 14.119375573921026, 14.655749413325172, 15.20212223242526,
15.758494031221304, 16.32486480971329, 16.90123456790123, 17.48760330578512, 18.083971023364963, 18.690337720640752,
19.306703397612484, 19.933068054280174, 20.569431690643803, 21.215794306703394, 21.87215590245893, 22.538516477910417,
23.214876033057852, 23.901234567901238, 24.597592082440563, 25.30394857667585, 26.020304050607084, 26.746658504234258,
27.48301193755739, 28.22936435057647, 28.985715743291504, 29.752066115702476, 30.528415467809406, 31.314763799612287,
32.11111111111111, 32.917457402305885, 33.733802673196614, 34.560146923783286, 35.39649015406591, 36.24283236404448,
37.099173553719005, 37.96551372308948, 38.8418528721559, 39.72819100091827, 40.62452810937659, 41.53086419753087,
42.44719926538108, 43.37353331292725, 44.30986634016937, 45.25619834710743, 46.21252933374145, 47.178859300071416,
48.15518824609734, 49.14151617181919, 50.13784307723701, 51.144168962350776, 52.160493827160494, 53.18681767166615,
54.22314049586777, 55.26946229976533, 56.32578308335884, 57.39210284664829, 58.46842158963371, 59.55473931231507,
60.651056014692365, 61.75737169676563, 62.87368635853484, 64.0]
```



A general function version of the least square program will look like :

$$\begin{bmatrix} \sum_{i=1}^{n}\phi_0(x_i)\phi_0(x_i) & \sum_{i=1}^{n}\phi_0(x_i)\phi_1(x_i) & \sum_{i=1}^{n}\phi_0(x_i)\phi_2(x_i) & \cdots & \sum_{i=1}^{n}\phi_0(x_i)\phi_n(x_i) \\ \sum_{i=1}^{n}\phi_1(x_i)\phi_0(x_i) & \sum_{i=1}^{n}\phi_1(x_i)\phi_1(x_i) & \sum_{i=1}^{n}\phi_1(x_i)\phi_2(x_i) & \cdots & \sum_{i=1}^{n}\phi_2(x_i)\phi_n(x_i) \\ \sum_{i=1}^{n}\phi_2(x_i)\phi_0(x_i) & \sum_{i=1}^{n}\phi_2(x_i)\phi_1(x_i) & \sum_{i=1}^{n}\phi_2(x_i)\phi_2(x_i) & \cdots & \sum_{i=1}^{n}\phi_2(x_i)\phi_n(x_i) \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \sum_{i=1}^{n}\phi_n(x_i)\phi_0(x_i) & \sum_{i=1}^{n}\phi_n(x_i)\phi_1(x_i) & \sum_{i=1}^{n}\phi_n(x_i)\phi_2(x_i) & \cdots & \sum_{i=1}^{n}\phi_n(x_i)\phi_n(x_i) \end{bmatrix} \begin{Bmatrix} a_0 \\ a_1 \\ a_2 \\ \cdots \\ a_n \end{Bmatrix} = \begin{Bmatrix} \sum_{i=1}^{n}\phi_0(x_i)f(x_i) \\ \sum_{i=1}^{n}\phi_1(x_i)f(x_i) \\ \sum_{i=1}^{n}\phi_2(x_i)f(x_i) \\ \cdots \\ \sum_{i=1}^{n}\phi_n(x_i)f(x_i) \end{Bmatrix} \quad (10.1-11)$$

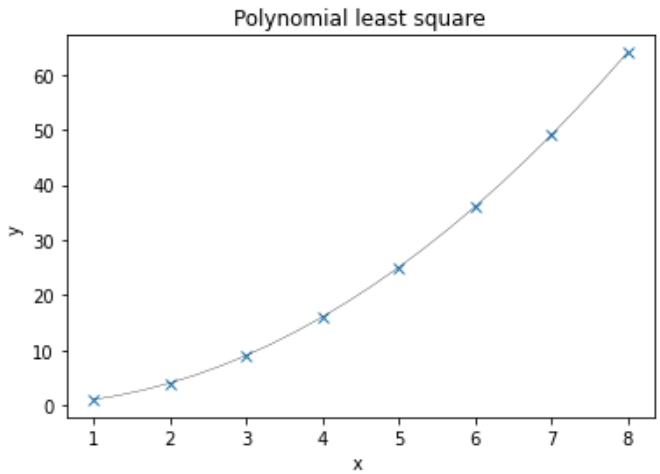
Program 6.1-4 General least square curve fitting (Python version)

```
 # -*- coding: utf-8 -*-
"""
Created on Sat Aug 25 09:31:06 2018

@author: Mustafa Turhan Çoban
"""
from math import *
from gauss import *;
import numpy as np;
import matplotlib.pyplot as plt;
from abc import ABC, abstractmethod


class f_xr(ABC):

    @abstractmethod
    def func(self,x,equation_ref):
        pass;

class f1(f_xr):
  def func(this,x,i):
      xx=0;
      if i==0: xx=1.0;
      elif i==1:xx=x;
      elif i==2:xx=x*x;
      elif i==3:xx=x*x*x;
      elif i==4:xx=x*x*x*x;
      elif i==5:xx=x*x*x*x*x;
      elif i==6:xx=x*x*x*x*x*x;
      return xx;

def PolynomialLSQ(f,xi,yi,n):
    #Polynomial least square
    m=len(xi);
    print("l=",m);
    np1=n+1;
```

```
    print("np1=",np1);
    A = [[0 for x in range(np1)] for y in range(np1)];
    B =[0 for x in range(np1)];
    X =[0 for x in range(np1)];
    for i in range(np1):
        for j in range(np1):
            for k in range(m):
                    pp=pow(xi[k],(i+j));
                    A[i][j]= A[i][j]+f.func(xi[k],i)*f.func(xi[k],j);
        for k in range(m):
            B[i]= B[i]+f.func(xi[k],i)*yi[k];
    X=gauss(A,B);
    return X;

def funcPolynomialLSQ(f,e,x):
    # this function calculates the value of
    # least square curve fitting function
    n=len(e);
    ff=0;
    if n!=0.0:
        for i in range(n-1,-1,-1):
            ff=ff+e[i]*f.func(x,i);
    return ff;

def listPolynomialLSQ(f,E,xi,aradegersayisi):
     #aradegersayisi:  x--o--o--x--o--o--x zincirinde x deneysel noktalar ise
                    # ara değer sayisi 2 dir
     n=len(xi);
     nn=(n-1)*(aradegersayisi+1)+1;
     z= [[0 for x in range(nn)] for y in range(2)];
     dx=0;
     k=0;
     for i in range(n-1):
         z[0][k]=xi[i];
         z[1][k]=funcPolynomialLSQ(f,E,z[0][k]);
         k=k+1;
         for j in range(aradegersayisi):
             dx=(xi[i+1]-xi[i])/(aradegersayisi+1.0);
             z[0][k]=z[0][k-1]+dx;
             z[1][k]=funcPolynomialLSQ(f,E,z[0][k]);
             k=k+1;
         z[0][k]=xi[i+1];z[1][k]=funcPolynomialLSQ(f,E,z[0][k]);
     return z;

def read_array2D(d):
    path = d
    file1 = open(path,'r')
    file1.seek(0,2)  #jumps to the end
    endLocation=file1.tell()
    file1.seek(0) #jumps to the start
    nn=0
    m=0
    while True:
        line=file1.readline()
        ee=line.split()
        nn+=1
        if  file1.tell()==endLocation:
            break
    m=len(ee)
    print("m=",m)
    a=[[0.0 for i in range(nn)] for j in range(m)]
    file1.seek(0) #jumps to the start
    for i in range(nn):
        line=file1.readline()
        ee=line.split()
        for j in range(m):
            a[j][i]=float(ee[j])
    file1.close()
    return a

a=read_array2D('a.txt')
print("a",a)
x=a[0]
y=a[1]
print("x",x)
print("y",y)
```

```
f=f1()
E=PolynomialLSQ(f,x,y,2);
print("E=",E)
Z=listPolynomialLSQ(f,E,x,2);
print("Z=",Z[0])
plt.title('Polynomial least square')
plt.xlabel('x ')
plt.ylabel('y ')
plt.plot(x,y,'x',Z[0],Z[1],'k',linewidth=0.3)
```

```
runfile('E:/okul/SCO1/gen_poly_least_sqr.py', wdir='E:/okul/SCO1')
Reloaded modules: gauss
m= 2
a [[1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0], [1.0, 4.0, 9.0, 16.0, 25.0, 36.0, 49.0, 64.0]]
x [1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0]
y [1.0, 4.0, 9.0, 16.0, 25.0, 36.0, 49.0, 64.0]
l= 8
np1= 3
E= [0.0, -0.0, 1.0]
Z= [1.0, 1.3333333333333333, 1.6666666666666665, 2.0, 2.3333333333333335, 2.666666666666667, 3.0, 3.3333333333333335,
3.666666666666667, 4.0, 4.333333333333333, 4.666666666666666, 5.0, 5.333333333333333, 5.666666666666666, 6.0,
6.333333333333333, 6.666666666666666, 7.0, 7.333333333333333, 7.666666666666666, 8.0]
```



An excel or Octave version of polynomial least square curve fitting is given below. In order to carry out curve fitting, data (x,y) and x^2,x^3,x^4,.. should be defined and LINEST function (DOT in turkish version) should be used with ctrl-shift-enter

=DOT(G2:G9;B2:F9;DOĞRU;DOĞRU)

=LINEST(G2:G9;B2:F9;TRUE;TRUE)

x	x^2	x^3	x^4	x^5	y	f(x)=a0+a1*x+..
1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1
2.0000	4.0000	4.0000	16.0000	32.0000	4.0000	4
3.0000	9.0000	9.0000	81.0000	243.0000	9.0000	9
4.0000	16.0000	16.0000	256.0000	1024.0000	16.0000	16
5.0000	25.0000	25.0000	625.0000	3125.0000	25.0000	25
6.0000	36.0000	36.0000	1296.0000	7776.0000	36.0000	36
7.0000	49.0000	49.0000	2401.0000	16807.0000	49.0000	49
8.0000	64.0000	64.0000	4096.0000	32768.0000	64.0000	64

a0	a1	a2	a3	a4	a5	turkish	english	
6.91E-19	-9.81031E-18	1	0	0	-	DOT	LINEST	ctrl-shift-enter

An excel or Octave version of polynomial least square curve fitting with our own gauss elimination linear system of equation solution is given below:

n	x	y	x^2	x^3	x^4	x*y	x^2*y
1	1	1.2452	1	1	1	1.2452	1.2452

1	2	4.34534	4	8	16	8.69068	17.38136
1	3	8.9746	9	27	81	26.9238	80.7714
1	4	15.2	16	64	256	60.8	243.2
1	5	22	25	125	625	110	550
1	6	34.8975	36	216	1296	209.385	1256.31
1	7	50.12378	49	343	2401	350.86646	2456.0652
1	8	63.61761	64	512	4096	508.94088	4071.527
1	9	81.74675	81	729	6561	735.72075	6621.4868
1	10	97	100	1000	10000	970	9700
10	55	379.15078	385	3025	25333	2982.5728	24997.987

GAUSS ELIMINATION 3 UNKNOWN

A B

10	55	385	379.15078
55	385	3025	2982.5728
385	3025	25333	24997.987

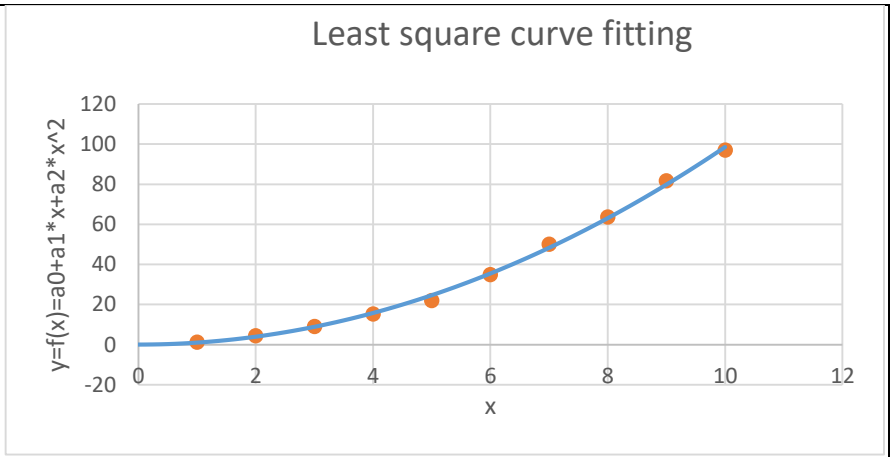
	10	55	385	379.15078
5.5	55	385	3025	2982.5728
38.5	385	3025	25333	24997.987

	10	55	385	379.15078
7.8571429	55	385	3025	2982.5728
	0	3025	25333	24997.987

x0	-0.0411622
x1	-0.0062948
x2	0.9867756

created data

x	y
0	-0.0411622
0.25	0.0189376
0.5	0.2023843
0.75	0.509178
1	0.9393186
1.25	1.4928062
1.5	2.1696408
1.75	2.9698223
2	3.8933507
2.25	4.9402261
2.5	6.1104485
2.75	7.4040178
3	8.820934
3.25	10.361197
3.5	12.024807
3.75	13.811765
4	15.722069
4.25	17.75572



10.2 LINEAR CURVE FITTING : ORTOGONAL POLYNOMIAL LEAST SQUARE

In the previous subsection , polynomial least square method is investigated. In order to find coefficient of polynomial least square a matrix solution process should be carried out. Furthermore,

the equation tends to create ill matrices, so that chance of having an error in the coefficient while solving the matrix is high. Orthogonal polynomials has the following usefull property

$$\begin{Bmatrix} p_i(x)p_j(x) = 0 & if \;\; i \neq j \\ p_i(x)p_j(x) = A & if \;\; i = j \end{Bmatrix} \quad (10.2-1)$$

Now Assume to have a jth order orthogonal polynomial$p_j(x)$. And assume that least square polynomial equation is

$$y_m(x) = \sum_{j=0}^{m} b_j^{(m)} p_j(x) \quad (10.2-2)$$

If x_i, f_i i=1,…,n is given as a data to be curve fit, The minimization function can be defined as

$$H\left(b_0^{(m)},..,b_0^{(m)}\right) = \sum_{i=1}^{n} w(x_i)[f(x_i) - y_m(x)]^2 = \sum_{i=1}^{n} w(x_i)\left[f(x_i) - \sum_{j=0}^{m} b_j^{(m)} p_j(x)\right]^2 \quad (10.2-3)$$

In order to minimize the function, derivative can be set as equal to zero

$$\frac{\partial H\left(b_0^{(m)},..,b_0^{(m)}\right)}{\partial b_k^{(m)}} = \sum_{i=1}^{n} w(x_i)[f(x_i) - y_m(x)]^2$$

$$= \sum_{i=1}^{n} w(x_i)\left[f(x_i) - \sum_{j=0}^{m} b_j^{(m)} p_j(x)\right] p_k(x) = 0 \;\; k = 0,..,m \quad (9.2-4)$$

for a shorter notation new variables can be defined as:

$$d_{jk} \sum_{i=1}^{n} w(x_i)p_j(x_i)\, p_k(x_i) \quad (10.2-5)$$

$$\omega_{jk} = \sum_{i=1}^{n} w(x_i)p_n(x_i)f(x_i) \quad (10.2-8)$$

$$\sum_{j=0}^{m} d_{jk}\left[b_j^{(m)}\right] = \omega_{jk} \quad k = 0,..,m \quad (10.2-7)$$

This equation came to the following matrix form

$$\begin{bmatrix} \sum_{i=1}^{n} w(x_i)p_0(x_i)\, p_0(x_i) & \sum_{i=1}^{n} w(x_i)p_0(x_i)\, p_1(x_i) & \sum_{i=1}^{n} w(x_i)p_0(x_i)\, p_2(x_i) & \cdots & \sum_{i=1}^{n} w(x_i)p_0(x_i)\, p_n(x_i) \\ \sum_{i=1}^{n} w(x_i)p_1(x_i)\, p_0(x_i) & \sum_{i=1}^{n} w(x_i)p_1(x_i)\, p_1(x_i) & \sum_{i=1}^{n} w(x_i)p_1(x_i)\, p_2(x_i) & \cdots & \sum_{i=1}^{n} w(x_i)p_1(x_i)\, p_n(x_i) \\ \sum_{i=1}^{n} w(x_i)p_2(x_i)\, p_0(x_i) & \sum_{i=1}^{n} w(x_i)p_2(x_i)\, p_1(x_i) & \sum_{i=1}^{n} w(x_i)p_2(x_i)\, p_2(x_i) & \cdots & \sum_{i=1}^{n} w(x_i)p_2(x_i)\, p_n(x_i) \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \sum_{i=1}^{n} w(x_i)p_n(x_i)\, p_0(x_i) & \sum_{i=1}^{n} w(x_i)p_n(x_i)\, p_1(x_i) & \sum_{i=1}^{n} w(x_i)p_n(x_i)\, p_2(x_i) & \cdots & \sum_{i=1}^{n} w(x_i)p_n(x_i)\, p_n(x_i) \end{bmatrix} \begin{Bmatrix} b_0 \\ b_1 \\ b_2 \\ \cdots \\ b_n \end{Bmatrix} =$$

$$\begin{Bmatrix} \sum_{i=1}^{n} w(x_i)p_0(x_i)f(x_i) \\ \sum_{i=1}^{n} w(x_i)p_1(x_i)f(x_i) \\ \sum_{i=1}^{n} w(x_i)p_2(x_i)f(x_i) \\ \cdots \\ \sum_{i=1}^{n} w(x_i)p_n(x_i)f(x_i) \end{Bmatrix} \text{(10.2-8a)}$$

Writing the equation this way does not give any advantage, But considering that $p_k(x_i)$ is an orthogonal function, due to the propertiy of the orthogonal functions. In this case the equation that formed a matrix converted to a simple multiplication equation becomes

$$\begin{bmatrix} \sum_{i=1}^{n} w(x_i)p_0(x_i)\, p_0(x_i) & & & & \\ & \sum_{i=1}^{n} w(x_i)p_1(x_i)\, p_1(x_i) & & & \\ & & \sum_{i=1}^{n} w(x_i)p_2(x_i)\, p_2(x_i) & & \\ & & & \cdots & \\ & & & & \sum_{i=1}^{n} w(x_i)p_n(x_i)\, p_n(x_i) \end{bmatrix} \begin{Bmatrix} b_0 \\ b_1 \\ b_2 \\ \cdots \\ b_n \end{Bmatrix} =$$

$$\begin{Bmatrix} \sum_{i=1}^{n} w(x_i)p_0(x_i)f(x_i) \\ \sum_{i=1}^{n} w(x_i)p_1(x_i)f(x_i) \\ \sum_{i=1}^{n} w(x_i)p_2(x_i)f(x_i) \\ \cdots \\ \sum_{i=1}^{n} w(x_i)p_n(x_i)f(x_i) \end{Bmatrix} \text{(10.2-8b)}$$

$$d_{kk}\left[b_j^{(m)}\right] = \omega_{kk} \quad k = 0,\dots,m \quad (10.2-8c)$$

To solve $b_j^{(m)}$

$$\left[b_j^{(m)}\right] = \frac{\omega_{kk}}{d_{kk}} = \frac{\sum_{i=1}^{n} w(x_i)p_k(x_i)f(x_i)}{\sum_{i=1}^{n} w(x_i)p_k(x_i)\, p_k(x_i)} \quad k = 0,..,m \quad (10.2-8d)$$

Anthony Ralston,Philip Rabinowitz orthogonal polynomial least square:

Several different orthogonal polynomials are available. The orthogonal polynomial that we will utilise in our first program has the following form. This equation is taken from "A First Course in Numerical Analysis, Anthony Ralston,Philip Rabinowitz, Mc Graw Hill ISBN 0-07-051158-6" [4]

$$p_{j+1}(x) = (x - \alpha_{j+1})p_j(x) - \beta_j p_{j-1}(x) \quad j = 0,1,\ldots \quad (10.2-9)$$

$p_0(x) = 1 \quad p_{-1}(x) = 0$

α_{j+1} and β_j are polynomial constants to be calculated.

$$\beta_k = \frac{\sum_{i=1}^{n} w(x_i)x_i p_{k-1}(x_i)\, p_k(x_i)}{\sum_{i=1}^{n} w(x_i)x_i p_{k-1}(x_i)\, p_{k-1}(x_i)} = \frac{\sum_{i=1}^{n} w(x_i)p_k(x_i)\, p_k(x_i)}{\sum_{i=1}^{n} w(x_i)p_{k-1}(x_i)\, p_{k-1}(x_i)} \quad 10.2-10)$$

$$\alpha_k = \frac{\sum_{i=1}^{n} w(x_i)x_i p_k(x_i)\, p_k(x_i)}{\sum_{i=1}^{n} w(x_i)p_{k-1}(x_i)\, p_{k-1}(x_i)} \quad (10.2-11)$$

Least square algorithm can be given as follows:

$$y_m(x) = \sum_{j=0}^{m} b_j^{(m)} p_j(x) \qquad (10.2.12)$$

$$b_j = \frac{\omega_j}{\gamma_j}$$

$$\omega_j = \sum_{i=1}^{n} w(x_i)p_j(x_i)f(x_i)$$

$$\gamma_j = \sum_{i=1}^{n} w(x_i)p_j(x_i)p_j(x_i)$$

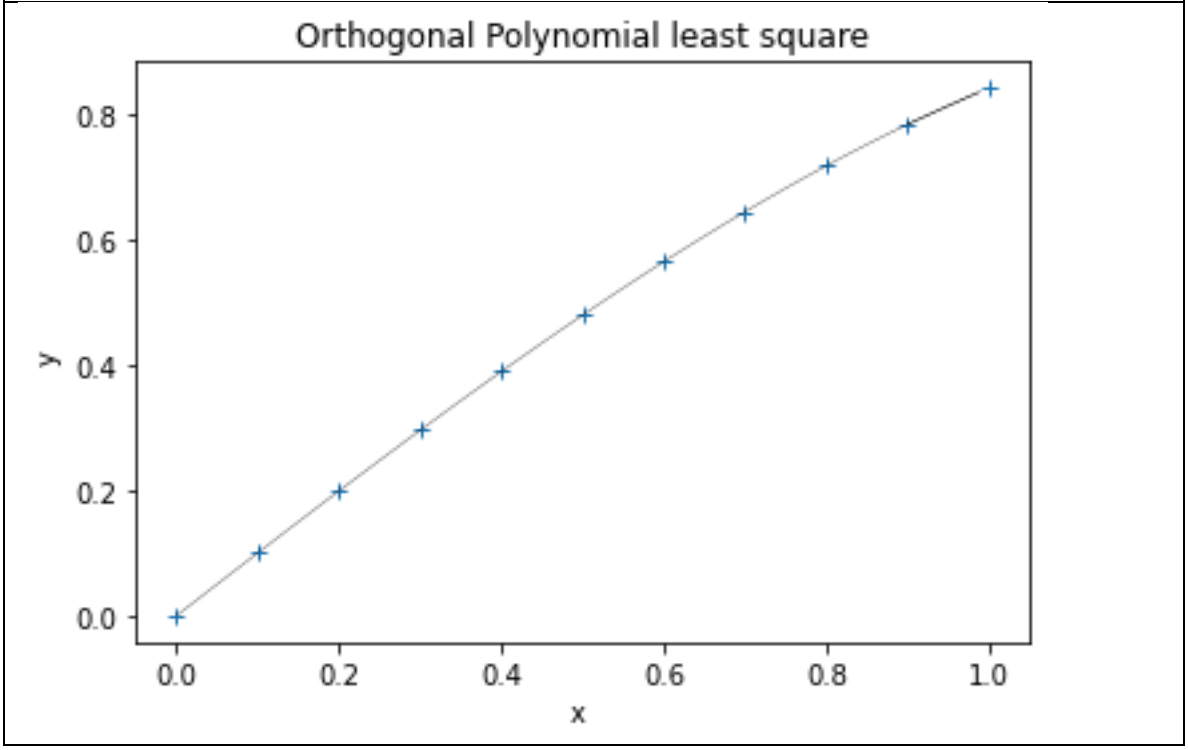
Because Orthogonal polynomial equations are not solving any system of equations, roundoff errors due to matrix solutions will not be existed. But the polynomial equation used has a more complex form compare to polynomial form.

```
from math import *
import numpy as np;
import matplotlib.pyplot as plt

def  OPEKK(xi,fi,m):
   #ortogonal polinomial least square method
   #Referance : A First Course in Numerical Analysis
   #Anthony Ralston,Philip Rabinowitz, Mc Graw Hill ISBN 0-07-051158-6
   #m degree of polynomial xi fi input data
   # int i,j,k;
   n=len(xi)
   mp2=n+2
   mp1=n+1
   p=[[0.0 for j in range(n)] for i in range(mp2)]
   gamma=[0.0 for j in range(mp1)]
   beta=[0.0 for j in range(mp1)]
   omega=[0.0 for j in range(mp1)]
   alpha=[0.0 for j in range(mp1)]
   b=[0.0 for j in range(mp1)]
   wi=[0.0 for j in range(n)]
   a=[[0.0 for j in range(mp1)] for i in range(3)]
   for i in range(n):
      p[1][i]=1.0
      p[0][i]=0.0
      wi[i]=1.0
   gamma[0]=0
   for i in range(n):
      gamma[0]=gamma[0]+wi[i]
   beta[0]=0.0
   for j in range(m+1):
      omega[j]=0;
      for i in range(n):
         omega[j]=omega[j]+wi[i]*fi[i]*p[j+1][i]
      b[j]=omega[j]/gamma[j]
      if j != m:
         alpha[j+1]=0
         for i in range(n):
```

```
            alpha[j+1]+=wi[i]*xi[i]*p[j+1][i]*p[j+1][i]/gamma[j]
        for i in range(n):
            p[j+2][i]=(xi[i]-alpha[j+1])*p[j+1][i]-beta[j]*p[j][i]
        gamma[j+1]=0
        for i in range(n):
            gamma[j+1]+=wi[i]*p[j+2][i]*p[j+2][i]
        beta[j+1]=gamma[j+1]/gamma[j]
    for j in range(m+1):
      a[0][j]=b[j]
      a[1][j]=alpha[j]
      a[2][j]=beta[j]
    return a


def func(a,x):
    #polinom değerleri hesaplama fonksiyonu
    yy=0
    k=0
    m=len(a[0])-1
    mp2=m+2;
    q=[0.0 for j in range(mp2)]
    for k in range(m-1,-1,-1):
       q[k]=a[0][k]+(x-a[1][k+1])*q[k+1]-a[2][k+1]*q[k+2]
       yy=q[k]
    return yy;

def funcOPEKK(xi,yi,polynomialcoefficient,numberofmidpoints):
    #number of midpoints:  x--o--o--x--o--o--x  x data points midpoints are 2
    n=len(xi)
    nn=(n-1)*(numberofmidpoints+1)+1
    z=[[0.0 for j in range(nn)] for i in range(2)]
    E=OPEKK(xi,yi,polynomialcoefficient);
    dx=0.0
    k=0
    for i in range(n-1):
       z[0][k]=xi[i]
       z[1][k]=func(E,z[0][k])
       k=k+1
       for j in range(numberofmidpoints):
           dx=(xi[i+1]-xi[i])/(float(numberofmidpoints)+1.0)
           z[0][k]=z[0][k-1]+dx
           z[1][k]=func(E,z[0][k])
           k=k+1
       z[0][k]=xi[i]
       z[1][k]=func(E,z[0][k])
    return z

x=[0.0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0]
y=[0.0,0.099833,0.1986693,0.2955202,0.38941834,0.479425,0.5646424,0.6442176,0.717356,0.783326,0.84147]
z3=funcOPEKK(x,y,3,8);
print("z3",z3)
plt.title('Orthogonal Polynomial least square')
plt.xlabel('x ')
plt.ylabel('y ');
plt.plot(x,y,'+',z3[0],z3[1],'k',linewidth=0.3,)
```

```
runfile('E:/okul/SCO1/OPEKK.py', wdir='E:/okul/SCO1')
z3 [[0.0, 0.011111111111111112, 0.022222222222222223, 0.03333333333333333, 0.044444444444444446, 0.05555555555555556,
0.06666666666666667, 0.07777777777777778, 0.08888888888888889, 0.1, 0.11111111111111112, 0.12222222222222223,
0.13333333333333333, 0.14444444444444443, 0.15555555555555553, 0.16666666666666663, 0.17777777777777773,
0.18888888888888883, 0.2, 0.2111111111111111, 0.2222222222222222, 0.2333333333333333, 0.2444444444444444,
0.25555555555555554, 0.26666666666666666, 0.2777777777777778, 0.2888888888888889, 0.3, 0.3111111111111111,
0.32222222222222224, 0.33333333333333337, 0.3444444444444445, 0.3555555555555556, 0.36666666666666675,
0.3777777777777779, 0.388888888888889, 0.4, 0.4111111111111115, 0.4222222222222223, 0.4333333333333334,
0.44444444444444453, 0.45555555555555566, 0.4666666666666668, 0.4777777777777779, 0.48888888888888904, 0.5,
0.5111111111111111, 0.5222222222222221, 0.5333333333333332, 0.5444444444444443, 0.5555555555555554,
0.5666666666666664, 0.5777777777777775, 0.5888888888888886, 0.6, 0.611111111111111, 0.6222222222222221,
0.6333333333333332, 0.6444444444444443, 0.6555555555555553, 0.6666666666666664, 0.6777777777777775,
0.6888888888888886, 0.7, 0.711111111111111, 0.7222222222222221, 0.7333333333333332, 0.7444444444444442,
0.7555555555555553, 0.7666666666666664, 0.7777777777777775, 0.7888888888888885, 0.8, 0.8111111111111111,
0.8222222222222222, 0.8333333333333333, 0.8444444444444443, 0.8555555555555554, 0.8666666666666665,
0.8777777777777775, 0.8888888888888886, 0.9, 0.9111111111111111, 0.9222222222222222, 0.9333333333333332,
0.9444444444444443, 0.9555555555555554, 0.9666666666666665, 0.9777777777777775, 0.9888888888888886, 0.9], [-
0.0001333131468531090 2, 0.011022273501971321, 0.02217203082505974, 0.03331477390831393, 0.04444931783763565,
0.05557447769892668, 0.06668906857808862, 0.0777919055610234, 0.0888818037336327, 0.09995757818181823,
0.1110180439914818, 0.1220620162485252, 0.1330883100388500 8, 0.1440957404483582 7, 0.1550831225629514 5,
```

0.16604927146853146, 0.17699300225099998, 0.1879131299962589, 0.19880846979020986, 0.20967783671875456, 0.22052004586779472, 0.23133391232323228, 0.24211825117096894, 0.25287187749690637, 0.2635936063869464, 0.2742822529269907, 0.2849366322029412, 0.2955555593006993, 0.30613784930616705, 0.31668231730524615, 0.32718777838383845, 0.3376530476278455, 0.3480769401231691, 0.35845827095571103, 0.3687958552113731, 0.379088507976057, 0.38933504433566435, 0.3995342793760972, 0.40968502818325714, 0.4197861058430459, 0.42983632744136524, 0.439834508064117, 0.4497794627972029, 0.4596700067265246, 0.46950495493798394, 0.4792831225174825, 0.48900332455092227, 0.49866437612420494, 0.5082650923232322, 0.5178042882339059, 0.5272807789421277, 0.5366933795337994, 0.5460409050948227, 0.5553221707110993, 0.5645359914685315, 0.5736811824530201, 0.5827565587504675, 0.5917609354467753, 0.6006931276278452, 0.6095519503795791, 0.6183362187878786, 0.6270447479386455, 0.6356763529177815, 0.6442298488111887, 0.6527040507047684, 0.6610977736844225, 0.6694098328360527, 0.6776390432455609, 0.6857842199988488, 0.693844178181818, 0.7018177328803704, 0.7097036991804079, 0.7175008921678322, 0.7252081269285447, 0.7328242185484475, 0.740347982113442, 0.7477782327094304, 0.7551137854223142, 0.7623534553379951, 0.7694960575423752, 0.7765404071213559, 0.7834853191608392, 0.7903296087467265, 0.7970720909649198, 0.8037115809013209, 0.8102468936418314, 0.8166768442723531, 0.8230002478787879, 0.8292159195470372, 0.8353226743630031, 0.7834853191608392]]



Chebyshev polynomial orthogonal polynomial least square curve fitting:
A second example to orthogonal polynomial curve fitting will be Chebyshev polynomial curve fitting. Chebyshev polynomials defined at (-1, 1) region.General definition of Chebyshev polynomials

$T_n(x) = \cos(n.cos^{-1}(x)$ (10.2-13)

If this equation is written in open form

$T_0(x)=1$

$T_1(x)=x$

$T_2(x)=2x^2-x$ (10.2-13a)

…..

$T_{n+1}(x)=2x\ T_n(x)- T_{n-1}(x)$

It is clear that it is a serial equation. The weight function for the Chebchev polynomials

$w(x) = \frac{1}{\sqrt{1-x^2}}$ (6.2-13)

The root of Chebchev polynomials

$t_k = \cos\left(\pi \frac{2k+1}{2n+2}\right)$ k=0,…,n (10.2-14)

n in this equation indicates the degree of polynomial. In order to curve fit data y=f(x) using this set as an orthogonal polynomial data should be given at the root points of chebchev polynomials.

Furthermore due to regional limit of the data, to apply this equation to real data a conversion process is required. If the real data is in region (a,b) The convertion equations will be

$x_k = \left(\frac{b-a}{2}\right) t_k + \left(\frac{a+b}{2}\right)$ (10.2-15)

$t_k = \left(\frac{x_k - a}{b-1}\right) - 1$ (10.2-156

If x_k and corresponding y_k ,is given

Coefficient of polynomial $P_n(x) = \sum_{k=0}^{n} C_k T_k(x)$ can be calculated by using least sqaure method. Due to orthogonality of the polynomials, matrix solution is not required, coefficients is found directly.

$C_0 = \frac{1}{n+1} \sum_{k=0}^{n} y_k T_0\,(t_k)$

$C_j = \frac{2}{n+1} \sum_{k=0}^{n} y_k T_j\,(t_k) \;\; j = 1,2,\dots,n$ (10.2-17)

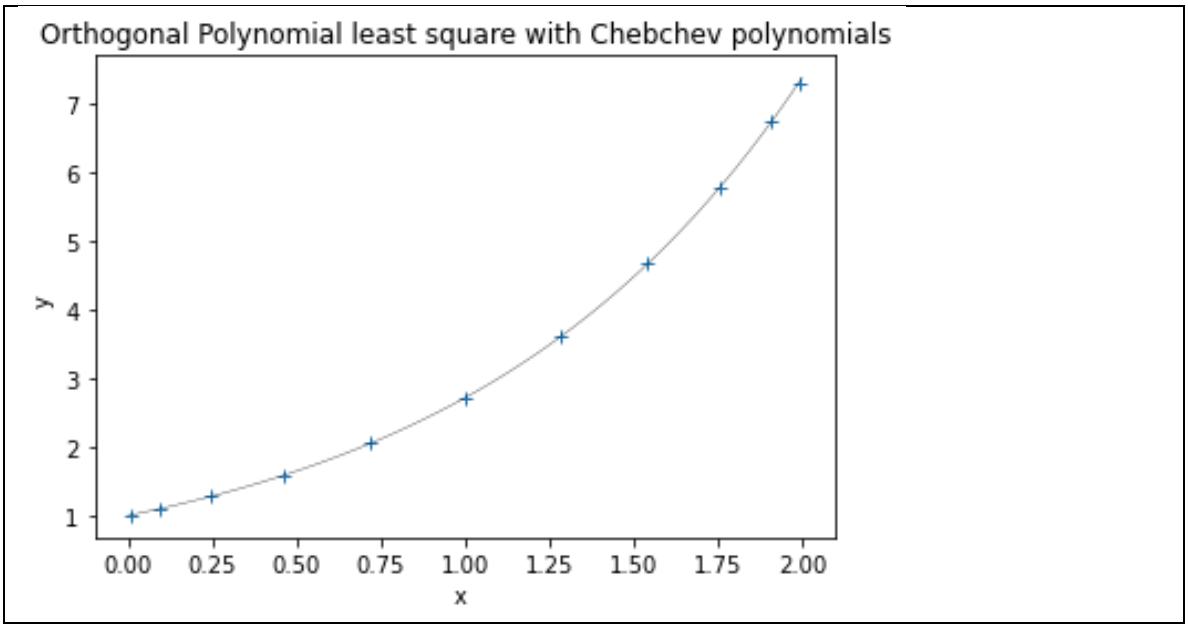
As an example problem f(x)=e^x function in the limit 0 to 2 will be investigated. If Chebchev polynomials are to be used in curve fitting, experiments should be design so that the data should be taken exactly in the roots of the Chebchev polynomials.

x_k	y_k
0.010178558	1.010231
0.090368005	1.094577
0.244250426	1.276664
0.459359183	1.583059
0.718267443	2.050877
1	2.718282
1.281732557	3.602877
1.540640817	4.66758
1.755749574	5.787784
1.909631995	6.750604
1.989821442	7.314228

```
#Checychev Orthogonal Polynomial Curve Fitting  chebchev_OPEKK.py
from math import *
import numpy as np;
import matplotlib.pyplot as plt

def Ti(x,equation_ref):
    #Chebysev function
    T=[0.0 for j in range(equation_ref+1)]
    T[0]=1.0
    if equation_ref>=1:
        T[1]=x
    for i in range(2,equation_ref+1):
        T[i]=2.0*x*T[i-1]-T[i-2]
    return T[equation_ref]

def xi(a,b,N):
    nn=N+1;
    x=[0.0 for j in range(nn)]
    t=[0.0 for j in range(nn)]
    for k in range(nn):
        t[k]=cos((2.0*N+1-2.0*k)*pi/(2.0*N+2.0))
        x[k]=(b-a)/2.0*t[k]+(a+b)/2.0
    return x

def Chebyshev(a,b,y,N):
    #region a b
    nn=N+1;
    #int k,j;
    d=pi/(2.0*N+2)
    x=[0.0 for j in range(nn)]
    t=[0.0 for j in range(nn)]
    # -1<=x<=1 region convertion
    for k in range(nn):
        t[k]=cos((2.0*N+1-2.0*k)*pi/(2.0*N+2.0))
        x[k]=(b-a)/2.0*t[k]+(a+b)/2.0
    C=[0.0 for j in range(nn)]
    z=0.0
    for k in range(nn):
```

```
        z+=y[k]
    C[0]=z/(N+1)
    for j in range(1,N+1):
        z=0
        for k in range(N+1):
            z=z+y[k]*Ti(t[k],j)
        C[j]=2.0*z/(N+1)
    return C

def func(C,a,b,x):
    t=2.0*(x-a)/(b-a)-1;
    n=len(C)
    ff=0
    if n!=0.0:
        for i in range(n):
            ff=ff+C[i]*Ti(t,i)
    return ff

def funcChebyshev(a,b,y,polinomkatsayisi,verisayisi):
    n=verisayisi+1
    z=[[0.0 for j in range(n)] for i in range(2)]
    C=Chebyshev(a,b,y,polinomkatsayisi)
    dx=(b-a)/float(verisayisi)
    z[0][0]=a;
    for i in range(verisayisi+1):
        if i==0:
            z[0][i]=a
        else:
            z[0][i]=z[0][i-1]+dx
        z[1][i]=func(C,a,b,z[0][i])
    return z

a=0
b=2.0
y=[1.010230536,1.094577019,1.276664,1.583059208,2.05087687,2.718281828,
        3.60287651,4.66758038,5.787784491,6.750604088,7.314227631]
#x values are roots of chebchev polynomials
x= xi(a,b,10)
z=funcChebyshev(a,b,y,10,50)
print("Chebyschev orthogonal least square curve fitting\n",z)
plt.title('Orthogonal Polynomial least square with Chebchev polynomials')
plt.xlabel('x ')
plt.ylabel('y ');
plt.plot(x,y,'+',z[0],z[1],'k',linewidth=0.3,)
```

```
runfile('E:/okul/SCO1/chebychev_OPEKK.py', wdir='E:/okul/SCO1')
Chebyschev orthogonal least square curve fitting
 [[0, 0.04, 0.08, 0.12, 0.16, 0.2, 0.24000000000000002, 0.28, 0.32, 0.36, 0.39999999999999997, 0.43999999999999995,
0.4799999999999999, 0.5199999999999999, 0.5599999999999999, 0.6, 0.64, 0.68, 0.7200000000000001, 0.7600000000000001,
0.8000000000000002, 0.8400000000000002, 0.8800000000000002, 0.9200000000000003, 0.9600000000000003,
1.0000000000000002, 1.0400000000000003, 1.0800000000000003, 1.1200000000000003, 1.1600000000000004,
1.2000000000000004, 1.2400000000000004, 1.2800000000000005, 1.3200000000000005, 1.3600000000000005,
1.4000000000000006, 1.4400000000000006, 1.4800000000000006, 1.5200000000000007, 1.5600000000000007,
1.6000000000000008, 1.6400000000000008, 1.6800000000000008, 1.7200000000000009, 1.760000000000001,
1.800000000000001, 1.840000000000001, 1.880000000000001, 1.920000000000001, 1.960000000000001, 2.000000000000001],
[1.0000000003363947, 1.0408107741052222, 1.0832870676174815, 1.1274968516551822, 1.1735108711582891,
1.2214027583411369, 1.2712491504601175, 1.323129812413029, 1.3771277643597053, 1.4333294145629911,
1.4918246976585994, 1.5527072185721034, 1.6160744023111264, 1.68202764987093, 1.7506725005018824, 1.822118800597927,
1.8964808794760928, 1.973877732328336, 2.054433210638661, 2.138276220370504, 2.2255409282418266, 2.316366976418341,
2.4108997059686983, 2.509290389439452, 2.6116964729221515, 2.718281828000004, 2.8292170139773067,
2.9446795508112373, 3.064854203182635, 3.1899332761602364, 3.3201169229313003, 3.455613465090903, 3.596639726002278,
3.7434213777615013, 3.8961933023216746, 4.055199967354418, 4.2206958174501725, 4.392945681283381,
4.5722251953942035, 4.758821245265077, 4.9530324243980175, 5.155169512127402, 5.365555970932761, 5.584528464047143,
5.812437394188804, 6.049647464277367, 6.296538261030195, 6.553504862370688, 6.820958469617304, 7.099327065460759,
7.389056098776642]]
```



10.3 **LINEAR CURVE FITTING : LEAST SQUARE METHOD WITH MULTI VARIABLES**

Curve fitting of a multivariable function basically the same of general curve fitting of the single variable function. The only basic difference is definition of the function to be used. Assume that

$$f(x_0, x_{1,}.., x_n) = \sum_{j=0}^{m} a_j^{(m)} \phi_j((x_0, x_{1,}.., x_n) \qquad (10.3-1)$$

j th degree function is given. It is desired to fit x_0, $x_{1,}$ x_2,…, x_n ,$f_{i,}$ i=0...n data set into this function. The best fitted $a_j^{(m)}$ values to be found. For this purpose Minimum value of

$$H(x_0, x_{1,}.., x_n) = \sum_{i=1}^{n} w(x_0, x_{1,}.., x_n) \left[f_i - \sum_{j=0}^{m} a_j^{(m)} \phi_j(x_0, x_{1,}.., x_n) \right]^2 \qquad (10.3-2)$$

function should be found. $w(x_0, x_{1,}.., x_n)$ is called weight function and it should satisfy the condition $w(x_0, x_{1,}.., x_n) \geq 0 \quad i = 1, \dots, n.$ The minimum of the function is the root of the derivative of the function.

$$\frac{\partial H(x_0, x_{1,}.., x_n)}{\partial a_k^{(m)}} = 2 \sum_{i=1}^{n} w(x_0, x_{1,}.., x_n) \left[f_i - \sum_{j=0}^{m} a_j^{(m)} \phi_j(x_0, x_{1,}.., x_n) \right] \phi_k(x_0, x_{1,}.., x_n) = 0 \quad k = 0, .., m \quad (10.3-3)$$

If unit value is taken as weight function, equation becomes:

$$2 \sum_{i=1}^{n} \left[f_i - \sum_{j=0}^{m} a_j^{(m)} \phi_j(x_0, x_{1,}.., x_n) \right] \phi_k(x_0, x_{1,}.., x_n) = 0 \qquad k = 0, .., m \quad (10.3-3a)$$

As an example a surface polinomial can be defined as

$f(x,y) = a_0 + a_1x + a_2y + a_3x^2 + a_4y^2 + a_5xy + a_6x^3 + a_7y^3 + a_8x^2y + a_9xy^2 + a_{10}x^4 + a_{11}y^4 + a_{12}x^2y^2 + a_{13}x^3y + a_{14}xy^2$

This polynomial is defined in the example problem

```
from abc import ABC, abstractmethod
from math import *

class f_xir(ABC):

    @abstractmethod
    def func(self,x,equation_ref):
        pass;
```

Data file a.txt

```
0	0	0
0	1	1
0	2	4
0	3	9
0	4	16
1	0	1
1	1	2
1	2	5
1	3	10
1	4	17
2	0	4
2	1	5
2	2	8
2	3	13
2	4	20
3	0	9
3	1	10
3	2	13
3	3	18
3	4	25
4	0	16
4	1	17
4	2	20
4	3	25
4	4	32
```

```
from f_xir import *
from gauss import *
class f1(f_xir):
#f(x,y)=a0+a1*x+a2*y+a3*x^2+a4*y^2+a5*xy+a6*x^3+a7*y^3+a8*x^2y
#      +a9*xy^2+a10*x^4+a11*y^4+a12*x^2y^2+a13x^3y+a14*xy^3
    def func(self,x,i):
        xx=0.0
        if i==0: xx=1.0
        elif i==1:   xx=x[0]
        elif i==2:   xx=x[1]
        elif i==3:   xx=x[0]*x[0]
        elif i==4:   xx=x[1]*x[1]
        elif i==5:   xx=x[0]*x[1]
        elif i==6:   xx=x[0]*x[0]*x[0]
        elif i==6:   xx=x[0]*x[0]*x[0]
        elif i==7:   xx=x[1]*x[1]*x[1]
        elif i==8:   xx=x[0]*x[0]*x[1]
        elif i==9:   xx=x[0]*x[1]*x[1]
        elif i==10:   xx=x[0]*x[0]*x[0]*x[0]
        elif i==11:   xx=x[1]*x[1]*x[1]*x[1]
        elif i==12:   xx=x[0]*x[0]*x[1]*x[1]
        elif i==13:   xx=x[0]*x[0]*x[0]*x[1]
        elif i==14:   xx=x[0]*x[1]*x[1]*x[1]
        return xx

# General least square curve fitting with multivariable
def Transpose(left):
    # transpose matrix (if A=a(i,j) Transpose(A)=a(j,i)
    # int i,j;
    n=len(left)
    m=len(left[0])
    b=[[0 for j in range(n)] for i in range(m)]
    for i in range(n):
        for j in range(m):
            b[j][i]=left[i][j]
    return b

def GeneralLeastSquare(c,n):
    n1=len(c)
    n2=len(c[0])-1
    xi=[[0 for j in range(n2)] for i in range(n1)]
    yi=[0 for j in range(n1)]
    for i in range(n1):
        for j in range(n2):
            xi[i][j]=c[i][j]
        yi[i]=c[i][n2]
```

```
    return GLS(xi,yi,n)

def GLS(xi,yi,n):
# n dimensional surface general least square curve fitting
    f=f1()
    l=len(xi)
    np1=n+1
    A=[[0 for j in range(np1)] for i in range(np1)]
    B=[0 for j in range(np1)]
    X=[0 for j in range(np1)]
    for i in range(n+1):
        B[i]=0
        for j in range(np1):
            A[i][j]=0.0
            for k in range(l):
                A[i][j]=A[i][j]+f.func(xi[k],i)*f.func(xi[k],j)
        for k in range(l):
                B[i]=B[i]+f.func(xi[k],i)*yi[k]
    X=gauss(A,B)
    max=0
    for i in range(n+1):
        if abs(X[i]) > max: max=abs(X[i])
    for i in range(n+1):
        if abs(X[i])/max > 0 and abs(X[i]/max) < 1.0e-100: X[i]=0.0
    #print(X)
    return X

def func(e,x):
    #multidimensional function calculation
    f=f1()
    n=len(e)
    ff=0.0
    if n!=0:
        for i in range(n-1,-1,-1):
            ff=ff+e[i]*f.func(x,i)
    return ff

def funcGeneralLeastSquare(e,xi):
    # multidimensional function calcu0000000000 nnnmlation
    f=f1()
    n=len(e)
    m=len(xi)
    ff=[0 for j in range(m)]
    for k in range(m):
       if n!=0:
           for i in range(n-1,-1,-1):
            ff[k]=ff[k]+e[i]*f.func(xi[k],i)
    return ff

def read(name):
    path = name
    file1 = open(path,'r')
    file1.seek(0,2)  #jumps to the end
    endLocation=file1.tell()
    file1.seek(0) #jumps to the start
    nn=0
    while True:
        line=file1.readline()
        nn+=1
        if  file1.tell()==endLocation:
            break
    x=[0.0 for i in range(nn)]
    y=[0.0 for i in range(nn)]
    z=[0.0 for i in range(nn)]
    file1.seek(0) #jumps to the start
    for i in range(nn):
        line=file1.readline()
        ee=line.split()
        x[i]=float(ee[0])
        y[i]=float(ee[1])
        z[i]=float(ee[2])
        #print(x[i]," ",y[i]," ",z[i])
    file1.close()
    a=[x,y,z]
    return Transpose(a)
def output1(c,polinomkatsayisi,aradegersayisi):
```

```
    n1=len(c)
    n2=len(c[0])-1
    xi=[[0 for j in range(n2)] for i in range(n1)]
    yi=[0 for j in range(n1)]
    for i in range(n1):
        for j in range(n2):
            xi[i][j]=c[i][j]
        yi[i]=c[i][n2]
    return output2(xi,yi,polinomkatsayisi,aradegersayisi);

def output2(xi,yi,polinomkatsayisi,aradegersayisi):
    n=len(xi)
    nk=len(xi[0])
    nn=(n-1)*(aradegersayisi+1)+1
    E=GLS(xi,yi,polinomkatsayisi)
    x=[[0 for j in range(nk)] for i in range(nn)]
    c=[[0 for j in range(nk+1)] for i in range(nn)]
    yy=[0.0 for i in range(nn)]
    dx=[0.0 for i in range(nk)]
    k=0
    # int i,j,w;
    for i in range(n-1):
        for w in range(nk):
            x[k][w]=xi[i][w]
            dx[w]=(xi[i+1][w]-xi[i][w])/(float(aradegersayisi)+1.0)
        k=k+1
        for j in range(aradegersayisi):
            for w in range(nk):
                x[k][w]=x[k-1][w]+dx[w]
            k=k+1
    for w in range(nk):
        x[k][w]=xi[i][w]
    yy=funcGeneralLeastSquare(E,x)
    for i in range(len(x)):
        for w in range(nk):
            c[i][w]=x[i][w]
        c[i][nk]=yy[i]
    return c

a=read('a.txt')
e=GeneralLeastSquare(a,9)
#print(e)
y=output1(a,9,1)
print(y)
```

```
runfile('E:/okul/SCO1/multidimensional_LSCF.py', wdir='E:/okul/SCO1')
Reloaded modules: f_xir, gauss
[[0.0, 0.0, -2.1827872842550278e-14], [0.0, 0.5, 0.24999999999998757], [0.0, 1.0, 0.9999999999999939], [0.0, 1.5,
2.249999999999998], [0.0, 2.0, 4.0], [0.0, 2.5, 6.25], [0.0, 3.0, 9.0], [0.0, 3.5, 12.249999999999998], [0.0, 4.0, 15.999999999999996],
[0.5, 2.0, 4.249999999999998], [1.0, 0.0, 0.9999999999999859], [1.0, 0.5, 1.249999999999992], [1.0, 1.0, 1.999999999999996],
[1.0, 1.5, 3.249999999999998], [1.0, 2.0, 4.999999999999998], [1.0, 2.5, 7.249999999999997], [1.0, 3.0, 9.999999999999998], [1.0,
3.5, 13.249999999999996], [1.0, 4.0, 16.999999999999996], [1.5, 2.0, 6.249999999999999], [2.0, 0.0, 3.999999999999995], [2.0,
0.5, 4.249999999999998], [2.0, 1.0, 5.0], [2.0, 1.5, 6.25], [2.0, 2.0, 8.000000000000002], [2.0, 2.5, 10.25], [2.0, 3.0,
12.999999999999998], [2.0, 3.5, 16.25], [2.0, 4.0, 20.0], [2.5, 2.0, 10.250000000000002], [3.0, 0.0, 9.000000000000002], [3.0, 0.5,
9.250000000000004], [3.0, 1.0, 10.000000000000004], [3.0, 1.5, 11.250000000000004], [3.0, 2.0, 13.000000000000002], [3.0, 2.5,
15.25], [3.0, 3.0, 18.0], [3.0, 3.5, 21.25], [3.0, 4.0, 25.000000000000004], [3.5, 2.0, 16.249999999999996], [4.0, 0.0, 16.0], [4.0, 0.5,
16.25], [4.0, 1.0, 16.999999999999996], [4.0, 1.5, 18.249999999999996], [4.0, 2.0, 19.999999999999993], [4.0, 2.5,
22.249999999999993], [4.0, 3.0, 24.999999999999996], [4.0, 3.5, 28.25], [4.0, 3.0, 24.999999999999996]]
```

A general polinomial formula for 3d surface curve fitting with python language

In one dimesional curve fitting it is a common practice to use a general polynomial formula

$$y = f(x) = \sum_{i=0}^{n} a_i x^i$$

By assuming this general form of polynomial function we can avoid to define function to be curve fitted. Similar approach can be used for 3D surface curve fitting by defining

$$z = f(x,y) = \sum_{i=0}^{m} \sum_{j=0}^{n} a_{i(m+1)+j}\, x^i y^j$$

Degree of polynomial

$$k = i(m+1) + j$$

When k is given, i and j can be calculated as:

$$i = (integer)\frac{k}{(m+1)}$$
$$j = k - (m+1)i$$

Derivatives of function can also be calculated directly from the formulation

$$\frac{dz}{dx} = \frac{df(x,y)}{dx} = \sum_{i=1}^{m}\sum_{j=0}^{n} a_{i(m+1)+j}\, i\, x^{i-1}y^{j}$$
$$\frac{dz}{dy} = \frac{df(x,y)}{dy} = \sum_{i=0}^{m}\sum_{j=1}^{n} a_{i(m+1)+j}\, j\, x^{i}y^{j-1}$$
$$\frac{d^2z}{dx^2} = \frac{d^2f(x,y)}{dx^2} = \sum_{i=2}^{m}\sum_{j=0}^{n} a_{i(m+1)+j}\, i\,(i-1)x^{i-2}y^{j}$$
$$\frac{d^2z}{dy^2} = \frac{d^2f(x,y)}{dy^2} = \sum_{i=0}^{m}\sum_{j=2}^{n} a_{i(m+1)+j}\, j\,(j-1)x^{i}y^{j-2}$$
$$\frac{d^2z}{dxdy} = \frac{d^2f(x,y)}{dxdz} = \sum_{i=1}^{m}\sum_{j=1}^{n} a_{i(m+1)+j}\, ij\, x^{i-1}y^{j-1}$$

This general form of function can be utilised in several applications, curve fitting, soltion of differential eqautions etc. We will apply here to curve fitting. Minimisation function for curve fitting:

For this purpose Minimum value of

$$H(x,y) = \sum_{w=0}^{nw} w(x,y)\left[z_w - \sum_{i=0}^{m}\sum_{j=0}^{n} a_{i(m+1)+j}x_w^i y_w^j\right]^2$$
$$\frac{\partial H(x,y)}{\partial a_k^{(m)}} = 2\sum_{w=0}^{nw} w(x,y)\left[z_w - \sum_{i=0}^{m}\sum_{j=0}^{n} a_{i(m+1)+j}x_w^i y_w^j\right]\left(\sum_{i=0}^{m}\sum_{j=0}^{n} a_{i(m+1)+j}x_w^i y_w^j\right) = 0$$
$$k = 0,..,(m+1)(n+1) \quad (6.3-2)$$

Python language code for general function class if_poly2:

```
 # -*- coding: utf-8 -*-
"""
Created on Sun Dec 19 06:19:18 2021

@author: M. Turhan ÇOBAN
"""
from f_xj import *
class if_poly2(f_xj):
    def __init__(self, ai,ni,mi):
        self.a=ai
        self.n=ni
        self.m=mi

    def seta(self, ai,ni,mi):
        self.a=ai
        self.n=ni
        self.m=mi

    def kk(self, k):
        i=int(k/(self.m+1))
        j=int(k-i*(self.m+1))
        k1=[i,j]
        return k1

    def func(self,z):
        k=self.n*self.m
        x=z[0]
        y=z[1]
        k1=0
        xx=0.0
        total=0.0
```

```
        for i in range(0, self.n+1):
            for j in range(0, self.m+1):
                k1=i*(self.m+1)+j
                xx=self.pow(x,y,i,j)
                total=total+self.a[k1]*xx
        return total

    def dfuncx(self,z):
        k=self.n*self.m
        x=z[0]
        y=z[1]
        k1=0
        xx=0.0
        total=0.0
        for i in range(1, self.n+1):
            for j in range(0, self.m+1):
                k1=i*(self.m+1)+j
                xx=self.dpowx(x,y,i,j)
                total=total+self.a[k1]*xx
        return total

    def dfuncy(self,z):
        k=self.n*self.m
        x=z[0]
        y=z[1]
        k1=0
        xx=0.0
        total=0.0
        for i in range(0, self.n+1):
            for j in range(1, self.m+1):
                k1=i*(self.m+1)+j
                xx=self.dpowy(x,y,i,j)
                total=total+self.a[k1]*xx
        return total

    def d2funcx2(self,z):
        k=self.n*self.m
        x=z[0]
        y=z[1]
        k1=0
        xx=0.0
        total=0.0
        for i in range(2, self.n+1):
            for j in range(0, self.m+1):
                k1=i*(self.m+1)+j
                xx=self.d2powx2(x,y,i,j)
                total=total+self.a[k1]*xx
        return total

    def d2funcy2(self,z):
        k=self.n*self.m
        x=z[0]
        y=z[1]
        k1=0
        xx=0.0
        total=0.0
        for i in range(0, self.n+1):
            for j in range(2, self.m+1):
                k1=i*(self.m+1)+j
                xx=self.d2powy2(x,y,i,j)
                total=total+self.a[k1]*xx
        return total

    def d2funcxy(self,z):
        k=self.n*self.m
        x=z[0]
        y=z[1]
        k1=0
        xx=0.0
        total=0.0
        for i in range(1, self.n+1):
            for j in range(1, self.m+1):
                k1=i*(self.m+1)+j
                xx=self.d2powxy(x,y,i,j)
                total=total+self.a[k1]*xx
        return total
```

```
    def pow(self,x,y,n1,m1):
        pow1=1.0
        if n1==0 and m1==0:
            return pow1
        elif m1==0 and n1!=0:
            for i in range(n1):
                pow1=pow1*x
        elif n1==0 and m1!=0:
            for i in range(m1):
                pow1=pow1*y
        else:
            for i in range(n1):
                pow1=pow1*x
            for j in range(m1):
                pow1=pow1*y
        return pow1

    def dpowx(self,x,y,n1,m1):
        powx=n1
        powy=1.0
        for i in range(1,n1):
            powx=powx*x
            for j in range(m1):
                powy=powy*y
        return powx*powy

    def dpowy(self,x,y,n1,m1):
        powx=1.0
        powy=m1
        for i in range(n1):
            powx=powx*x
        for j in range(1,m1):
            powy=powy*y
        return powx*powy

    def d2powx2(self,x,y,n1,m1):
         powx=n1*(n1-1)
         powy=1.0
         for i in range(2,n1):
             powx=powx*x
         for j in range(m1):
             powy=powy*y
         return powx*powy

    def d2powy2(self,x,y,n1,m1):
         powx=1.0
         powy=m1*(m1-1)
         for i in range(n1):
             powx=powx*x
         for j in range(2,m1):
             powy=powy*y
         return powx*powy

    def d2powxy(self,x,y,n1,m1):
         powx=n1
         powy=m1
         for i in range(1,n1):
             powx=powx*x
         for j in range(1,m1):
             powy=powy*y
         return powx*powy
"""
a=[1,1,1,1,1,1,1,1,1]
z=[0.5,0.5]
n=2
m=2
f=if_poly2(a,n,m)
w=f.func(z)
print("w=",w)
w1=f.dfuncx(z);
print("dwx=",w1)
w2=f.dfuncy(z);
print("dwy=",w2)
"""
```

A curve fitting program in python:

```
 # -*- coding: utf-8 -*-
"""
Created on Sun Dec 19 06:19:18 2021

@author: M. Turhan ÇOBAN
"""
# Import libraries
from mpl_toolkits import mplot3d
import numpy as np
import matplotlib.pyplot as plt
from if_poly2 import *
class general_LSTSQR:
    #constructor method
    def __init__(self, ci,ni,mi):
        c=ci
        self.n=ni
        self.m=mi
        np1=(self.n+1)*(self.m+1);
        self.e=[0.0 for z in range(np1)]
        self.f=if_poly2(self.e,self.n,self.m)
        n1=len(c)
        n2=len(c[0])-1
        xi=[[0.0 for x in range(n2)] for y in range(n1)];
        yi=[0.0 for z in range(n1)]
        for i in range(0, n1):
            for j in range(0, n2):
                xi[i][j]=c[i][j]
            yi[i]=c[i][n2]
        xi=Transpose(xi)
        l=len(yi)
        A=[[0.0 for x in range(np1)] for y in range(np1)]
        B=[0.0 for z in range(np1)]
        for i in range(0, np1):
            B[i]=0.0
            ij1=self.f.kk(i)
            i1=ij1[0]
            j1=ij1[1]
            for j in range(np1):
                A[i][j]=0.0
                ij2=self.f.kk(j)
                i2=ij2[0]
                j2=ij2[1]
                for k in range(l):
                    fi1=self.f.pow(xi[0][k],xi[1][k],i1,j1)
                    fi2=self.f.pow(xi[0][k],xi[1][k],i2,j2)
                    A[i][j]=A[i][j]+fi1*fi2
            for k in range(l):
                B[i]=B[i]+self.f.pow(xi[0][k],xi[1][k],i1,j1)*yi[k]
        self.e=gauss(A,B)
        self.f=if_poly2(self.e,self.n,self.m)
        self.xi=xi
        self.yi=yi

    #function value at point (x[0],x[1])
    # z=f(x,y)
    def func(self,x):
        return self.f.func(x)

    #output method
    def output(self,x0,xn,dx,y0,yn,dy):
        n=int((xn-x0)/dx+0.00001)+1
        m=int((yn-y0)/dy+0.00001)+1
        mm=n*m
        a=mesh(x0,xn,dx,y0,yn,dy)
        n=len(a[0])
        b=[[0.0 for x in range(mm)] for y in range(3)]
        for i in range(n):
            b[0][i]=a[0][i]
            b[1][i]=a[1][i]
            xx=[a[0][i],a[1][i]]
            b[2][i]=a[1][i]=self.func(xx)
        return b
```

```
    #plotting curve fit data
    def plot(self,xmin,xmax,Nx,ymin,ymax,Ny,PLabel,XLabel,YLabel,ZLabel):
        #f=f2()
        x=[0.0 for j in range(Nx)]
        y=[0.0 for j in range(Ny)]
        z=[[0.0 for j in range(Ny)] for i in range(Nx)]
        dx=(xmax-xmin)/(Nx-1)
        dy=(ymax-ymin)/(Ny-1)
        for i in range(Nx):
            for j in range(Ny):
                x[i]=xmin+dx*i
                y[j] = ymin+dy*j
                xx=[x[i],y[j]]
                z[i][j]=self.func(xx)
        xi=np.array(x)
        yi=np.array(y)
        xi, yi = np.meshgrid(xi, yi)
        zi=np.array(z)
        fig = plt.figure(figsize =(14, 9))
        ax = plt.axes(projection ='3d')
        # Creating color map
        my_cmap = plt.get_cmap('terrain')
        # Creating plot
        surf = ax.plot_surface(xi, yi, zi,cmap = my_cmap,edgecolor ='none')
        fig.colorbar(surf, ax = ax,shrink = 0.5,aspect = 5)
        ax.set_title(PLabel)
        ax.set_xlabel(XLabel)
        ax.set_xlim(xmin, xmax)
        ax.set_ylabel(YLabel)
        ax.set_xlim(ymin, ymax)
        ax.set_zlabel(ZLabel)
        ax.set_zlim(np.min(zi), np.max(zi))
        plt.show()


#create data from a given function
def create_data(f,x0,xn,dx,y0,yn,dy):
    n=int((xn-x0)/dx+0.00001)+1
    m=int((yn-y0)/dy+0.00001)+1
    mm=n*m
    a=mesh(x0,xn,dx,y0,yn,dy)
    n=len(a[0])
    print("create data n=",n)
    b=[[0.0 for x in range(mm)] for y in range(3)]
    for i in range(n):
        b[0][i]=a[0][i]
        b[1][i]=a[1][i]
        xx=[a[0][i],a[1][i]]
        b[2][i]=a[1][i]=f.func(xx)
    b=Transpose(b)
    return b

#mesh : creates a mesh a=[x,y] x=x0:dx.xn y=y0:dy:yn
def mesh(x0,xn,dx,y0,yn,dy):
    n=int((xn-x0)/dx+0.00001)+1
    m=int((yn-y0)/dy+0.00001)+1
    mm=n*m
    x=[0.0 for z in range(mm)]
    y=[0.0 for z in range(mm)]
    a=[[0.0 for x in range(mm)] for y in range(2)]
    for i in range(n):
        x[i]=x0+dx*i
        for j in range(m):
            y[j]=y0+dy*j
            a[0][i*n+j]=x[i]
            a[1][i*n+j]=y[j]
    return a

def gauss(a,b):
    #Gauss elimination with pivoting
  b1=b
  n=len(a)
  x=[0 for i in range(n)]
  for k in range(n):
```

```
    p=k
    buyuk=abs(a[k][k])
    for ii in range(k+1,n):
      dummy=abs(a[ii][k])
      if dummy>buyuk:
         buyuk=dummy
         p=ii
    if p!=k:
      for jj in range(k,n):
        dummy=a[p][jj]
        a[p][jj]=a[k][jj]
        a[k][jj]=dummy
      dummy=b1[p]
      b1[p]=b1[k]
      b1[k]=dummy
    for i in range(k+1,n):
       carpan=a[i][k]/a[k][k]
       a[i][k]=0;
       for j in range(k+1,n):
          a[i][j]=a[i][j]-carpan*a[k][j]
       b1[i]   =b1[i]-carpan*b[k]
  x[n-1]=b1[n-1]/a[n-1][n-1]
  for i in range(n-1,-1,-1):
    toplam=0;
    for j in range(i+1,n):
      toplam=toplam+a[i][j]*x[j]
    x[i]=(b[i]-toplam)/a[i][i];
  return x;

def Transpose(left):
    n1=len(left)
    m1=len(left[0])
    b=[[0.0 for x in range(n1)] for y in range(m1)]
    for i in range(n1):
        for j in range(m1):
            b[j][i]=left[i][j]
    return b

def read(name):
    path = name
    file1 = open(path,'r')
    file1.seek(0,2)  #jumps to the end
    endLocation=file1.tell()
    file1.seek(0) #jumps to the start
    nn=0
    while True:
        line=file1.readline()
        nn+=1
        if  file1.tell()==endLocation:
            break
    x=[0.0 for i in range(nn)]
    y=[0.0 for i in range(nn)]
    z=[0.0 for i in range(nn)]
    file1.seek(0) #jumps to the start
    for i in range(nn):
        line=file1.readline()
        ee=line.split()
        x[i]=float(ee[0])
        y[i]=float(ee[1])
        z[i]=float(ee[2])
        #print(x[i]," ",y[i]," ",z[i])
    file1.close()
    a=[x,y,z]
    return Transpose(a)

"""
a=read("d.txt")
g=general_LSTSQR(a,2,2)
print("least square coefficients e = \n",g.e)
x=[0.5,0.5]
print("y=",g.func(x))
x0=-4.0
xn=4.0
dx=0.1
y0=-4.0
yn=4.0
```

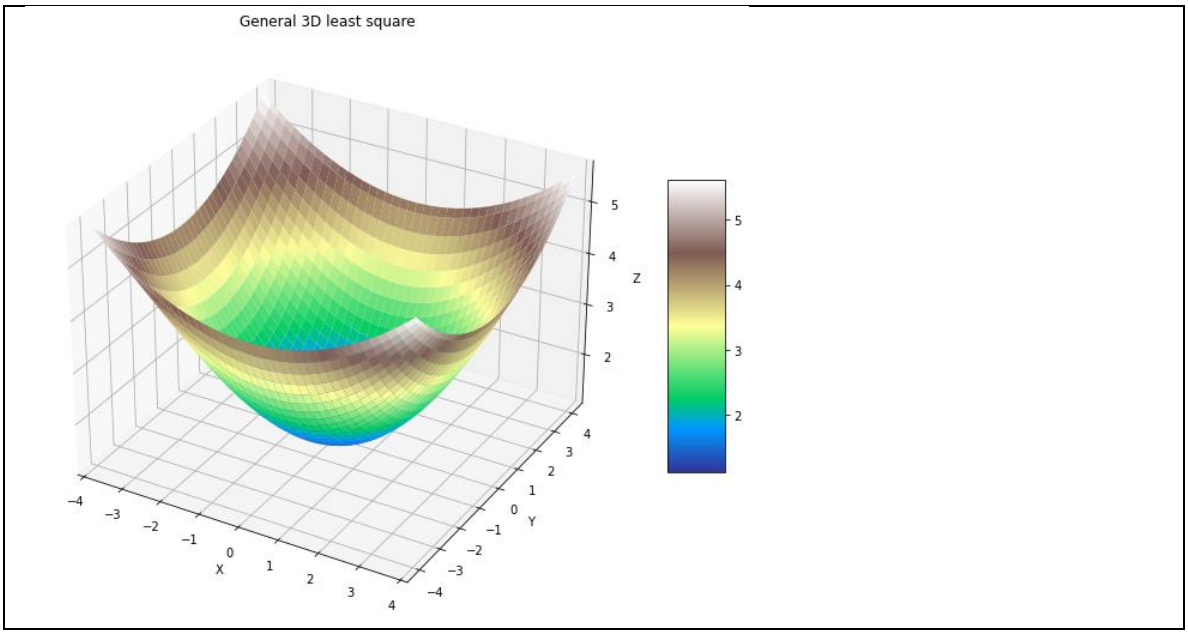
```
dy=0.1
Nx=int((xn-x0)/dx)
Ny=int((yn-y0)/dy)
Plabel="General 3D least square"
Xlabel="X"
Ylabel="Y"
Zlabel="Z"
c=g.output(x0,xn,dx,y0,yn,dy)
#print("c = \n",c)
g.plot(x0,xn,Nx,y0,yn,Ny,Plabel,Xlabel,Ylabel,Zlabel)
"""
```

Sample data: c.txt

0	0	1
0.1	0	1.11
0.2	0	1.24
0.3	0	1.39
0.4	0	1.56
0.5	0	1.75
0.6	0	1.96
0.7	0	2.19
0.8	0	2.44
0.9	0	2.71
1	0	3
0	0.1	1.11
0.1	0.1	1.2321
0.2	0.1	1.3764
0.3	0.1	1.5429
0.4	0.1	1.7316
0.5	0.1	1.9425
0.6	0.1	2.1756
0.7	0.1	2.4309
0.8	0.1	2.7084
0.9	0.1	3.0081
1	0.1	3.33
0	0.2	1.24
0.1	0.2	1.3764
0.2	0.2	1.5376
0.3	0.2	1.7236
0.4	0.2	1.9344
0.5	0.2	2.17
0.6	0.2	2.4304
0.7	0.2	2.7156
0.8	0.2	3.0256
0.9	0.2	3.3604
1	0.2	3.72
0	0.3	1.39
0.1	0.3	1.5429
0.2	0.3	1.7236
0.3	0.3	1.9321
0.4	0.3	2.1684
0.5	0.3	2.4325
0.6	0.3	2.7244
0.7	0.3	3.0441
0.8	0.3	3.3916
0.9	0.3	3.7669
1	0.3	4.17
0	0.4	1.56
0.1	0.4	1.7316
0.2	0.4	1.9344
0.3	0.4	2.1684
0.4	0.4	2.4336
0.5	0.4	2.73
0.6	0.4	3.0576
0.7	0.4	3.4164
0.8	0.4	3.8064
0.9	0.4	4.2276
1	0.4	4.68
0	0.5	1.75
0.1	0.5	1.9425
0.2	0.5	2.17
0.3	0.5	2.4325
0.4	0.5	2.73
0.5	0.5	3.0625
0.6	0.5	3.43
0.7	0.5	3.8325

0.8	0.5	4.27
0.9	0.5	4.7425
1	0.5	5.25
0	0.6	1.96
0.1	0.6	2.1756
0.2	0.6	2.4304
0.3	0.6	2.7244
0.4	0.6	3.0576
0.5	0.6	3.43
0.6	0.6	3.8416
0.7	0.6	4.2924
0.8	0.6	4.7824
0.9	0.6	5.3116
1	0.6	5.88
0	0.7	2.19
0.1	0.7	2.4309
0.2	0.7	2.7156
0.3	0.7	3.0441
0.4	0.7	3.4164
0.5	0.7	3.8325
0.6	0.7	4.2924
0.7	0.7	4.7961
0.8	0.7	5.3436
0.9	0.7	5.9349
1	0.7	6.57
0	0.8	2.44
0.1	0.8	2.7084
0.2	0.8	3.0256
0.3	0.8	3.3916
0.4	0.8	3.8064
0.5	0.8	4.27
0.6	0.8	4.7824
0.7	0.8	5.3436
0.8	0.8	5.9536
0.9	0.8	6.6124
1	0.8	7.32
0	0.9	2.71
0.1	0.9	3.0081
0.2	0.9	3.3604
0.3	0.9	3.7669
0.4	0.9	4.2276
0.5	0.9	4.7425
0.6	0.9	5.3116
0.7	0.9	5.9349
0.8	0.9	6.6124
0.9	0.9	7.3441
1	0.9	8.13
0	1	3
0.1	1	3.33
0.2	1	3.72
0.3	1	4.17
0.4	1	4.68
0.5	1	5.25
0.6	1	5.88
0.7	1	6.57
0.8	1	7.32
0.9	1	8.13
1	1	9

```
runfile('D:/okul/SCO1/general_LSTSQR.py', wdir='D:/okul/SCO1')
Reloaded modules: if_poly2, f_xj
least square coefficients e =
 [1.1067152847257073, 1.9156629959693034e-16, 0.20177161497449503, -
6.816136390829964e-16, 7.372344618314836e-18, 7.82107690022991e-17,
0.20177161497449347, -1.737608496219506e-17, -0.007341656396243165]
y= 1.207142238688189
n= 6561
```

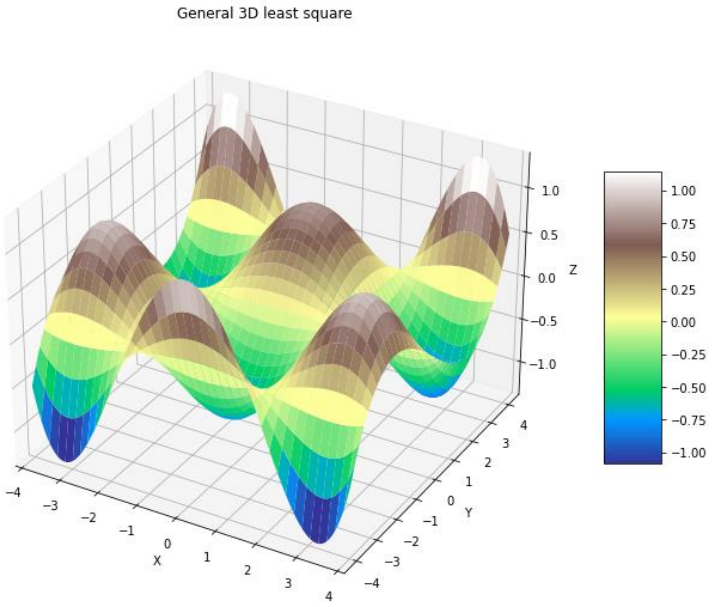


Data for curvefitting can be created by using a real function, if it done this way, it is called function fitting (creating a new function to replace original function plus some error)

```
# -*- coding: utf-8 -*-
"""
Created on Fri Aug 24 16:46:48 2018

@author: Mustafa Turhan Çoban
"""
from math import *
from f_xj import *
from general_LSTSQR import *

class f2(f_xj):
  def func(self,x):
      y=sin(x[0])*cos(x[1]);
      return y;

f=f2();
x0=-4.0
xn=4.0
dx=0.05
y0=-4.0
yn=4.0
dy=0.05
a=create_data(f,x0,xn,dx,y0,yn,dy)
#a=read("d.txt")
g=general_LSTSQR(a,4,4)
print("least square coefficients e = \n",g.e)
x=[0.5,0.5]
print("y=",g.func(x))

Nx=int((xn-x0)/dx)
Ny=int((yn-y0)/dy)
Plabel="General 3D least square"
Xlabel="X"
Ylabel="Y"
Zlabel="Z"
c=g.output(x0,xn,dx,y0,yn,dy)
#print("c = \n",c)
g.plot(x0,xn,Nx,y0,yn,Ny,Plabel,Xlabel,Ylabel,Zlabel)
```

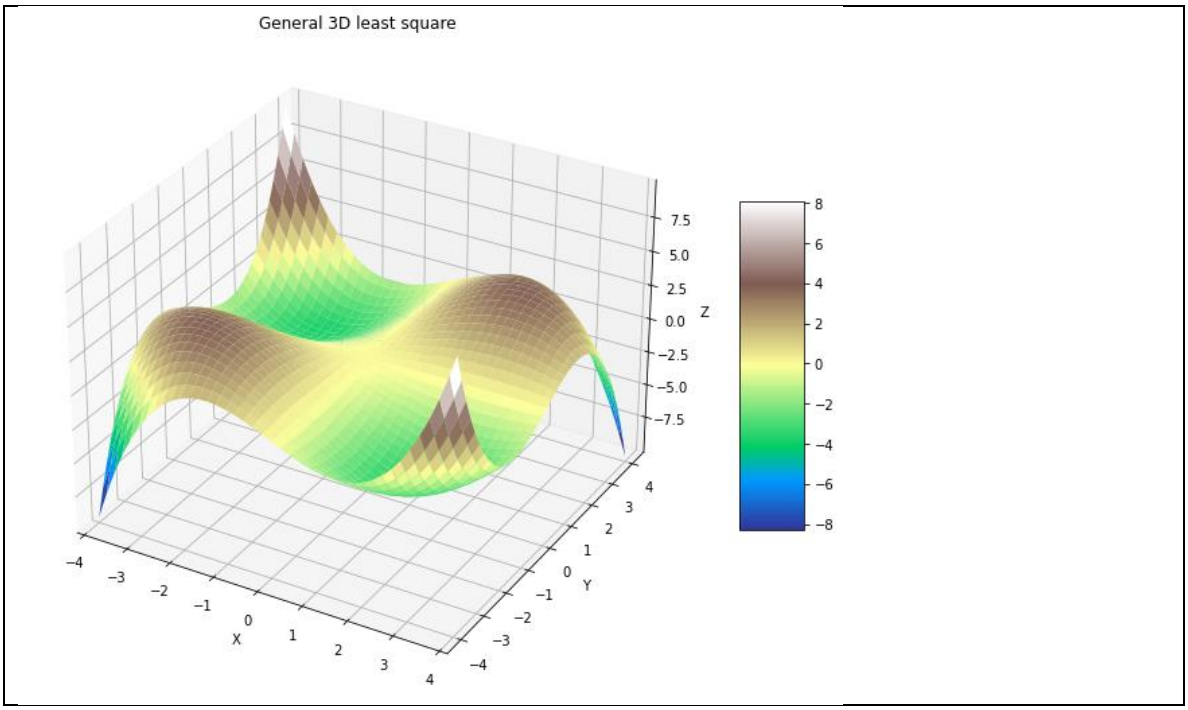
```
least square coefficients e =
 [1.4801238114907834e-13, -2.650893627865041e-15, -7.3757470555987e-14, 2.0916175758491894e-16, 4.970214965436211e-15, 0.6288990092767766, -1.927681086258701e-15, -0.2670171187946007, 1.7890733517997218e-16, 0.012990432369600872, -9.840903711405612e-14, 1.2086933777773007e-15, 5.030145442454727e-14, -9.343845820017479e-17, -3.4154019365970342e-15, -0.056941470156177995, 1.7892548710336614e-16, 0.02417613492271091, -1.6546242212861204e-17, -0.0011761734494387336, 7.228281874941274e-15, -7.925728790310766e-17, -3.724816162405148e-15, 6.106063004041828e-18, 2.5351145751318363e-16]
```

```
y= 0.27510694739245906
n= 25921
```




```
# -*- coding: utf-8 -*-
"""
Created on Fri Aug 24 16:46:48 2018

@author: Mustafa Turhan Çoban
"""
from math import *
from f_xj import *
from general_LSTSQR import *

class f1(f_xj):func=lambda self,x: sin(x[0])*x[1]+sin(x[1])*x[0]
f=f1()
x0=-4.0
xn=4.0
dx=0.05
y0=-4.0
yn=4.0
dy=0.05
a=create_data(f,x0,xn,dx,y0,yn,dy)
#a=read("d.txt")
g=general_LSTSQR(a,4,4)
print("least square coefficients e = \n",g.e)
Nx=int((xn-x0)/dx)
Ny=int((yn-y0)/dy)
Plabel="General 3D least square"
Xlabel="X"
Ylabel="Y"
Zlabel="Z"
c=g.output(x0,xn,dx,y0,yn,dy)
#print("c = \n",c)
g.plot(x0,xn,Nx,y0,yn,Ny,Plabel,Xlabel,Ylabel,Zlabel)
```

```
runfile('D:/okul/SCO1/f_xjtest2.py', wdir='D:/okul/SCO1')
Reloaded modules: f_xj, general_LSTSQR, if_poly2
create data n= 25921
least square coefficients e =
 [-1.3836153386354733e-14, -3.7673643480520174e-14, 6.8378407527254534e-15, 3.4819724447542655e-15, -
4.48339203167637e-16, -4.887138402483711e-15, 1.3661993775470942, 2.6424210331665854e-15, -0.061848881884457856, -
1.7586697755381467e-16, 6.2194758858130085e-15, 2.171630454430557e-14, -3.035666243961801e-15, -1.9956360550410714e-
15, 1.9788626152472801e-16, 4.807100056984052e-16, -0.06184888188446137, -2.624176421227302e-16, 1.989730252107513e-
15, 1.7574277786403278e-17, -4.007219309910294e-16, -1.5485211697230224e-15, 1.9393069568430817e-16,
1.4242836949261273e-16, -1.2598563213076173e-17]
```



10.4 LEAST SQUARE CURVE FITTING OF NONLINEAR EQUATIONS

If curve fitting to a function $y = f(a; x)$ with non-linear coefficients is required. Process will be a little bit different compare to equations with linear coefficients.
$y = f(a; x)$ is given and data x_i,f_i, i=0...n will be fit into the equation. In this equation coefficients are placed in a non-linear form. For example : $y = a_0(1 - e^{-a_1 x})$
Least square error formula was defined as

$$H\left(a_j^{(m)}; x_0, x_{1,}.., x_n\right) = \sum_{i=1}^{n} w\left(x_0, x_{1,}.., x_n\right)\left[y_i - f\left(a_j^{(m)}; x_0, x_{1,}.., x_n\right)\right]^2 \quad (10.4 - 1)$$

This equation should be solved by using one of the multidimensional optimization methods such as Nelder-Mead, Newton-Raphson, Steepest descent, Generic algorithms
sample : (input3.txt)

```
5.0 16.0
10.0 25.0
15.0 32.0
20.0 33.0
25.0 38.0
30.0 36.0
```

A nonlinear function to fit the data:
pi=ai[0]*(1.0-exp(-ai[1]*x));
In this function ai are coefficients of the equations and x is independent variable. In order to solve non-linear system of equations Nelder-Mead optimisation method is used.

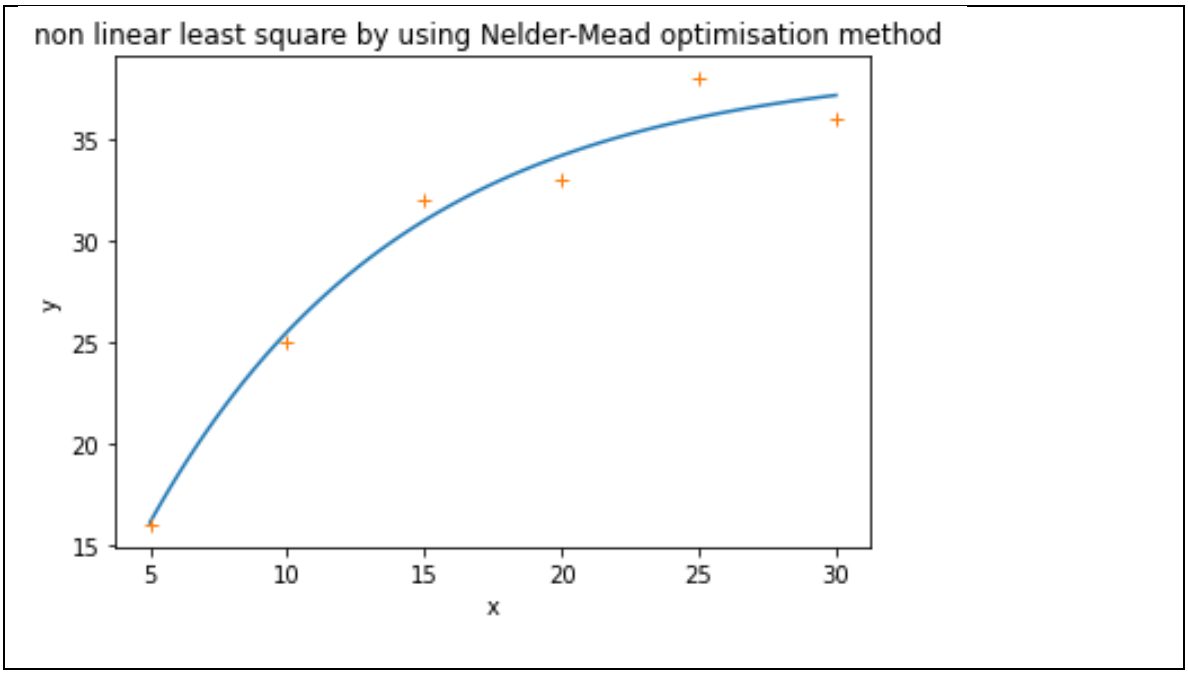
```
from math import *
from f_xj import *
from matplotlib import pyplot as plt
import numpy as np


class y(f_xj):
    def __init__(self,filename,ia):
        self.name = 'yy',
        self.filename=filename
        path = filename
        fin=open(path,'r')
        n=len(ia)
```

```
        self.a=ia
        path = self.filename
        file1 = open(path,'r')
        file1.seek(0,2)  #jumps to the end
        endLocation=file1.tell()
        file1.seek(0) #jumps to the start
        nn=0
        while True:
            line=file1.readline()
            nn+=1
            if  file1.tell()==endLocation:
                break
        xi=[0.0 for i in range(nn)]
        yi=[0.0 for i in range(nn)]
        file1.seek(0) #jumps to the start
        for i in range(nn):
            line=file1.readline()
            ee=line.split()
            xi[i]=float(ee[0])
            yi[i]=float(ee[1])
            #print(xi[i]," ",yi[i])
        file1.close()
        self.xi=xi
        self.yi=yi
        self.nn=nn

    def Ps(self,x,ai):
        pi=ai[0]*(1-exp(-ai[1]*x))
        return pi
    def func(self,ai):
        ff=0
        for i in range(self.nn):
            w=self.Ps(self.xi[i],ai)
            yy=self.yi[i]
            w=w-yy
            ff=ff+w*w
        return ff

def  nelder(fnelder,a,da,maxiteration,tolerance,printlist):
    #print("a",a)
    #print("da",da)
    n=len(a)
    m=n+1
    x=[[0.0 for j in range(n)] for i in range(n+1)]
    p=[[0.0 for j in range(m)] for i in range(m)]
    s=""
    NDIMS = len(x)-1
    NPTS = len(x)
    FUNC = NDIMS
    ncalls = 0;
    for i in range(m):
        for j in range(n):
            if i==j:
                x[i][j]=a[j]+da[j]
                p[i][j]=x[i][j]
            else:
                x[i][j]=a[j]
                p[i][j]=x[i][j]
        p[i][FUNC] = fnelder.func(p[i])
        #print("x[",i,"]=",x[i])
        #print("p[",i,"]=",p[i])
    #print("p",p)
    #Inlet variable definitions
    #fnelder : abstract multivariable function f(x)
    #x : independent variable set of n+1 simplex elements
    #maxiteration : maximum iteration number
    #tolerance :

    #///// construct the starting simplex ////////////////////
    z=[0.0 for i in range(NDIMS)]
    best = 1.0E99;
    #/////////////// calculate the first function values for the simplex ////////////////
    iter=0;
    for iter in range(1,maxiteration):
        #////////  define lo, nhi, hi (low high next_to_high //////////////
        ilo=0
```

```
ihi=0
inhi = -1
# -1 means missing
flo = p[0][FUNC];
fhi = flo
#print("flo=",flo,"fhi=",fhi)
#double pavg,sterr;
for i in range(1,NPTS):
    if p[i][FUNC] < flo:
        flo=p[i][FUNC]
        ilo=i
    if  p[i][FUNC] > fhi:
        fhi=p[i][FUNC]
        ihi=i
    #print("i=",i,"flo=",flo,"fhi=",fhi,"ilo=",ilo,"ihi=",ihi,"pifunc=",p[i][FUNC])
fnhi = flo
inhi = ilo
#print("fnhi=",fnhi,"inhi=",inhi)
for i in range(NPTS):
    if (i != ihi) and (p[i][FUNC] > fnhi):
        fnhi=p[i][FUNC]
        inhi=i
#///////// exit criteria //////////////
if (iter % 4*NDIMS) == 0:
    #calculate the avarage (including maximum value)
    pavg=0
    for i in range(NPTS):
         pavg=pavg+p[i][FUNC]
    pavg=pavg/NPTS
    tot=0.0
    if printlist!=0:
        print(iter)
        for j in range(NDIMS):
            print(p[ilo][j]," ")
    for i in range(NPTS):
        tot=(p[i][FUNC]-pavg)*(p[i][FUNC]-pavg)
    sterr=sqrt(tot/NPTS)
    for j in range(NDIMS):
        z[j]=p[ilo][j]
    best = p[ilo][FUNC];
# end of if iter%4*NDMS)==0
#//// calculate avarage without maximum value //////
ave=[0.0 for i in range(NDIMS)]
for j in range(NDIMS):
    ave[j] = 0
for i in range(NPTS):
    if i != ihi:
        for j in range(NDIMS):
            ave[j] = ave[j]+p[i][j];
for j in range(NDIMS):
    ave[j]=ave[j]/(NPTS-1)
#print("ave[",j,"]=",ave[j])
#//////// reflect ////////////////
#print("reflect")
r=[0.0 for i in range(NDIMS)]
for j in range(NDIMS):
    r[j] = 2.0*ave[j] - p[ihi][j]
fr = fnelder.func(r)
if (flo <= fr) and (fr < fnhi): #in zone: accept
    for j in range(NDIMS):
        p[ihi][j] = r[j]
    p[ihi][FUNC] = fr
    continue
#///(//////////// expand
if fr < flo:
    #print("expand")
    e=[0.0 for i in range(NDIMS)]
    for j in range(NDIMS):
        e[j] = 3.0*ave[j] - 2.0*p[ihi][j];
    fe = fnelder.func(e);
    if fe < fr:
        for j in range(NDIMS):
            p[ihi][j] = e[j]
        p[ihi][FUNC] = fe
        continue
    else:
```

```
            for j in range(NDIMS):
                p[ihi][j] = r[j]
            p[ihi][FUNC] = fr
            continue
        #//////////// shrink:
        if fr < fhi:
            #print("srink")
            c=[0.0 for i in range(NDIMS)]
            for j in range(NDIMS):
                c[j] = 1.5*ave[j] - 0.5*p[ihi][j]
            fc = fnelder.func(c);
            if fc <= fr:
                for j in range(NDIMS):
                    p[ihi][j] = c[j];
                p[ihi][FUNC] = fc
                continue
            else:   #////// shrink
                for i in range(NPTS):
                    if i != ilo:
                        for j in range(NDIMS):
                            p[i][j] = 0.5*p[ilo][j] + 0.5*p[i][j]
                        p[i][FUNC] = fnelder.func(p[i])
                continue
        #/////////////// contract
        if fr >= fhi:
            #print("contract")
            cc=[0.0 for i in range(NDIMS)]
            for j in range(NDIMS):
                cc[j] = 0.5*ave[j] + 0.5*p[ihi][j];
            fcc = fnelder.func(cc)
            if fcc < fhi:
                for j in range(NDIMS):
                    p[ihi][j] = cc[j];
                p[ihi][FUNC] = fcc
                continue
            else:
                for i in range(NPTS):
                    if i != ilo:
                        for j in range(NDIMS):
                            p[i][j] = 0.5*p[ilo][j] + 0.5*p[i][j]
                        p[i][FUNC] = fnelder.func(p[i]);
    return z

a=[30.0,1.0]
da=[1.0,0.2]
name="input3.txt"
f=y(name,a)
ai=nelder(f,a,da,100,1.0e-15,0)
print("ai = ",ai)
n1=100
xx1 =[0 for w in range(n1)]
yy1 =[0 for w in range(n1)]
dx=(30.0-5.0)/(n1-1)
for i in range(n1):
    xx1[i]=5.0+dx*i
    yy1[i]=f.Ps(xx1[i],ai)
plt.plot(xx1,yy1,f.xi,f.yi,"+")
plt.xlabel("x")
plt.ylabel("y")
plt.title("non linear least square by using Nelder-Mead optimisation method ")
plt.show()
```

```
runfile('E:/okul/SCO1/nelder_opt.py', wdir='E:/okul/SCO1')
Reloaded modules: f_xj
ai =  [38.727045551319954, 0.1074564960913112]
```



10.5 MOVİNG LEAST SQUARES WITH SINGLE AND MULTI DIMENSIONAL VARIABLES

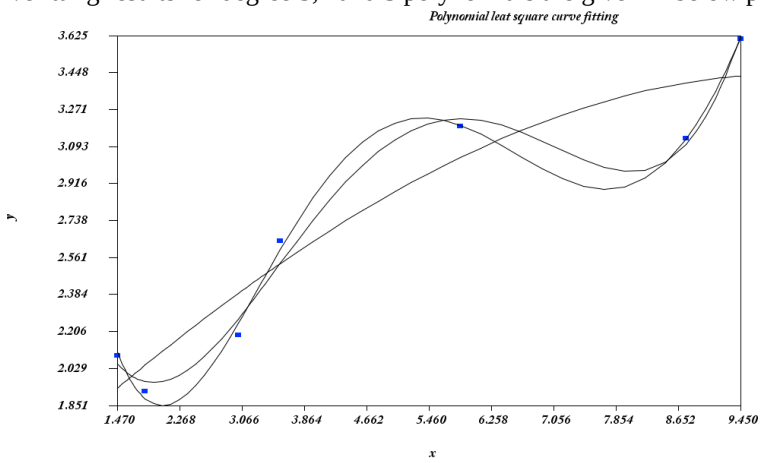
Polynomial least square equation was defined as

$$\frac{\partial H(a_0^{(m)},..,a_0^{(m)})}{\partial a_k^{(m)}} = 2\sum_{i=1}^{n} w(x_i)\left[f(x_i) - \sum_{j=0}^{m} a_j^{(m)} x^j\right] x^k = 0 \quad k = 0,\dots,m \qquad (10.5-1)$$

and this equation was solved for weight function $w(x_i)$=1. If it is desired that the accuracy of the equation to be improved, it will be seen that increasing coeefficients will not always give beter accuracy. To see this effect assume the following data and make 3rd 4th and 5th degree curve fitting into the data.

x	1.47	1.83	3.02	3.56	5.86	8.75	9.45
f	2.09	1.92	2.19	2.64	3.19	3.13	3.61

Curvefitting results for degree 3,4 and 5 polynomials are given in below plot.



In here all the equations given different kind of results. Which one is correct or closer to the data then? It is not always easy to answer. As a different approach to this curve fitting, lets apply a continiously changing weight factor

$$w_i(x) = \frac{1}{|x - x_i|^k} \quad (10.5.2)$$

In this function when x value become equal to x_i the value of weight factor will go to infinity. $x \to x_i$ $w_i(x) \to \infty$. If it is assumed that for $x=x_i$, $f=f_i$ and solve the least square for the remaining points solution will be existed. But in this type of conditions due to the fact that w_i was changing in each region, least square curve fitting coefficients should be calculated for each point(region) separetely. This type of least square method is called **moving least squares.** An example program to moving least squares are presented below

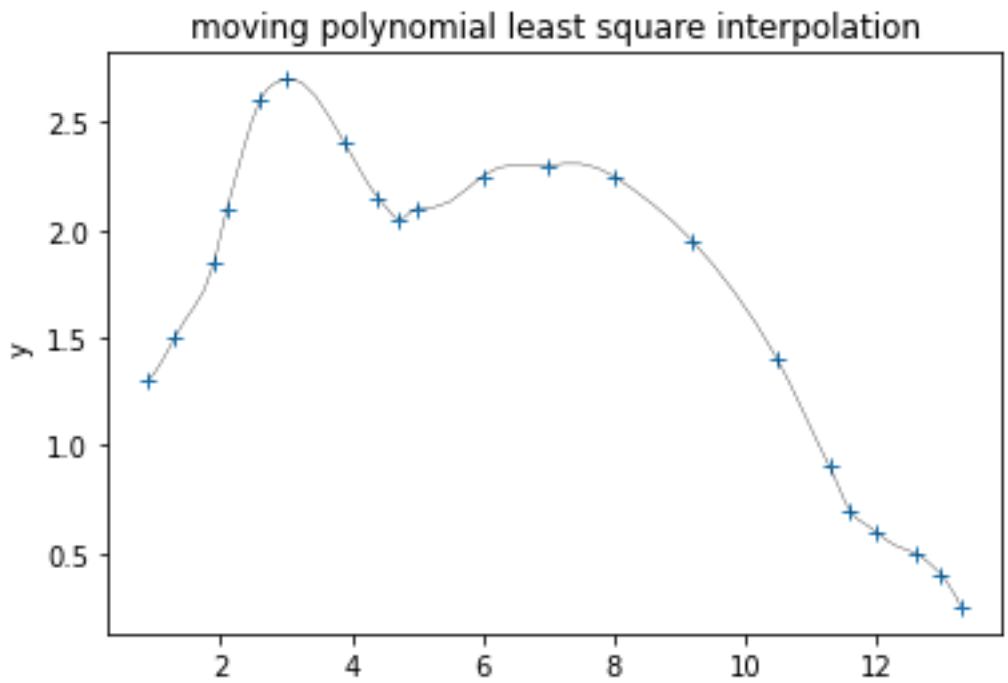
Program Moving least square curve fitting

```
# Moving least Square interpolation
from math import *
from gauss import *
import matplotlib.pyplot as plt

def movingLSTSQR(x,xi,yi,n,kk):
    # polynomial moving least square
    l=len(xi)
    np1=n+1
    A=[[0 for j in range(np1)] for i in range(np1)]
    B=[0 for j in range(np1)]
    X=[0 for j in range(np1)]
    w=[0 for j in range(l)]
    for i in range(l):
        if x==xi[i]:
            return yi[i]
    for i in range(l):
        w[i]=1.0/pow(abs(x-xi[i]),kk)
    for i in range(np1):
        for j in range(np1):
            if i==0 and j==0:
                A[i][j]=l;
            else:
                for k in range(l):
                    A[i][j] = A[i][j]+w[k]*pow(xi[k],(i+j));
        for k in range(l):
            if i==0:
                B[i]= B[i]+w[k]*yi[k];
            else:
                B[i]= B[i] +w[k]*pow(xi[k],i)*yi[k]
    X=gauss(A,B)
    #print("X=",X)
    max=0.0
    for i in range(n+1):
        if abs(X[i]) > max:
            max = abs(X[i])
    for i in range(n+1):
        if abs(X[i]/max) > 0 and abs(X[i]/max) < 1.0e-100:
            X[i]=0
    Y=funcEKK(X,x)
    #print("x=",x,"Y=",Y)
    return Y


def funcEKK(e,x):
    # this function calculates the value of
    # least square curve fitting function
    n=len(e)
    if n!=0:
        ff=e[n-1]
        for i in range(n-2,-1,-1):
            ff=ff*x+e[i]
    else:
        ff=0.0;
    return ff


def output(xi,yi,polinomkatsayisi,aradegersayisi,kk):
    n=len(xi)
    nn=(n-1)*(aradegersayisi+1)+1
    z=[[0 for j in range(nn)] for i in range(2)]
    dx=0.0
    k=0
```

```
    for i in range(n-1):
        z[0][k]=xi[i];z[1][k]=movingLSTSQR(z[0][k],xi,yi,polinomkatsayisi,kk)
        k=k+1
        for j in range(aradegersayisi):
            dx=(xi[i+1]-xi[i])/(float(aradegersayisi)+1.0)
            z[0][k]=z[0][k-1]+dx;z[1][k]=movingLSTSQR(z[0][k],xi,yi,polinomkatsayisi,kk)
            k=k+1
    z[0][k]=xi[n-1]
    z[1][k]=movingLSTSQR(z[0][k],xi,yi,polinomkatsayisi,kk)
    return z

def read(name):
    path = name
    file1 = open(path,'r')
    file1.seek(0,2)  #jumps to the end
    endLocation=file1.tell()
    file1.seek(0) #jumps to the start
    nn=0
    while True:
        line=file1.readline()
        nn+=1
        if  file1.tell()==endLocation:
            break
    x=[0.0 for i in range(nn)]
    y=[0.0 for i in range(nn)]
    file1.seek(0) #jumps to the start
    for i in range(nn):
        line=file1.readline()
        ee=line.split()
        x[i]=float(ee[0])
        y[i]=float(ee[1])
        #print(x[i]," ",y[i]," ",z[i])
    file1.close()
    a=[x,y]
    #return Transpose(a)
    return a
def Transpose(left):
    # transpose matrix (if A=a(i,j) Transpose(A)=a(j,i)
    # int i,j;
    n=len(left)
    m=len(left[0])
    b=[[0 for j in range(n)] for i in range(m)]
    for i in range(n):
        for j in range(m):
            b[j][i]=left[i][j]
    return b

x=[0.9,1.3,1.9,2.1,2.6,3.0,3.9,4.4,4.7,5.0,6.0,7.0,8.0,9.2,10.5,11.3,11.6,12.0,12.6,13.0,13.3]
y=[1.3,1.5,1.85,2.1,2.6,2.7,2.4,2.15,2.05,2.1,2.25,2.3,2.25,1.95,1.4,0.9,0.7,0.6,0.5,0.4,0.25]
print("x=",x,"\ny=",y)
z1=output(x,y,10,3,2)
print("Moving least square interpolation \n",Transpose(z1))
plt.title('moving polynomial least square interpolation')
plt.ylabel('y ');
plt.plot(x,y,'+',z1[0],z1[1],'k',linewidth=0.3)
```

```
runfile('E:/okul/SCO1/movingLSTSQR.py', wdir='E:/okul/SCO1')
Reloaded modules: gauss
x= [0.9, 1.3, 1.9, 2.1, 2.6, 3.0, 3.9, 4.4, 4.7, 5.0, 6.0, 7.0, 8.0, 9.2, 10.5, 11.3, 11.6, 12.0, 12.6, 13.0, 13.3]
y= [1.3, 1.5, 1.85, 2.1, 2.6, 2.7, 2.4, 2.15, 2.05, 2.1, 2.25, 2.3, 2.25, 1.95, 1.4, 0.9, 0.7, 0.6, 0.5, 0.4, 0.25]
Moving least square interpolation
 [[0.9, 1.3], [1.0, 1.3399074194810645], [1.1, 1.3884269966852716], [1.2000000000000002, 1.4432593682824828], [1.3, 1.5], [1.45,
1.57801268377964], [1.5999999999999999, 1.643820985979865], [1.7499999999999998, 1.7204114787805167], [1.9, 1.85], [1.95,
1.9101641480772407], [2.0, 1.9757394093681733], [2.05, 2.040388880901058], [2.1, 2.1], [2.225, 2.2385686507389932], [2.35,
2.3760628693879484], [2.475, 2.5019715447208606], [2.6, 2.6], [2.7, 2.6493687856781385], [2.8000000000000003,
2.675628371156134], [2.9000000000000004, 2.690627045775627], [3.0, 2.7], [3.225, 2.687941544806875], [3.45,
2.621196612787153], [3.6750000000000003, 2.5172433271799317], [3.9, 2.4], [4.025, 2.333542777433391], [4.15,
2.266759243816306], [4.275, 2.2031290658102662], [4.4, 2.15], [4.4750000000000005, 2.123298654620633], [4.550000000000001,
2.0964777481116728], [4.625000000000001, 2.06908012415949], [4.7, 2.05], [4.775, 2.0511411090741265], [4.8500000000000005,
2.0696495097577334], [4.925000000000001, 2.0891955432432976], [5.0, 2.1], [5.25, 2.1044708997799955], [5.5,
2.130582590521252], [5.75, 2.187517149684822], [6.0, 2.25], [6.25, 2.2905851628753586], [6.5, 2.3008898758714458], [6.75,
2.2977113105299214], [7.0, 2.3], [7.25, 2.3076216999052632], [7.5, 2.3072595475316318], [7.75, 2.288264584352389], [8.0, 2.25],
[8.3, 2.188014530103003], [8.600000000000001, 2.118324795204275], [8.900000000000002, 2.040457489270313], [9.2, 1.95],
[9.524999999999999, 1.836620236227467], [9.849999999999998, 1.7091493662389496], [10.174999999999997, 1.565955655881343],
[10.5, 1.4], [10.7, 1.282028995015975], [10.899999999999999, 1.153959493443806], [11.099999999999998, 1.0242851249841232],
[11.3, 0.9], [11.375, 0.8512700359358453], [11.45, 0.7980280248453694], [11.524999999999999, 0.7443948590388215], [11.6, 0.7],
```

[11.7, 0.6635742924114457], [11.799999999999999, 0.6430473518972486], [11.899999999999999, 0.6230474980782192], [12.0, 0.6], [12.15, 0.5643837632309205], [12.3, 0.5356126866637697], [12.450000000000001, 0.5181854543312083], [12.6, 0.5], [12.7, 0.4820938600467023], [12.799999999999999, 0.4588386603926626], [12.899999999999999, 0.43150243883920625], [13.0, 0.4], [13.075, 0.3725154872046411], [13.149999999999999, 0.33564367020276686], [13.224999999999998, 0.2996157341148143], [13.3, 0.25]]



The biggest disadvantage of this method is the requirement of matrix solution for each curve fitting point seperately. The equations are passing through each point. Therefore, this method is an interpolation methods compare to least square interpolations. Method can be applied also by using orthogonal polynomials, instead of normal polynomials, and this way disadvantage of applying many matrix solutions will be removed from the equation set.

Moving least square method can similarly be applied into multi-independent variable curve fitting problems. In principles number of independent variables in the equation does not effect the curve fitting process, except the definition of the functions. If a degree m polynomial

$$f(x_0, x_{1,}.., x_n) = \sum_{j=0}^{m} a_j^{(m)} \phi_j((x_0, x_{1,}.., x_n) \qquad (6.3-1)$$

j th degree function is given. It is desired to fit x_0, x_1, x_2,…, x_n ,f_i, i=0...n data set into this function. The best fitted $a_j^{(m)}$ values to be found. For this purpose Minimum value of

$$H(x_0, x_{1,}.., x_n) = \sum_{i=1}^{n} w(x_0, x_{1,}.., x_n) \left[f_i - \sum_{j=0}^{m} a_j^{(m)} \phi_j(x_0, x_{1,}.., x_n) \right]^2 \qquad (6.3-2)$$

function should be found. $w(x_0, x_{1,}.., x_n)$ is called weight function and it should satisfy the condition $w(x_0, x_{1,}.., x_n) \geq 0 \quad i = 1, \dots, n.$ The minimum of the function is the root of the derivative of the function.

$$\frac{\partial H(x_0, x_{1,}.., x_n)}{\partial a_k^{(m)}} = 2 \sum_{i=1}^{n} w(x_0, x_{1,}.., x_n) \left[f_i - \sum_{j=0}^{m} a_j^{(m)} \phi_j(x_0, x_{1,}.., x_n) \right] \phi_k(x_0, x_{1,}.., x_n) = 0 \quad k = 0,..,m \quad (6.3-2)$$

If unit value is taken as weight function, equation becomes:

$$2 \sum_{i=1}^{n} \left[f_i - \sum_{j=0}^{m} a_j^{(m)} \phi_j(x_0, x_{1,}.., x_n) \right] \phi_k(x_0, x_{1,}.., x_n) = 0 \qquad k = 0,..,m \quad (6.3-2a)$$

This equation was solved in chapter 6.3 by using $w(x_0, x_{1,}.., x_n)$=1 as classical least square approach. In order to define weight factor function Euclidian distance between the points will be used. If two vectoral point is defined in Euclidian space as:

$x = (x_0, x_{1,}.., x_n)$ (6.10-1)
$x^i = (x_0^i, x_1^i, .., x_n^i)$ (6.10-2)
Euclidian distance between x and x^i will be given by

$$d(x, x^i) = \sqrt{\sum_{j=0}^{n} (x_j - x_j^i)^2} \qquad (6.10-3)$$

$w_i(x) = w(d(x, x^i))$ (6.10-4)
Now the weight function can be defined as a function of Euclidian distance as:
.
As an example definition of an actual weight function the following equation can be given

$$w(d) = \frac{e^{-d^2/16}}{d^2 + 0.001}$$

In this equations x is the point that curve fitting value is desired and x^i is the neighbouring point. When this weight factor is applied, program in the chapter 6.3 will be converted to the moving least square form.

Example data for the program x_{ij},y_i sample data is generated by using the function $y=x_0^2+x_1^2$

0	0	0	3	0	9
0	1	1	3	1	10
0	2	4	3	2	13
0	3	9	3	3	18
0	4	16	3	4	25
1	0	1	4	0	16
1	1	2	4	1	17
1	2	5	4	2	20
1	3	10	4	3	25
1	4	17	4	4	32
2	0	4			
2	1	5			
2	2	8			
2	3	13			
2	4	20			

```
 from f_xir import *
from gauss import *

class f1(f_xir):
#f(x,y)=a0+a1*x+a2*y+a3*x^2+a4*y^2+a5*xy+a6*x^3+a7*y^3+a8*x^2y
#      +a9*xy^2+a10*x^4+a11*y^4+a12*x^2y^2+a13x^3y+a14*xy^3
   def func(self,x,i):
      xx=0.0
      if i==0: xx=1.0
      elif i==1:   xx=x[0]
      elif i==2:   xx=x[1]
      elif i==3:   xx=x[0]*x[0]
      elif i==4:   xx=x[1]*x[1]
      elif i==5:   xx=x[0]*x[1]
      elif i==6:   xx=x[0]*x[0]*x[0]
      elif i==6:   xx=x[0]*x[0]*x[0]
      elif i==7:   xx=x[1]*x[1]*x[1]
      elif i==8:   xx=x[0]*x[0]*x[1]
      elif i==9:   xx=x[0]*x[1]*x[1]
      elif i==10:   xx=x[0]*x[0]*x[0]*x[0]
      elif i==11:   xx=x[1]*x[1]*x[1]*x[1]
      elif i==12:   xx=x[0]*x[0]*x[1]*x[1]
      elif i==13:   xx=x[0]*x[0]*x[0]*x[1]
      elif i==14:   xx=x[0]*x[1]*x[1]*x[1]
      return xx

# moving least square curve fitting with multivariable
#Euler distance
```

```
def d(x,xi):
    # Euler distance
    total=0
    n=len(x)
    for i in range(n):
        total=total+(x[i]-xi[i])*(x[i]-xi[i])
    return sqrt(total)
#weight function
def w(x,xi):
    d1=d(x,xi)
    eps=0.001
    alpha=1.0/16.0
    ww=exp(-alpha*d1*d1)/(d1*d1+eps)
    return ww

def Transpose(left):
    # transpose matrix (if A=a(i,j) Transpose(A)=a(j,i)
    # int i,j;
    n=len(left)
    m=len(left[0])
    b=[[0 for j in range(n)] for i in range(m)]
    for i in range(n):
        for j in range(m):
            b[j][i]=left[i][j]
    return b

def HEKK(x,xi,yi,n):
# n dimensional surface general least square curve fitting
# for the vector point x
    f=f1()
    l=len(xi)
    np1=n+1
    A=[[0 for j in range(np1)] for i in range(np1)]
    B=[0 for j in range(np1)]
    E=[0 for j in range(np1)]
    for i in range(n+1):
        B[i]=0
        for j in range(np1):
            A[i][j]=0.0
            for k in range(l):
                xx1=f.func(xi[k],i)
                xx2=f.func(xi[k],j)
                A[i][j]=A[i][j]+w(x,xi[k])*xx1*xx2
        for k in range(l):
                B[i]=B[i]+w(x,xi[k])*f.func(xi[k],i)*yi[k]
    E=gauss(A,B)
    max=0
    for i in range(n+1):
        if abs(E[i]) > max: max=abs(E[i])
    for i in range(n+1):
        if abs(E[i])/max > 0 and abs(E[i]/max) < 1.0e-100: E[i]=0.0
    X=func(E,x)
    return X

def func(e,x):
    #multidimensional function calculation
    f=f1()
    n=len(e)
    ff=0.0
    if n!=0:
        for i in range(n-1,-1,-1):
            ff=ff+e[i]*f.func(x,i)
    return ff

def HEKKn(x,xi,yi,n):
    # n dimension surface curve fitting for the array of vector points x
```

```
    m=len(x)
    z=[0 for j in range(m)]
    for i in range(m):
        z[i]= HEKK(x[i],xi,yi,n)
    return z;

def read(name):
    path = name
    file1 = open(path,'r')
    file1.seek(0,2)  #jumps to the end
    endLocation=file1.tell()
    file1.seek(0) #jumps to the start
    nn=0
    while True:
        line=file1.readline()
        nn+=1
        if  file1.tell()==endLocation:
            break
    x=[0.0 for i in range(nn)]
    y=[0.0 for i in range(nn)]
    z=[0.0 for i in range(nn)]
    file1.seek(0) #jumps to the start
    for i in range(nn):
        line=file1.readline()
        ee=line.split()
        x[i]=float(ee[0])
        y[i]=float(ee[1])
        z[i]=float(ee[2])
        #print(x[i]," ",y[i]," ",z[i])
    file1.close()
    a=[x,y,z]
    return a
    #return Transpose(a)
def read1(name):
    path = name
    file1 = open(path,'r')
    file1.seek(0,2)  #jumps to the end
    endLocation=file1.tell()
    file1.seek(0) #jumps to the start
    nn=0
    while True:
        line=file1.readline()
        nn+=1
        if  file1.tell()==endLocation:
            break
    x=[0.0 for i in range(nn)]
    y=[0.0 for i in range(nn)]
    #z=[0.0 for i in range(nn)]
    file1.seek(0) #jumps to the start
    for i in range(nn):
        line=file1.readline()
        ee=line.split()
        x[i]=float(ee[0])
        y[i]=float(ee[1])
        #z[i]=float(ee[2])
        #print(x[i]," ",y[i]," ",z[i])
    file1.close()
    a=[x,y]
    #return a
    return Transpose(a)

def output(x,xi,yi,n):
    #gives curve fitting values together with the independent value set
    m=len(x)
    nn=len(x[0])
    c=[[0.0 for j in range(nn+1)] for i in range(m)]
```

```
    #z=[[0.0 for j in range(nn+1)] for i in range(m)]
    for i in range(m):
        for j in range(nn):
            c[i][j]=x[i][j]
    for i in range(m):
        c[i][nn]= HEKK(x[i],xi,yi,n)
    return c

def z(xmin,xmax,dx,ymin,ymax, dy):
    n=int((xmax-xmin)/dx+0.0001)
    m=int((ymax-ymin)/dy+0.0001)
    nn=(n+1)*(m+1)
    z=[[0 for j in range(2)] for i in range(nn)]
    x=0.0
    y=0.0
    for i in range(n+1):
        x=xmin+i*dx
        for j in range(m+1):
            y=ymin+j*dy
            z[i*(m+1)+j][0]=x
            z[i*(m+1)+j][1]=y
    return z
# Creating dataset


a=read('d.txt')
xy=z(0.0,4.0,0.5,0.0,4.0,0.5)
x=Transpose([a[0],a[1]])
y=a[2]
z=output(xy,x,y,12)
print(z)
```

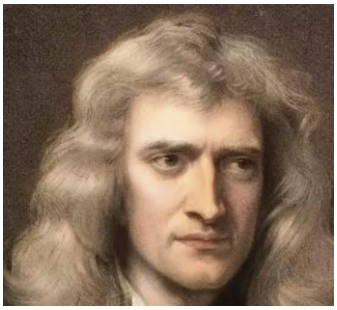
```
 runfile('E:/okul/SCO1/moving_multidimensional_LSCF.py', wdir='E:/okul/SCO1')
Reloaded modules: f_xir, gauss
[[0.0, 0.0, -1.0620629998416146e-17], [0.0, 0.5, 0.25000000000000744], [0.0, 1.0, 0.9999999999999989], [0.0, 1.5,
2.2499999999999676], [0.0, 2.0, 3.999999999999995], [0.0, 2.5, 6.249999999999732], [0.0, 3.0, 9.000000000000027], [0.0, 3.5,
12.25000000000442], [0.0, 4.0, 15.999999999999977], [0.5, 0.0, 0.2499999999999984], [0.5, 0.5, 0.5000000000002016], [0.5, 1.0,
1.250000000000321], [0.5, 1.5, 2.4999999999998512], [0.5, 2.0, 4.249999999999982], [0.5, 2.5, 6.499999999999857], [0.5, 3.0,
9.25000000000014], [0.5, 3.5, 12.500000000010392], [0.5, 4.0, 16.250000000000547], [1.0, 0.0, 0.9999999999999979], [1.0, 0.5,
1.249999999999851], [1.0, 1.0, 2.0000000000000018], [1.0, 1.5, 3.2499999999999925], [1.0, 2.0, 5.00000000000002], [1.0, 2.5,
7.249999999999838], [1.0, 3.0, 9.999999999999963], [1.0, 3.5, 13.250000000011042], [1.0, 4.0, 17.00000000000005], [1.5, 0.0,
2.250000000000023], [1.5, 0.5, 2.49999999999966], [1.5, 1.0, 3.249999999999964], [1.5, 1.5, 4.500000000000141], [1.5, 2.0,
6.25000000000015], [1.5, 2.5, 8.500000000000169], [1.5, 3.0, 11.249999999999902], [1.5, 3.5, 14.499999999998959], [1.5, 4.0,
18.249999999999897], [2.0, 0.0, 4.000000000000005], [2.0, 0.5, 4.249999999999921], [2.0, 1.0, 5.000000000000009], [2.0, 1.5,
6.249999999999875], [2.0, 2.0, 8.000000000000053], [2.0, 2.5, 10.250000000000226], [2.0, 3.0, 13.00000000000017], [2.0, 3.5,
16.249999999996344], [2.0, 4.0, 19.999999999999957], [2.5, 0.0, 6.249999999999629], [2.5, 0.5, 6.499999999999341], [2.5, 1.0,
7.249999999999541], [2.5, 1.5, 8.50000000000014], [2.5, 2.0, 10.250000000000174], [2.5, 2.5, 12.500000000000274], [2.5, 3.0,
15.25000000000006], [2.5, 3.5, 18.5000000000023], [2.5, 4.0, 22.25000000000043], [3.0, 0.0, 9.000000000000007], [3.0, 0.5,
9.250000000000133], [3.0, 1.0, 10.000000000000012], [3.0, 1.5, 11.249999999999844], [3.0, 2.0, 13.000000000000135], [3.0, 2.5,
15.250000000000053], [3.0, 3.0, 18.00000000000015], [3.0, 3.5, 21.25000000000457], [3.0, 4.0, 25.000000000000234], [3.5, 0.0,
12.25000000000178], [3.5, 0.5, 12.500000000005063], [3.5, 1.0, 13.250000000004862], [3.5, 1.5, 14.499999999999472], [3.5, 2.0,
16.250000000001727], [3.5, 2.5, 18.49999999999745], [3.5, 3.0, 21.249999999990678], [3.5, 3.5, 24.499999999997236], [3.5, 4.0,
28.249999999999815], [4.0, 0.0, 16.000000000000004], [4.0, 0.5, 16.249999999999723], [4.0, 1.0, 17.000000000000043], [4.0, 1.5,
18.25000000000004], [4.0, 2.0, 19.999999999999986], [4.0, 2.5, 22.250000000000902], [4.0, 3.0, 25.00000000000011], [4.0, 3.5,
28.250000000000192], [4.0, 4.0, 32.00000000000008]]
```

10.6 INTERPOLATION: NEWTON AND LAGRANGE INTERPOLATION METHODS

We sometimes know the value of a function $f(x)$ at a set of points $x_1, x_2, \ldots, x_N$ (say, with $x_1 < \ldots < x_N$), but we don't have an analytic expression for $f(x)$ that lets us calculate its value at an arbitrary point. For example, the $f(x_i)$'s might result from some physical measurement or from long numerical calculation that cannot be cast into a simple functional form. Often the x_i's are equally spaced, but not necessarily.

The task now is to estimate $f(x)$ for arbitrary x by, in some sense, drawing a smooth curve through (and perhaps beyond) the x_i. If the desired x is in between the largest and smallest of the x_i's, the problem is called *interpolation*; if x is outside that range, it is called *extrapolation*,which is considerably more hazardous. Basic difference of interpolation compare to least square curve fitting is that interpolation is always passing through the data points. Some of the most common interpolation methods are given here:

10.6.1 Newton Interpolation Polynomial



Sir Isaac Newton

Newton interpolation Formula of degree n+1 is given as:

$$f_n(x) = b_0 + b_1(x - x_0) + \cdots + b_n(x - x_0)\big(x - x_1..(x - x_n)\big)$$

$$b_0 = f(x_0)$$

$$b_1 = f[x_1, x_0]$$

$$b_2 = f[x_2, x_1, x_0]$$

…

$$b_n = f[x_n, x_{n-1}, .., x_2, x_1, x_0] \quad (10.6\text{-}1)$$

In this definition :

$$f[x_i, x_j] = \frac{f(x_i) - f(x_j)}{x_i - x_i} \quad (10.6.2a)$$

$$f[x_i, x_j, x_k] = \frac{f[x_i, x_j] - f[x_j, x_k]}{x_i - x_k} \quad (10.6.2b)$$

Similarly nth degree divided difference Formula can be given as:

$$f[x_n, x_{n-1}, \dots, \dots, x_1, x_0] = \frac{f[x_n, x_{n-1}, .., x_1] - f[x_{n-1}, x_{n-2}, \dots, \dots, x_1, x_0]}{x_n - x_0} \quad (10.6.2c)$$

This equation called as Newton's interpolation polynomial.

An example problem :

Population of City A is given in the table below. Calculate population of the city for year 1945

year	1940	1950	1960	1970	1980	1990
Population	132165	151336	179323	203302	226542	249633

Newton's divided difference table :

Xi	yi	f[xi,xj]	f[xi,xj,xk]	f[xi,xj,xk,xl]	f[xi,xj,xk,xl,xn]	f[xi,xj,xk,xl,xn,xm]
1940	132165	1917.1	44.08	-2.137333333	0.067054167	-0.001564333
1950	151336	2798.7	-20.04	0.544833333	-0.0111625	
1960	179323	2397.9	-3.695	0.098333333		
1970	203302	2324	-0.745			
1980	226542	2309.1				
1990	249633					

Solution with increasing order can be given as:

132165+(1945-1940)*1917.1= 141750.5

141750.5+(1945-1940)* (1945-1950)* 44.08=140648.5

140648.5+(1945-1940)(1945-1950)*(1945-1960)* -2.137333333=139847

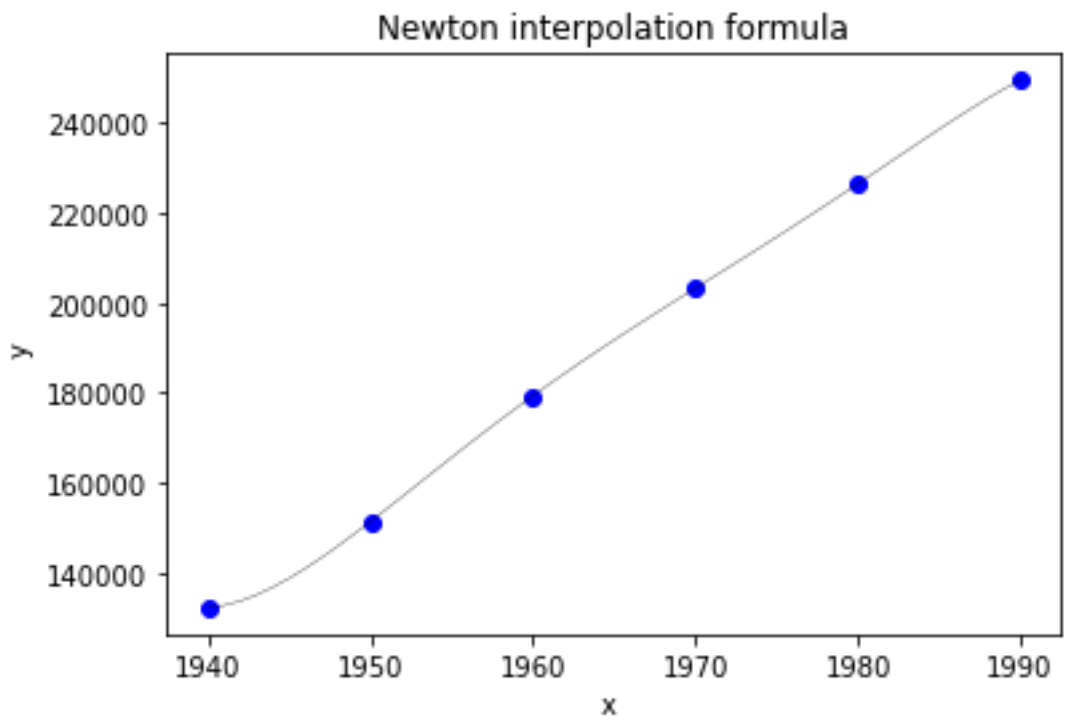
139847+(1945-1940)(1945-1950)*(1945-1960)*(1945-1970)* 0.067054167 =139218.4

139218.4+(1945-1940)(1945-1950)*(1945-1960)*(1945-1970)*(1945-1980)* -0.001564333=138705.1

The most important advantage of Newton intrepolation formula is the additive characters. In order to calculate (n+1) th degree of polynomial all you need is nth degree polynomial difference values.

```
 from math import *
import numpy as np;
import matplotlib.pyplot as plt

def Newton(xi,yi):
    n=len(xi)
    F=[[0 for j in range(n)] for i in range(n+1)]
    for i in range(n):
        F[i][0]=yi[i]
    for i in range(n):
        for j in range(1,i+1):
            F[i][j]=(F[i][j-1]-F[i-1][j-1])/(xi[i]-xi[i-j])
    for i in range (n):
         F[n][i]=xi[i]
    return F

def func(F,x):
    n=len(F[0])
    carpim=1.0
    toplam=0
    for i in range(n):
        carpim=F[i][i]
        for j in range(i):
            carpim=carpim*(x-F[n][j])
        toplam=toplam+carpim;
    return toplam;

def funcNewton(xi,yi,numberofmidpoints):
    # numberofmidpoints :  in x--o--o--x--o--o--x chain if x's are esxperimental points
    # numberofmidpoints is 2
    n=len(xi)
    nn=(n-1)*(numberofmidpoints+1)+1
    z=[[0 for j in range(nn)] for i in range(2)]
    Q=Newton(xi,yi)
    dx=0.0
    k=0
    i=0
    for i in range(n-1):
        z[0][k]=xi[i];z[1][k]=func(Q,z[0][k])
        k=k+1
        for j in range(numberofmidpoints):
            dx=(xi[i+1]-xi[i])/(float(numberofmidpoints)+1.0)
            z[0][k]=z[0][k-1]+dx
            z[1][k]=func(Q,z[0][k])
            k=k+1
    z[0][k]=xi[n-1]
    z[1][k]=func(Q,z[0][k])
    return z
def Transpose(left):
    # transpose matrix (if A=a(i,j) Transpose(A)=a(j,i)
    # int i,j;
    n=len(left)
    m=len(left[0])
    b=[[0 for j in range(n)] for i in range(m)]
    for i in range(n):
        for j in range(m):
            b[j][i]=left[i][j]
    return b


x=[1940,1950,1960,1970,1980,1990]
y=[132165,151336,179323,203302,226542,249633]
print("x=",x,"\ny=",y)
z1=funcNewton(x,y,9)
print("Newton interpolation\n",Transpose(z1))
plt.title('Newton interpolation formula')
plt.xlabel('x ')
plt.ylabel('y ');
plt.plot(x,y,'bo',z1[0],z1[1],'k',linewidth=0.3,)
```

```
runfile('E:/okul/SCO1/newton_interpolation.py', wdir='E:/okul/SCO1')
x= [1940, 1950, 1960, 1970, 1980, 1990]
y= [132165, 151336, 179323, 203302, 226542, 249633]
Newton interpolation
 [[1940, 132165.0], [1941.0, 132684.8307565], [1942.0, 133658.281408], [1943.0, 135023.3483545], [1944.0, 136723.01625599997],
[1945.0, 138705.0703125], [1946.0, 140921.908544], [1947.0, 143330.3540705], [1948.0, 145891.467392], [1949.0, 148570.3586685],
[1950, 151336.0], [1951.0, 154161.0377065], [1952.0, 157021.60460800002], [1953.0, 159897.13230449997], [1954.0,
162770.16345599998], [1955.0, 165626.1640625], [1956.0, 168453.33574399998], [1957.0, 171242.42802049997], [1958.0,
173986.55059199999], [1959.0, 176680.98561849998], [1960, 179323.0], [1961.0, 181911.65765650003], [1962.0, 184447.631808],
[1963.0, 186933.01725449998], [1964.0, 189371.142656], [1965.0, 191766.3828125], [1966.0, 194123.97094400003], [1967.0,
196449.8109705], [1968.0, 198750.289792], [1969.0, 201032.0895685], [1970, 203302.0], [1971.0, 205566.7306065], [1972.0,
207832.72300800003], [1973.0, 210105.9632045], [1974.0, 212391.793856], [1975.0, 214694.7265625], [1976.0, 217018.254144],
[1977.0, 219364.6629205], [1978.0, 221734.844992], [1979.0, 224128.1105185], [1980, 226542.00000000003], [1981.0,
228972.09655649998], [1982.0, 231411.83820800006], [1983.0, 233852.33015450006], [1984.0, 236282.15705599997], [1985.0,
238687.1953125], [1986.0, 241050.42534399996], [1987.0, 243351.74387050004], [1988.0, 245567.776192], [1989.0,
247671.68846850004], [1990, 249633.0]]
```



10.6.2 Lagrange Interpolation formula

Joseph-Loise Lagrange

Lagrange interpolation polynomial is the reformulated version of the Newton's interpolation polynomial. Lagrange Formula do not require calculation of divided differences.

$f_n(x) = \sum_{i=0}^{n} L_i(x) f(x_i)$ (10.6.3) Where

$L_i(x) = \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{x-x_i}{x_i-x_j}$. (10.6-4) Where $\prod$ indicates multiplication process.

Open form of the equation;

for the first degree (linear) polinomial (n=1)

$f_1(x) = \frac{x-x_1}{x_0-x_1} f(x_0) + \frac{x-x_0}{x_1-x_0} f(x_1)$ (10.6-5)

for the first degree (quadratic) polinomial (n=2)

$f_2(x) = \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} f(x_0) + \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)} f(x_1) + \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_2)} f(x_2)$ (10.6-6)

İnput data: input1.txt

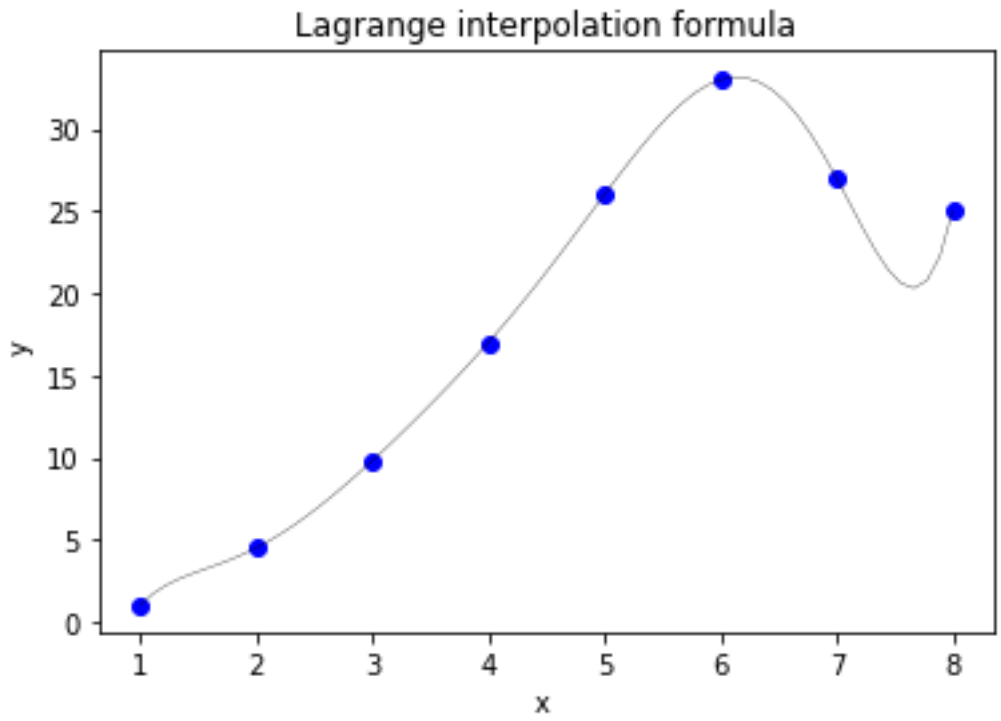
```
1 1
2 4.5
3 9.8
4 17
5 26
6 33
7 27
8 25
```

```
 from math import *
import numpy as np;
import matplotlib.pyplot as plt

def Lagrange(xi,yi):
    n=len(xi)
    L=[[0 for j in range(n)] for i in range(2)]
    for i in range(n):
        L[0][i]=yi[i]
        for j in range(n):
            if i!=j:
                L[0][i]=L[0][i]/(xi[i]-xi[j])
    for i in range (n):
         L[1][i]=xi[i]
    return L

def func(L,x):
    n=len(L[0])
    carpim=1.0
    toplam=0
    for i in range(n):
        carpim=L[0][i]
        for j in range(n):
            if i!=j:
                carpim=carpim*(x-L[1][j])
        toplam=toplam+carpim
    return toplam

def funcLagrange(xi,yi,numberofmidpoints):
    # numberofmidpoints :  in x--o--o--x--o--o--x chain if x's are esxperimental points
    # numberofmidpoints is 2
    n=len(xi)
    nn=(n-1)*(numberofmidpoints+1)+1
    z=[[0 for j in range(nn)] for i in range(2)]
    Q=Lagrange(xi,yi)
    dx=0.0
    k=0
    i=0
    for i in range(n-1):
        z[0][k]=xi[i]
        z[1][k]=func(Q,z[0][k])
        k=k+1
        for j in range(numberofmidpoints):
            nx=float(numberofmidpoints+1.0)
            dx=(xi[i+1]-xi[i])/nx
            z[0][k]=z[0][k-1]+dx
            z[1][k]=func(Q,z[0][k])
            k=k+1
    z[0][k]=xi[n-1]
    z[1][k]=func(Q,z[0][k])
```

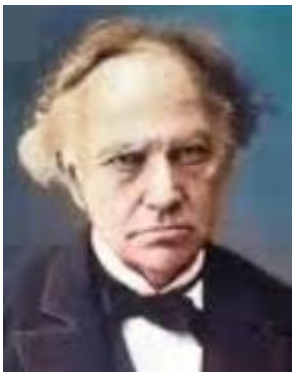
```
    return z
def read(name):
    path = name
    file1 = open(path,'r')
    file1.seek(0,2)  #jumps to the end
    endLocation=file1.tell()
    file1.seek(0) #jumps to the start
    nn=0
    while True:
        line=file1.readline()
        nn+=1
        if  file1.tell()==endLocation:
            break
    x=[0.0 for i in range(nn)]
    y=[0.0 for i in range(nn)]
    #z=[0.0 for i in range(nn)]
    file1.seek(0) #jumps to the start
    for i in range(nn):
        line=file1.readline()
        ee=line.split()
        x[i]=float(ee[0])
        y[i]=float(ee[1])
        #z[i]=float(ee[2])
    file1.close()
    a=[x,y]
    return a
    #return Transpose(a)

def Transpose(left):
    # transpose matrix (if A=a(i,j) Transpose(A)=a(j,i)
    # int i,j;
    n=len(left)
    m=len(left[0])
    b=[[0 for j in range(n)] for i in range(m)]
    for i in range(n):
        for j in range(m):
            b[j][i]=left[i][j]
    return b

a=read('input1.txt')
x=a[0]
y=a[1]
print("x=",x,"\ny=",y)
z=funcLagrange(x,y,8)
print("Lagrange interpolation\n",Transpose(z))
plt.title('Lagrange interpolation formula')
plt.xlabel('x ')
plt.ylabel('y ');
plt.plot(x,y,'bo',z[0],z[1],'k',linewidth=0.3,)
```

```
runfile('E:/okul/SCO1/Lagrange_interpolation.py', wdir='E:/okul/SCO1')
x= [1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0]
y= [1.0, 4.5, 9.8, 17.0, 26.0, 33.0, 27.0, 25.0]
Lagrange interpolation
 [[1.0, 1.0000000000000002], [1.1111111111111112, 1.70980181974836], [1.2222222222222223, 2.22388332435355],
[1.3333333333333335, 2.6107910379515307], [1.4444444444444446, 2.924346363106264], [1.5555555555555558,
3.2057979050251073], [1.666666666666667, 3.4857796067672626], [1.7777777777777781, 3.786082786654062],
[1.8888888888888893, 4.121250169089536], [2.0, 4.5], [2.111111111111111, 4.926488338101292], [2.2222222222222223,
5.4014176132021765], [2.3333333333333335, 5.922999542752632], [2.4444444444444446, 6.48778049784559], [2.555555555555556,
7.091337409880771], [2.666666666666667, 7.7288523090992305], [2.777777777777778, 8.395573586197193], [2.8888888888888893,
9.087172068227918], [3.0, 9.800000000000002], [3.111111111111111, 10.531261022180992], [3.2222222222222223,
11.279099237314735], [3.3333333333333335, 12.042615454961133], [3.4444444444444446, 12.821818707167038],
[3.555555555555556, 13.617521125476673], [3.666666666666667, 14.431184270690446], [3.777777777777778,
15.264725006580646], [3.8888888888888893, 16.120289008772588], [4.0, 17.0], [4.111111111111111, 17.905692802943104],
[4.222222222222221, 18.838638301858108], [4.333333333333332, 19.799268404206668], [4.444444444444443,
20.786909093494007], [4.555555555555554, 21.799529664524254], [4.666666666666664, 22.833516232281646],
[4.777777777777775, 23.883477605646174], [4.888888888888886, 24.94209161715241], [5.0, 26.0], [5.111111111111111,
27.04575990352436], [5.222222222222221, 28.06586013833666], [5.333333333333332, 29.044810242341093], [5.444444444444443,
29.96531045883842], [5.555555555555554, 30.80851071792435], [5.666666666666664, 31.554366712391392], [5.777777777777775,
32.18210115934265], [5.888888888888886, 32.67077833872641], [6.0, 32.99999999999999], [6.111111111111111,
33.15073072813142], [6.222222222222221, 33.106260860147735], [6.333333333333332, 32.8533150434385], [6.444444444444443,
32.38331452702287], [6.555555555555554, 31.693801276989266], [6.666666666666664, 30.790032007315975], [6.777777777777775,
29.686750217281386], [6.888888888888886, 28.410144326672444], [7.0, 27.0], [7.111111111111111, 25.512054750929813],
[7.222222222222221, 24.020562918137273], [7.333333333333332, 22.621079103795168], [7.444444444444443,
```

21.433468165902845], [7.555555555555554, 20.60514985566499], [7.666666666666664, 20.314586191129408], [7.777777777777775, 20.77501965829173], [7.888888888888886, 22.238470330876012], [8.0, 25.0]]



10.6.3 Hermit Interpolation Polynomial



Charles Hermite(1822-1901)

Hermit interpolation requires and uses derivative data in addition to the function data.

General formula

$H(x) = Q_{0,0}+ Q_{1,1}(x-x_0)+ Q_{2,2}(x-x_0)^2+ Q_{3,3}(x-x_0)^2(x-x_1)+ Q_{4,4}(x-x_0)^2(x-x_1)^2+\ldots+ Q_{2n+1,2n+1}(x-x_0)^2(x-x_1)^2\ldots(x-x_{n-1})^2(x-x_n)$

Calculation of the coefficients is explain at Algorithm 6.4-1 and Table 6.4-1.

Algorithm 6.4-1 Hermit interpolation

INPUT $x_0,x_1,\ldots,x_n$ değerleri; $f(x_0), f(x_1), \ldots,f(x_n)$ değerleri; $f'(x_0), f'(x_1), \ldots,f'(x_n)$ değerleri

OUTPUT : $Q_{0,0},Q_{1,1},\ldots, Q_{2n+1,2n+1}$ değerleri

$H(x)= Q_{0,0}+ Q_{1,1}(x-x_0)+ Q_{2,2}(x-x_0)^2+ Q_{3,3}(x-x_0)^2(x-x_1)+ Q_{4,4}(x-x_0)^2(x-x_1)^2+\ldots+ Q_{2n+1,2n+1}(x-x_0)^2(x-x_1)^2\ldots(x-x_{n-1})^2(x-x_n)$

Step 1: for i=0,1,…,n

{

Step 2: $z_{2i}=x_i$;

$z_{2i+1}=x_i$;

$Q_{2i,0}=f(x_i)$;

$Q_{2i+1,0}=f(x_i)$;

$Q_{2i+1,0}=f'(x_i)$;

Step 3: if i $\neq$ 0 then $Q_{2i,j} = \frac{Q_{2i,0}-Q_{2i-1,0}}{z_{2i}-z_{2i-1}}$

} End of Step 1 in sonu

Step 4: for i=2,3,…,2n+1

{ for j=2,3,…,i $Q_{i,j} = \frac{Q_{i,j-1}-Q_{i-1,j-1}}{z_i-z_{2i-j}}$ }

Table 6.4-1 Hermit interpolation difference formulation

Z	f(z)	First difference equation	Second difference equation
$Z_0 = x_0$	$f[z_0]=f(x_0)$	$f[z_0, z_1] = f(x_0)$	$f[z_0, z_1, z_2] = \frac{f[z_1, z_2] - f[z_0, z_1]}{z_2 - z_0}$
$Z_1 = x_0$	$f[z_1]=f(x_0)$	$f[z_1, z_2] = \frac{f[z_2] - f[z_1]}{z_2 - z_1}$	$f[z_1, z_2, z_3] = \frac{f[z_2, z_3] - f[z_1, z_2]}{z_2 - z_1}$
$Z_2 = x_1$	$f[z_2]=f(x_1)$	$f[z2, z3] = f(x1)$	$f[z_2, z_3, z_4] = \frac{f[z_3, z_4] - f[z_2, z_3]}{z_4 - z_2}$
$Z_3 = x_1$	$f[z_3]=f(x_1)$	$f[z_3, z_4] = \frac{f[z_4] - f[z_3]}{z_4 - z_3}$	$f[z_3, z_4, z_5] = \frac{f[z_4, z_5] - f[z_3, z_4]}{z_5 - z_4}$
$Z_4 = x_2$	$f[z_4]=f(x_2)$	$f[z4, z5] = f(x2)$	
$Z_5 = x_2$	$f[z_5]=f(x_2)$		

An example problem :

k	x_k	$f(x_k)$	$f'(x_k)$
0	1.3	0.6200860	-0.5220232
1	1.6	0.4554022	-05698959
2	1.9	0.2818186	-0.5811571

Difference table:

1.3	0.620086					
		-0.5220232				
1.3	0.620086		-0.08974267			
		-0.548946		0.06636556		
1.6	0.4554022		-0.069833		0.00266667	
		-0.5698959		0.06796556		-0.00277469
1.6	0.4554022		-0.02905367		0.00100185	
		-0.578612		0.06856667		
1.9	0.2818186		-0.00848367			
		-0.5811571				
1.9	0.2818186					

Interpolation value at x=1.5:

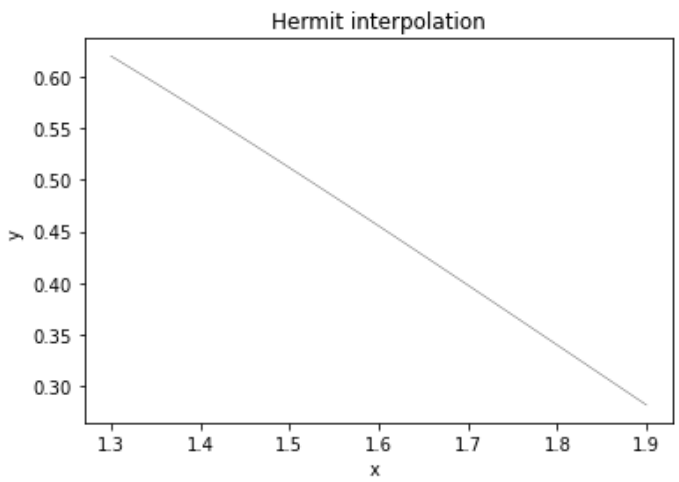
$H_5(1.5)$ =0.620086+(1.5-1.3)*(-0.5220232)+ (1.5-1.3)^2*(1.5-1.6)*(0.06636556+ (1.5-1.3)^2* (1.5-1.6)^2*(0.00266667)+ 1.5-1.3)^2* (1.5-1.6)^2*(1.5-1.9) (-0.00277469)=0.5118277

```
# Hermit interpolation
from math import *
from f_x import *
import gauss
import numpy as np
import matplotlib.pyplot as plt
import os

def Hermite(xi,fi,dfi):
    # m degree of polynomial
    m=len(xi)
    n=m-1
    Q=[[0.0 for i in range(2*n+2)] for j in range(2*n+2)]
    QQ=[0.0 for i in range(2*n+2)]
    z=[0.0 for i in range(2*n+2)]
    for i in range(n+1):
        ii=2*i
        iip1=2*i+1
```

```
        z[ii]=xi[i]
        z[iip1]=xi[i]
        Q[ii][0]=fi[i]
        Q[iip1][0]=fi[i]
        Q[iip1][1]=dfi[i]
        if i!=0:
            Q[2*i][1]=(Q[2*i][0]-Q[2*i-1][0])/(z[2*i]-z[2*i-1])
    for i in range(2,(2*n+2)):
        for j in range(2,i+1):
            Q[i][j]=(Q[i][j-1]-Q[i-1][j-1])/(z[i]-z[i-j])
    for i in range(2*n+2):
        QQ[i]=Q[i][i]
    return QQ

def func(QQ,xi,x):
    #Hermit polynomial
    m=len(QQ)-1
    xx=1.0
    toplam=QQ[0]
    for i in range(1,m,2):
        j=int((i-1)/2)
        xx=xx*(x-xi[j])
        toplam=toplam+QQ[i]*xx
        xx=xx*(x-xi[j])
        toplam=toplam+QQ[i+1]*xx
    return toplam

x=[1.3,1.6,1.9]
y=[0.620086,0.4554022,0.281818186]
dy=[-0.5220232,-0.5698959,-0.5811571]
QQ=Hermite(x,y,dy)
n=100
xmax=max(x)
xmin=min(x)
h=(xmax-xmin)/(n-1)
xx=[0.0 for i in range(n)]
yy=[0.0 for i in range(n)]
for i in range(n):
    xx[i]=xmin+h*i
    yy[i]=func(QQ,x,xx[i])
print("xx=",xx)
print("yy=",yy)
plt.title('Hermit interpolation')
plt.xlabel('x ')
plt.ylabel('y ');
plt.plot(xx,yy,'k',linewidth=0.3)
```

```
runfile('E:/okul/SCO1/Hermite_interpolation.py', wdir='E:/okul/SCO1')
Reloaded modules: f_x, gauss
xx= [1.3, 1.3060606060606061, 1.3121212121212122, 1.3181818181818181, 1.3242424242424242, 1.3303030303030303,
1.3363636363636364, 1.3424242424242425, 1.3484848484848486, 1.3545454545454545, 1.3606060606060606, 1.3666666666666667,
1.3727272727272728, 1.378787878787879, 1.3848484848484848, 1.3909090909090909, 1.396969696969697, 1.403030303030303,
1.4090909090909092, 1.415151515151515, 1.4212121212121211, 1.4272727272727272, 1.4333333333333333, 1.4393939393939394,
1.4454545454545455, 1.4515151515151514, 1.4575757575757575, 1.4636363636363636, 1.4696969696969697, 1.4757575757575758,
1.481818181818182, 1.4878787878787878, 1.493939393939394, 1.5, 1.506060606060606, 1.5121212121212122, 1.518181818181818,
1.5242424242424242, 1.5303030303030303, 1.5363636363636364, 1.5424242424242425, 1.5484848484848484, 1.5545454545454545,
1.5606060606060606, 1.5666666666666667, 1.5727272727272728, 1.5787878787878786, 1.5848484848484847, 1.5909090909090908,
1.596969696969697, 1.603030303030303, 1.6090909090909091, 1.6151515151515152, 1.621212121212121, 1.6272727272727272,
1.6333333333333333, 1.6393939393939394, 1.6454545454545455, 1.6515151515151514, 1.6575757575757575, 1.6636363636363636,
1.6696969696969697, 1.6757575757575758, 1.6818181818181817, 1.6878787878787878, 1.6939393939393939, 1.7,
1.706060606060606, 1.712121212121212, 1.718181818181818, 1.7242424242424241, 1.7303030303030302, 1.7363636363636363,
1.7424242424242422, 1.7484848484848485, 1.7545454545454544, 1.7606060606060605, 1.7666666666666666, 1.7727272727272727,
1.7787878787878788, 1.7848484848484847, 1.7909090909090908, 1.7969696969696969, 1.803030303030303, 1.809090909090909,
1.815151515151515, 1.821212121212121, 1.8272727272727272, 1.8333333333333333, 1.8393939393939394, 1.8454545454545452,
1.8515151515151516, 1.8575757575757574, 1.8636363636363635, 1.8696969696969696, 1.8757575757575755, 1.8818181818181818,
1.8878787878787877, 1.8939393939393938, 1.9]
yy= [0.620086, 0.6169182185924382, 0.6137424860292455, 0.6105588889550199, 0.6073675141002912, 0.6041684482815232,
0.6009617784011111, 0.5977475914473832, 0.5945259744946008, 0.5912970147029575, 0.5880607993185795, 0.5848174156735254,
0.5815669511857873, 0.578309493359289, 0.5750451297838881, 0.5717739481353733, 0.5684960361754673, 0.5652114817518247,
0.561920372798033, 0.5586227973336124, 0.5553188434640157, 0.5520085993806283, 0.5486921533607682, 0.5453695937676862,
0.5420410090505656, 0.5387064877445228, 0.5353661184706061, 0.5320199899357968, 0.528668190933009, 0.5253108103410893,
0.5219479371248169, 0.5185796603349039, 0.5152060691079948, 0.5118272526666667, 0.5084433003194296, 0.5050543014607259,
0.5016603455709311, 0.4982615222163525, 0.49485792104923093, 0.4914496318077394, 0.48803674431598354,
0.4846193484840022, 0.481197534307766, 0.47777139186917866, 0.47434101133607687, 0.4709064829622293,
0.46746789708733381, 0.46402534413703694, 0.4605789146228932, 0.45712869914240634, 0.4536747883790086,
```

0.450217273102065, 0.446756244166873, 0.44329179251466305, 0.43982400917259756, 0.43635298525377236, 0.4328788119572155, 0.42940158056788774, 0.4259213824566828, 0.42243830908042634, 0.41895245198187747, 0.4154639027897274, 0.4119727532186002, 0.4084790950690528, 0.4049830202275742, 0.4014846206665866, 0.3979839884444445, 0.39448121570543543, 0.3909763946797791, 0.3874696176836283, 0.38396097711906807, 0.3804505654741165, 0.37693847532272395, 0.37342479932477374, 0.3699096302260816, 0.36639306085839624, 0.36287518413939857, 0.3593560930727024, 0.35583588074785427, 0.3523146403403332, 0.34879246511155104, 0.34526944840885193, 0.34174568366551306, 0.33822126440074424, 0.33469628421968756, 0.33117083681341813, 0.32764501595894363, 0.32411891551920413, 0.32059262944307276, 0.31706625176535497, 0.3135398766067893, 0.31001359817404606, 0.3064875107597294, 0.302961708742375, 0.29943628658645194, 0.29591133884236165, 0.29238696014643806, 0.28886324522094836, 0.28534028887409163, 0.2818181860000001]



10.7 INTERPOLATION: SPLINE INTERPOLATION

10.7.1 CUBIC SPLINE INTERPOLATION POLYNOMIALS

Another way for the interpolation process is to fit a different polynomial in between each two point. If a third degree polinomial is considered:

$$r_k(x) = a_k(x - x_k)^3 + b_k(x - x_k)^2 + c(x - x_k)^3 + y_k \quad 1 \le k \le n \quad (10.6 - 7)$$

In the interpolation proses polinoms should be passing through all data points

$$r_k(x_{k+1}) = y_{k+1} \quad 1 \le k \le n \quad (10.6 - 8)$$

In the same time the first derivative of the polynomial should also be continious while passing from one polynomial to the next one at the data point

$$r'_{k-1}(x_k) = r'_k(x_k) \quad 1 \le k \le n \quad (10.6 - 9)$$

For the third degree polinomial second derivative of the polynomial should also be continious while passing from one polynomial to the next one at the data point

$$r''_{k-1}(x_k) = r''_k(x_k) \quad 1 \le k \le n \quad (10.6 - 10)$$

All these conditions are not enough to solve the coefficients of the polinomials. Two more conditions are required. This two additional conditions (A and B of the following equation) can be given by user

$r''_1(x_1) = A \qquad r''_{n-1}(x_n) = B$ (10.6-11)

They are the second derivatives at the both hand of the series of polinomials. If A and B values are taken equals to 0, it is called a **natural cubic spline**. Other end conditions such as the ones depends one the first derivatives can also be set to solve the system of equations.

Defining $h_k = x_{k+1} - x_k \quad 1 \le k \le n \quad (10.6 - 12)$

System of equations become:

$$a_k h_k^3 + b_k h_k^2 + b_k h_k = y_{k+1} - y_k \quad 1 \le k \le n \quad (10.6 - 13)$$
$$3a_{k-1}h_{k-1}^2 + 2b_{k-1}h_{k-1} + c_{k-1} - c_k = 0$$
$$6a_{k-1}h_{k-1} + 2b_{k-1} + 2b_k = 0$$
$$3b_0 = 0$$
$$6a_{n-1}h_{n-1} + 2b_{n-1} = 0$$

This set contains 3n-3 equations. This could a considerable load to the system of equation solving programs. To make calculation load simpler a special third degree polinomial can be considered. If our cubic polinomial is in the form of:

$$s_k(x) = a_k(x - x_k) + b_k(x_{k+1} - x) + [(x - x_k)^3 c_{k+1} + (x_{k+1} - x)^3 c_k] / (6h_k) \quad 1 \le k \le n \quad (10.6 - 14)$$

then derivative equations becomes

$$s'_k(x) = a_k - b_k + [(x - x_k)^2 c_{k+1} - (x_{k+1} - x)^2 c_k] / h_k \quad 1 \le k \le n \quad (10.6\text{-}15)$$

$$s''_k(x) = [(x - x_k) c_{k+1} - (x_{k+1} - x) c_k] / h_k \quad 1 \le k \le n$$

a_k ve b_k coefficients can be expressed as a function of c_k

$$b_k = \frac{[6y_k - h_k c_k]}{6h_k} \qquad 1 \le k \le n \qquad (10.6 - 16)$$

$$a_k = \frac{[6y_{k+1} - h_k^2 c_{k+1}]}{6h_k} \qquad 1 \le k \le n \qquad (10.6 - 17)$$

In this case only c_k terms left in the system of equations to be solved.

$$h_{k-1} c_{k-1} + 2(h_{k-1} - h_k) c_{k-1} + h_k c_{k+1} c_{k+1} = 6\left[\frac{y_{k+1} - y_k}{h_k} - \frac{y_k - y_{k-1}}{h_{k-1}}\right] \qquad 1 \le k \le n \qquad (10.6 - 18)$$

This system of equation has only n-2 terms to be solved. By making definition

$w_k = \frac{y_{k+1} - y_k}{h_k}$, $1 \le k \le n$ (10.6-19)

System of equation becomes

$$\begin{bmatrix} 1 & & & & & & \\ h_1 & 2(h_1 + h_2) & h_2 & & & & \\ & h_2 & 2(h_2 + h_3) & & & & \\ & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ & & & \cdots & 2(h_{n-3} + h_{n-2}) & h_{n-2} & \\ & & & & h_{n-2} & 2(h_{n-2} + h_{n-1}) & h_{n-1} \\ & & & & & & 1 \end{bmatrix} \begin{Bmatrix} c_0 \\ c_1 \\ c_2 \\ \cdots \\ c_{n-2} \\ c_{n-1} \\ c_n \end{Bmatrix} = \begin{Bmatrix} A \\ 6(w_2 - w_1) \\ 6(w_3 - w_2) \\ \cdots \\ 6(w_{n-2} - w_{n-3}) \\ 6(w_{n-1} - w_{n-2}) \\ B \end{Bmatrix} \quad (10.6 - 20)$$

Where A and B are the second derivative end conditions. A and B should be defined by user. Another important property of the above matrix is that it is a band matrix, therefore less amount of calculation is required to solve it (by using band matrix algorithms such as Thomas algorithm) compare to gauss elimination type of solution methods.

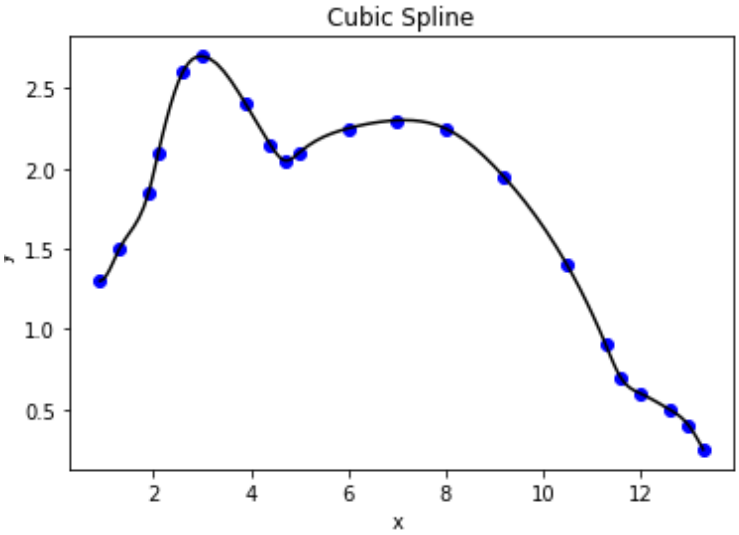
Cubic spline curve fitting (python version)

```
# -*- coding: utf-8 -*-
"""
Created on Sun Jun  3 21:38:38 2018

@author: Mustafa Turhan Çoban
"""
from math import *
import numpy as np;
import matplotlib.pyplot as plt;

def thomas(f,e,g,r):
    n=len(f);
    x=[0.0 for i in range(n)];
    for k in range(0,n): e[k]=e[k]/f[k-1];f[k]=f[k]-e[k]*g[k-1];
    for k in range(1,n): r[k]=r[k]-e[k]*r[k-1];
    x[n-1]=r[n-1]/f[n-1];
    for k in range(n-2,-1,-1):  x[k]=(r[k]-g[k]*x[k+1])/f[k];
    return x;

def cubic_spline(xi,yi,c0,cn):
    n=len(xi);
    h=[0.0 for i in range(n)];
    w=[0.0 for i in range(n)];
    f=[0.0 for i in range(n)];
    e=[0.0 for i in range(n)];
    g=[0.0 for i in range(n)];
    d=[0.0 for i in range(n)];
    x=[0.0 for i in range(n)];
    S= [[0 for x in range(n)] for y in range(4)];
    for k in range(0,n-1):
        h[k]=xi[k+1]-xi[k];
        w[k]=(yi[k+1]-yi[k])/h[k];
    d[0]=c0;
```

```
    d[n-1]=cn;
    for k in range(0,n-1):
        d[k]=6.0*(w[k]-w[k-1]);
    f[0]=1.0;
    f[n-1]=1.0;
    g[0]=0.0;
    g[n-1]=0.0;
    e[0]=0.0;
    e[n-1]=0.0;
    for k in range(0,n-1):
         f[k]=2.0*(h[k]+h[k-1]);e[k]=h[k-1];g[k]=h[k];
    S[2]=thomas(f,e,g,d);
    S[3]=xi;
    for k in range(0,n-1):
        S[0][k]=(6.*yi[k+1]-h[k]*h[k]*S[2][k+1])/(6.0*h[k]);
        S[1][k]=(6.*yi[k]-h[k]*h[k]*S[2][k])/(6.0*h[k]);
    return S;

def funcSpline(S,x):
    n=len(S[0]);
    xx1=0;
    xx2=0;
    y=0;
    hk=0;
    for k in range(0,n-1):
        if S[3][k]<=x and x<=S[3][k+1]:
            hk=(S[3][k+1]-S[3][k]);
            xx1=(x-S[3][k]);
            xx2=(S[3][k+1]-x);
            y=S[0][k]*xx1+S[1][k]*xx2+(xx1*xx1*xx1*S[2][k+1]+xx2*xx2*xx2*S[2][k])/(6.0*hk);
            break;
    if y==0 and S[3][n-2]<=x:
         k=n-2;
         hk=(S[3][k+1]-S[3][k]);
         xx1=(x-S[3][k]);
         xx2=(S[3][k+1]-x);
         y=S[0][k]*xx1+S[1][k]*xx2+(xx1*xx1*xx1*S[2][k+1]+xx2*xx2*xx2*S[2][k])/(6.0*hk);
    return y;

def listSpline(xi,yi,numberofmidpoints,c0,cn):
    #numberofmidpoints :  in x--o--o--x--o--o--x chain if x's are esxperimental points
                #numberofmidpoints is 2
    n=len(xi);
    nn=(n-1)*(numberofmidpoints+1)+1;
    z=[[0 for x in range(nn)] for y in range(2)];
    S= [[0 for x in range(nn)] for y in range(4)];
    S=cubic_spline(xi,yi,c0,cn);
    dx=0;
    k=0;
    for i in range(0,n-1):
        z[0][k]=xi[i];z[1][k]=funcSpline(S,z[0][k]);k=k+1;
        for j in range(numberofmidpoints):
            dx=(xi[i+1]-xi[i])/(numberofmidpoints+1.0);
            z[0][k]=z[0][k-1]+dx;z[1][k]=funcSpline(S,z[0][k]);k=k+1;
    z[0][k]=xi[i+1];z[1][k]=funcSpline(S,z[0][k]);
    return z;

x=[0.9,1.3,1.9,2.1,2.6,3.0,3.9,4.4,4.7,5.0,6.0,7.0,8.0,9.2,10.5,11.3,11.6,12.0,12.6,13.0,13.3];
y=[1.3,1.5,1.85,2.1,2.6,2.7,2.4,2.15,2.05,2.1,2.25,2.3,2.25,1.95,1.4,0.9,0.7,0.6,0.5,0.4,0.25];
Z=listSpline(x,y,5,0,0);
print("Z=",Z);
plt.title('Cubic Spline')
plt.xlabel('x ')
plt.ylabel('y ');
plt.plot(x,y,'bo',Z[0],Z[1],'k');
```

```
 runfile('D:/okul/SCO1/cubic_spline.py', wdir='D:/okul/SCO1')
Reloaded modules: gauss
Z= [[0.9, 0.9666666666666667, 1.0333333333333334, 1.1, 1.1666666666666667, 1.2333333333333334, 1.3, 1.4, 1.5, 1.6,
1.7000000000000002, 1.8000000000000003, 1.9, 1.9333333333333333, 1.9666666666666668, 2.0, 2.033333333333333,
2.0666666666666664, 2.1, 2.1833333333333336, 2.266666666666667, 2.3500000000000005, 2.433333333333334,
2.5166666666666675, 2.6, 2.666666666666667, 2.7333333333333334, 2.8, 2.8666666666666663, 2.9333333333333327, 3.0, 3.15, 3.3,
3.4499999999999997, 3.5999999999999996, 3.7499999999999996, 3.9, 3.9833333333333334, 4.066666666666666,
4.1499999999999995, 4.2333333333333325, 4.3166666666666655, 4.4, 4.45, 4.5, 4.55, 4.6, 4.6499999999999995, 4.7, 4.75, 4.8, 4.85,
4.8999999999999995, 4.949999999999999, 5.0, 5.166666666666667, 5.333333333333334, 5.500000000000001, 5.666666666666668,
```

5.833333333333335, 6.0, 6.166666666666667, 6.333333333333334, 6.500000000000001, 6.666666666666668, 6.833333333333335, 7.0, 7.166666666666667, 7.333333333333334, 7.500000000000001, 7.666666666666668, 7.833333333333335, 8.0, 8.2, 8.399999999999999, 8.599999999999998, 8.799999999999997, 8.999999999999996, 9.2, 9.416666666666666, 9.633333333333333, 9.85, 10.066666666666666, 10.283333333333333, 10.5, 10.633333333333333, 10.766666666666666, 10.899999999999999, 11.033333333333331, 11.166666666666664, 11.3, 11.350000000000001, 11.400000000000002, 11.450000000000003, 11.500000000000004, 11.550000000000004, 11.6, 11.666666666666666, 11.733333333333333, 11.799999999999999, 11.866666666666665, 11.933333333333332, 12.0, 12.1, 12.2, 12.299999999999999, 12.399999999999999, 12.499999999999998, 12.6, 12.666666666666666, 12.733333333333333, 12.799999999999999, 12.866666666666665, 12.933333333333332, 13.0, 13.05, 13.100000000000001, 13.150000000000002, 13.200000000000003, 13.250000000000004, 13.3], [1.2999999999999998, 1.3093751894619083, 1.3344450507225516, 1.3706260230943061, 1.413334545889548, 1.4579870584206536, 1.4999999999999996, 1.551964750663964, 1.5957343748206128, 1.6393657921512435, 1.690915922337154, 1.7584416850596403, 1.8500000000000003, 1.887079217060214, 1.926982280840211, 1.9689675411820904, 2.0122933479279497, 2.0562180509198864, 2.1, 2.205651779159039, 2.3043189530703647, 2.394544441922051, 2.4748711659021705, 2.543842045198796, 2.6, 2.6349754723676666, 2.661593547897045, 2.680629632506552, 2.6928591321146036, 2.699057452639615, 2.7000000000000006, 2.6860390570520196, 2.652938927647158, 2.6042422845124706, 2.5434918003750124, 2.4742301479618374, 2.4, 2.357861812873503, 2.315552989956062, 2.2733429642288336, 2.231501168672976, 2.190297036269646, 2.1499999999999995, 2.1265238713656305, 2.1044571079150636, 2.084785524350757 6, 2.0684949353751727, 2.056571155690767, 2.05, 2.0494053210004255, 2.0539631233699773, 2.062487449781682, 2.073792342908569, 2.086691845423666, 2.1, 2.1396051286242534, 2.1717712493844465, 2.1976960795913483, 2.2185773365557284, 2.235612737588356, 2.25, 2.262715615023503, 2.2738511695799972, 2.2832770245126888, 2.2908635406647826, 2.2964810788794843, 2.3, 2.301199077948401, 2.29949073896223, 2.2941958223578958, 2.284635167451807, 2.270129613560372, 2.25, 2.217799810228662, 2.1773615745280437, 2.129523884276508, 2.0751253308524147, 2.0150045056341255, 1.95, 1.8747328100503609, 1.7939415032693384, 1.7068493242653617, 1.61267951764686, 1.510655328022263, 1.4, 1.3272597895357943, 1.2506677564782636, 1.169934830911663, 1.0847719429202491, 0.9948900225882772, 0.9000000000000001, 0.8632203068251879, 0.8266057525736611, 0.7910663910053992, 0.7575122758803815, 0.7268534609585878, 0.6999999999999998, 0.6710795558939062, 0.6490500148136773, 0.632419332721743, 0.6196954655805323, 0.6093863693524749, 0.6, 0.585262369387163, 0.5694620676711136, 0.5528173873631121, 0.5355466209744191, 0.517868061016295, 0.5, 0.48794696771124646, 0.47511033878706393, 0.46069372039065026, 0.4439007196852034, 0.42393494383392094, 0.4000000000000001, 0.3790954038292061, 0.3559569510242995, 0.33103141291210286, 0.30476556081943873, 0.27760616607312966, 0.25]]



10.7.2 CUBIC B-SPLINE INTERPOLATION

B-spline interpolation will be investigated in this section. In general B-spline definition spline functions are defined in the region from minus infinity to plus infinity. We will specify cubic B-spline here for given $k_0,\ldots,k_N$ knots general curve fitting functions will be defined as

$$S(x) = \sum_{i=-3}^{N} a_i B_i(x) \quad (6.7-1)$$

$B_i(x)$ in the equation is called B-spline function. B-spline function for k_i, i=-3,-2,0,…,N,N+1,N+2 knots for the x value in the [a,b] interval , and with the limit values k_0=a, k_N=b can have the following property

$\sum_{i=-3}^{N} B_i(x) = 1 \quad (6.7-2)$

Interpolation equation S(x) has N+3 coefficient. There are N+1 data in the [a,b] data range so that additional equations will be needed to solve the data set. In order to define B-spline function B(x) a recursive calculation process will be defined. We first define B-splines that may be piecewise constant(n=1), linear(n=2), quadratic (n=3)and cubic (n=4). The condition that the sum of all B-spline function in all regions are still holds.

$\sum_{i=-3}^{N+3} B_i(x) = 1$ (n=1,2,3,4) (10.6-21)

The B-splines of order n are related to those of order n-1 by the recurrence relation

$B_{i,n}(x) = \frac{(x-k_i)}{k_{i+n-1}-k_i} B_{i,n-1}(x) + \frac{(k_{i+n}-x)}{k_{i+n}-k_{i+1}} B_{i+1,n-1}(x)$ (10.6-22)

Where i=-3,…N-1 range and n=(1,2,3,4) .In computing with this relation, B-spline of order one will be taken as a starting point. Therefore first order B spline should be independently defined.

$$B_{i,1}(x) = \begin{Bmatrix} 0 \\ 1 \\ 0 \end{Bmatrix} \begin{matrix} x < k_i \\ k_i \le x < k_{i+1} \\ k_{i+1} \le x \end{matrix} \quad i=-3,-2,\ldots,N+3 \quad (10.6\text{-}23)$$

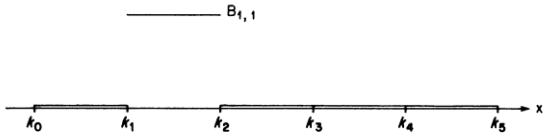


Figure 10.6-1 first order(constant) B-spline function

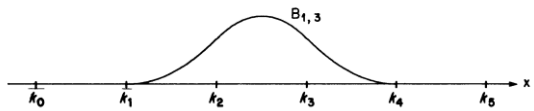


Figure 10.6-2 3th order(quadratic) B-spline function

By using first order equation and recurrence relation, linear, quadratic and cubic B spline functions can be calculated. First order(constant) and third order(quadratic) B-spline functions are shown in Figure 10.6-1 and 10.6-2 .

Now that B-spline functions are defined, we can return to solve coefficients of the iterations equation. By suppliying given k_j and f_j data points into the quation N+1 equations will be formed to solve

$\sum_{i=-3}^{N-1} a_i B_i(k_j) = f_j$ j=0,1,…,N (10.6-24)

In the jth equation only terms with B_{j-3}, B_{j-2}, B_{j-1} are different than zero the rest will be zero. In order to solve the data set two more equation will be needed. In order to established this, second derivative is at both end of the B-spline equation will be defined. If Natural cubic spline conditions are applied, this boundary values will be zero. Of course the problem can be solve by using any other type of boundary conditions.

$S''(k_0) = 0$

$S''(k_N) = 0$ (10.6-25)

When this condition applied to the curve fitting equations, they become

$a_{-3}B''_{-3}(k_0) + a_{-2}B''_{-2}(k_0) + a_{-1}B''_{-1}(k_0) = 0$

$a_{N-3}B''_{N-3}(k_N) + a_{N-2}B''_{N-2}(k_N) + a_{N-1}B''_{N-1}(k_N) = 0$ (10.6.26)

Second derivatives in the equation defined as:

$B''_{-3}(k_0) = \frac{6}{(k_1-k_{-1})(k_1-k_{-2})}$ (10.6.27)

$$B''_{-2}(k_0) = \frac{-6}{(k_1-k_{-1})}\left\{\frac{1}{(k_2-k_{-1})} + \frac{1}{(k_1-k_{-2})}\right\} \quad (10.6.28)$$

$$B''_{-1}(k_0) = \frac{6}{(k_1-k_{-1})(k_2-k_{-1})} \quad (10.6.29)$$

$$B''_{N-3}(k_0) = \frac{6}{(k_{N+1}-k_{N-1})(k_{N+1}-k_{N-2})} \quad (10.6-30)$$

$$B''_{N-2}(k_N) = \frac{-6}{(k_{N+1}-k_{N-1})}\left\{\frac{1}{(k_{N+2}-k_{N-1})} + \frac{1}{(k_{N+1}-k_{N-2})}\right\} \quad (10.6-31)$$

$$B''_{N-1}(k_0) = \frac{6}{(k_{N+1}-k_{N-1})(k_{N+2}-k_{N-1})} \quad (10.6-32)$$

It should be noted that k_{-3}, k_{-2}, k_{-1}, k_{N+1}, k_{N+2}, k_{N+3} values are not defined in the data set. This values can be set arbitrarily, but should satisfy $k_{-3}<k_{-2}<k_{-1}<k_0$ and $k_N<k_{N+1}<k_{N+2}<k_{N+3}$ conditions.

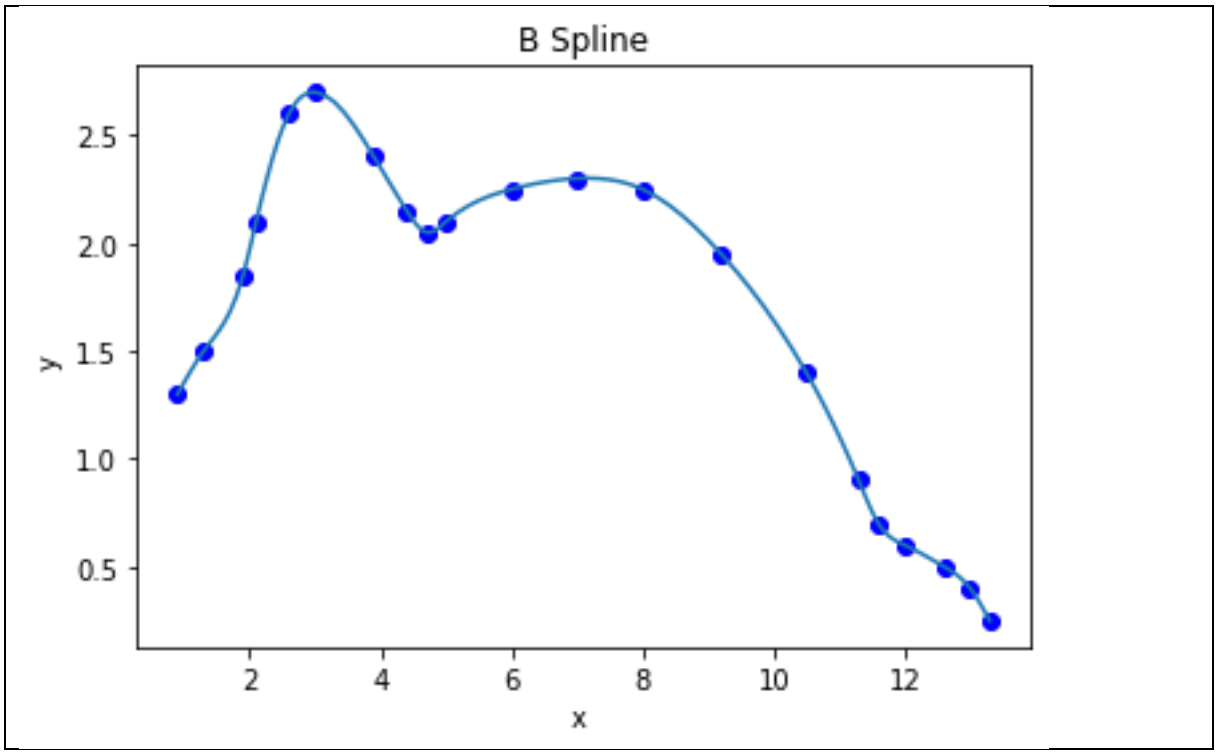
B-spline curve fitting python version

```
# -*- coding: utf-8 -*-
"""
Created on Tue Sep  4 09:52:03 2018
```

```
@author: Mustafa Turhan Çoban
"""
from gauss import *;
from math import *
import numpy as np;
import matplotlib.pyplot as plt;

class B_spline():

    def __init__(self,xi,yi):
      self.x=xi;
      self.f=yi;
      self.N=len(yi)-1;
      self.n1=self.N+7;
      self.x=[0.0 for i in range(self.N+1)];
      self.f=[0.0 for i in range(self.N+1)];
      self.k=[0.0 for i in range(self.n1)];
      self.m=5;
      for i in range(self.N+1):
          self.x[i]=xi[i];
          self.f[i]=yi[i];
          self.k[i+3]=self.x[i];
      dx=self.x[self.N]-self.x[0];
      self.k[0]=self.x[0]-6.0*dx;
      self.k[1]=self.x[0]-4.0*dx;
      self.k[2]=self.x[0]-2.0*dx;
      self.k[self.N+4]=self.x[self.N]+2.0*dx;
      self.k[self.N+5]=self.x[self.N]+4.0*dx;
      self.k[self.N+6]=self.x[self.N]+6.0*dx;
      self.B=[0.0 for i in range(self.n1)];
      self.B2=[0.0 for i in range(self.n1)];
      self.A();

    def A(self):
      N=self.N;
      self.B2[0]=6.0/(self.k[4]-self.k[2])/(self.k[4]-self.k[1]);
      self.B2[1]=-6.0/(self.k[4]-self.k[2])*(1.0/((self.k[5]-self.k[2])+1.0/(self.k[4]-self.k[1]) ));
      self.B2[2]=6.0/(self.k[4]-self.k[2])/(self.k[5]-self.k[2]);
      self.B2[self.N]=6.0/(self.k[N+4]-self.k[N+2])/(self.k[N+4]-self.k[N+1]);
      ss1=6.0/(self. k[N+4]-self.k[N+2]);
      ss2=(1.0/((self.k[N+5]-self.k[N+2])+1.0/(self.k[N+4]-self.k[N+1])));
      self.B2[N+1]=ss1*ss2;
      self.B2[N+2]=6.0/(self.k[N+4]-self.k[N+2])/(self.k[N+5]-self.k[N+2]);
      #print("B2=",self.B2);
      AA = [[0 for i in range(N+3)] for j in range(N+3)];
      BB=[0 for x in range(N+3)];
      AA[0][0]=self.B2[0];
      AA[0][1]=self.B2[1];
      AA[0][2]=self.B2[2];
      BB[0]=0;
      AA[N+2][N]=self.B2[N];
      AA[N+2][N+1]=self.B2[N+1];
      AA[N+2][N+2]=self.B2[N+2];
      BB[N+2]=0;
      for j in range(N+1):
          self.B=self.B1(self.k[j+3]);
          #print("self.B\n",self.B);
          for i in range(N+3):
              AA[j+1][i]=self.B[i];
          BB[j+1]=self.f[j];
      #print("AA\n",AA);
      #print("BB\n",BB);
      self.a=gauss(AA,BB);

    def B1(self,t):
        b=[[0.0 for i in range(self.N+7)] for i in range(self.m)];
        B1=[0.0 for i in range(self.N+7)];
        km1=len(self.k)-1;
        km4=len(self.k)-4;
        for i in range(km1):
            if t>=self.k[i] and t<self.k[i+1]: b[1][i]=1;
            else: b[1][i]=0.0;
        for n in range(2,self.m):
              for i in range(km4):
                  b[n][i]=(t-self.k[i])/(self.k[i+n-1]-self.k[i])*b[n-1][i]+(self.k[i+n]-t)/(self.k[i+n]-self.k[i+1])*b[n-1][i+1];
                  B1[i]=b[n][i];
```

```
        return B1;

    def S(self,x):
        self.B=self.B1(x);
        ss=0;
        for i in range(self.N+2):
            ss+=self.a[i]*self.B[i];
        return ss;

    def  funcB_Spline(self,numberofmidpoints):
        n=len(self.x);dx=0.0;k=0.0;
        nn=(n-1)*(numberofmidpoints+1)+1;
        z=[[0 for i in range(nn)] for j in range(2)]
        kk=0;
        for i in range(n-1):
            z[0][kk]=self.x[i];
            z[1][kk]=self.S(z[0][kk]);
            kk=kk+1;
            for j in range(numberofmidpoints):
                dx=(self.x[i+1]-self.x[i])/(numberofmidpoints+1.0);
                z[0][kk]=z[0][kk-1]+dx;
                z[1][kk]=self.S(z[0][kk]);
                kk=kk+1;
            z[0][kk]=self.x[i+1];z[1][kk]=self.S(z[0][kk]);
        return z;
#a=[[0.0,  0.6 ,1.5 ,1.7 ,1.9 ,2.1,2.3 ,2.6 ,2.8,3.0 ,3.6 ,4.7  ,5.2  ,5.7  ,5.8  , 6.0 , 6.4 ,6.9,7.6,8.0],
 # [-0.8,-0.34,0.59,0.59,0.23,0.1,0.28,1.03,1.5,1.44,0.74,-0.82,-1.27,-0.92,-0.92,-1.04,-0.79,-0.06,1.0,0.0]];
x=[0.9,1.3,1.9,2.1,2.6,3.0,3.9,4.4,4.7,5.0,6.0,7.0,8.0,9.2,10.5,11.3,11.6,12.0,12.6,13.0,13.3];
y=[1.3,1.5,1.85,2.1,2.6,2.7,2.4,2.15,2.05,2.1,2.25,2.3,2.25,1.95,1.4,0.9,0.7,0.6,0.5,0.4,0.25];
b1=B_spline(x,y);
aa=b1.funcB_Spline(5);
print("aa=",aa);
plt.title('B Spline')
plt.xlabel('x ')
plt.ylabel('y ');
plt.plot(x,y,'bo',aa[0],aa[1]);
```

```
 runfile('E:/okul/SCO1/B_spline.py', wdir='E:/okul/SCO1')
Reloaded modules: gauss
aa= [[0.9, 0.9666666666666667, 1.0333333333333334, 1.1, 1.1666666666666667, 1.2333333333333334, 1.3, 1.4, 1.5, 1.6,
1.7000000000000002, 1.8000000000000003, 1.9, 1.9333333333333333, 1.9666666666666668, 2.0, 2.033333333333333,
2.0666666666666664, 2.1, 2.1833333333333336, 2.266666666666667, 2.3500000000000005, 2.433333333333334,
2.5166666666666675, 2.6, 2.666666666666667, 2.7333333333333334, 2.8, 2.8666666666666663, 2.9333333333333327, 3.0, 3.15, 3.3,
3.4499999999999997, 3.5999999999999996, 3.7499999999999996, 3.9, 3.9833333333333334, 4.066666666666666,
4.1499999999999995, 4.2333333333333325, 4.3166666666666655, 4.4, 4.45, 4.5, 4.55, 4.6, 4.6499999999999995, 4.7, 4.75, 4.8, 4.85,
4.8999999999999995, 4.949999999999999, 5.0, 5.166666666666667, 5.333333333333334, 5.500000000000001, 5.666666666666668,
5.833333333333335, 6.0, 6.166666666666667, 6.333333333333334, 6.500000000000001, 6.666666666666668, 6.833333333333335,
7.0, 7.166666666666667, 7.333333333333334, 7.500000000000001, 7.666666666666668, 7.833333333333335, 8.0, 8.2,
8.399999999999999, 8.599999999999998, 8.799999999999997, 8.999999999999996, 9.2, 9.416666666666666, 9.633333333333333,
9.85, 10.066666666666666, 10.283333333333333, 10.5, 10.633333333333333, 10.766666666666666, 10.899999999999999,
11.033333333333331, 11.166666666666664, 11.3, 11.350000000000001, 11.400000000000002, 11.450000000000003,
11.500000000000004, 11.550000000000004, 11.6, 11.666666666666666, 11.733333333333333, 11.799999999999999,
11.866666666666665, 11.933333333333332, 12.0, 12.1, 12.2, 12.299999999999999, 12.399999999999999, 12.499999999999998, 12.6,
12.666666666666666, 12.733333333333333, 12.799999999999999, 12.866666666666665, 12.933333333333332, 13.0, 13.05,
13.100000000000001, 13.150000000000002, 13.200000000000003, 13.250000000000004, 13.3], [1.3000000000000003,
1.3358156165256743, 1.3712432354405089, 1.4058291699949308, 1.4391197334393684, 1.4706612390242488, 1.5,
1.540088948947137, 1.5798900948992916, 1.6250926054185597, 1.6813856480670368, 1.7544583904068183, 1.8500000000000003,
1.887678938040104, 1.9278292984362215, 1.9697938330093108, 2.012915293580331, 2.0565364319702413, 2.1,
2.205109523779884, 2.3035759201289565, 2.39384867748636, 2.474377284291239, 2.543611228982738, 2.6, 2.635084716783867,
2.661747345804522, 2.6807790378492156, 2.6929709437051943, 2.699114214159707, 2.7, 2.685954101706054,
2.6528241774059054, 2.6041370160923893, 2.5434194067583427, 2.474198138396601, 2.3999999999999995, 2.3578708610093897,
2.3155652567730742, 2.273354277922516, 2.2315090150891765, 2.190300558904517, 2.15, 2.126522741303414,
2.1044555473252013, 2.084784047135558, 2.0684938698046795, 2.056570644402762, 2.0500000000000003, 2.0494056417661426,
2.053963579269148, 2.062487898176526, 2.073792684155787, 2.086692022874441, 2.1000000000000005, 2.1396047137058978,
2.1717706769632557, 2.1976955384967094, 2.2185769470308943, 2.235612551290446, 2.25, 2.262715730058789,
2.273851331064437, 2.283277180789162, 2.290863657005184, 2.296481137484723, 2.2999999999999994, 2.3011990327256107,
2.2994906654456644, 2.2941957383466445, 2.2846350916150366, 2.270129565437327, 2.25, 2.2177998913836503,
2.1773617404356407, 2.1295241114896206, 2.0751255688792405, 2.0150046769381507, 1.95, 1.8747325006908233,
1.7939408493103775, 1.7068484097048677, 1.6126785457204977, 1.5106546212034742, 1.4000000000000001, 1.327260456846939,
1.2506690980931496, 1.1699366483850964, 1.0847738323692446, 0.994891374692058, 0.9000000000000001, 0.8632195806034588,
0.826604342645292, 0.7910645258656811, 0.7575103700048077, 0.7268521148028526, 0.7000000000000001, 0.6710826113076073,
0.6490565897071484, 0.6324286264873907, 0.6197054129371019, 0.6093936403450493, 0.6000000000000001, 0.5852423077385505,
0.5694178501250552, 0.5527540088519833, 0.5354781656118041, 0.5178177020969867, 0.5, 0.48799968460986426,
0.47521723740985977, 0.4608393518744288, 0.44405272147801383, 0.42404403969505705, 0.4, 0.37896698399557704,
0.3556883131752974, 0.3306457064694869, 0.30432088280847097, 0.2771955611225754, 0.24975146034212778]]
```



10.8 VANDERMONDE INTERPOLATION

Nicolaas van der Monde, Dutch

If the interpolating polynomial is written as

$$\boldsymbol{y = c_0 + c_1 x + c_2 x^2 + \cdots + c_N x^N} \quad \textbf{(6.5-1)}$$

then the c_i's are required to satisfy the linear equation

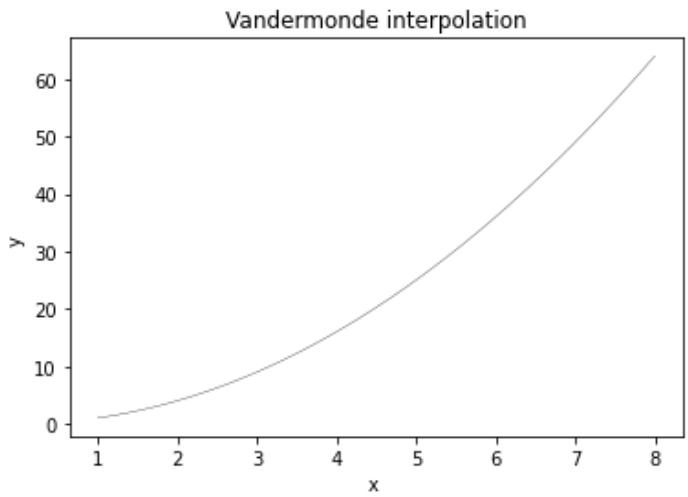
$$\begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^N \\ 1 & x_1 & x_1^2 & \cdots & x_1^N \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & x_{N-1} & x_{N-1}^2 & \cdots & x_{N-1}^2 \\ 1 & x_N & x_N^2 & \cdots & x_N^2 \end{bmatrix} \begin{Bmatrix} c_0 \\ c_1 \\ \cdots \\ c_{N-1} \\ c_N \end{Bmatrix} = \begin{Bmatrix} y_0 \\ y_1 \\ \cdots \\ y_{N-1} \\ y_N \end{Bmatrix} \quad (6.5\text{-}2)$$

This is a *Vandermonde matrix.* The matrix can be solved by using some of the Matrix solving method. In The next program, it is solved by using pivoted Gauss eliminaton method. Remember that Vandermonde systems can be quite ill-conditioned. In such a case, *no* numerical method is going to give a very accurate answer. So the method should be used for relatively small number of data points. In eigenvalues sectionVandermonde will be solved by using singularvalue decomposition (SVD) method. In that case it will be as efficient as least square method.

```
#Vandermonde interpolation
from math import *
from f_x import *
import gauss
import numpy as np
import matplotlib.pyplot as plt
import os

def Vandermonde_coef(xi,yi):
    n=len(xi)
    A=[[0.0 for i in range(n)] for j in range(n)]
    B=[0.0 for i in range(n)]
    X=[0.0 for i in range(n)]
    cc=1.0
    for i in range(n):
        B[i]=yi[i]
        cc=1.0
        for j in range(n):
            A[i][j]=cc;
            cc=cc*xi[i]
    X=gauss.gauss(A,B)
    return X;

def func(c,x):
    n=len(c)
    ff=0.0
    xx=1.0
    if n!=0.0:
        xx=1.0
        for i in range(n):
            ff=ff+c[i]*xx
            xx=xx*x
    return ff

def read_array2D(d):
    path = d
    file1 = open(path,'r')
    file1.seek(0,2)  #jumps to the end
    endLocation=file1.tell()
    file1.seek(0) #jumps to the start
    nn=0
    m=0
    while True:
        line=file1.readline()
        ee=line.split()
        nn+=1
        if  file1.tell()==endLocation:
            break
    m=len(ee)
    a=[[0.0 for i in range(nn)] for j in range(m)]
    file1.seek(0) #jumps to the start
    for i in range(nn):
        line=file1.readline()
        ee=line.split()
        for j in range(m):
            a[j][i]=float(ee[j])
    file1.close()
    return a

a=read_array2D('a.txt')
x=a[0];
print("x=",x)
xmin=min(x)
xmax=max(x)
y=a[1];
print("y=",y)
c=Vandermonde_coef(x,y)
print("c=",c)
h=0.1
n=100
h=(xmax-xmin)/(n-1)
xx=[0.0 for i in range(n)]
yy=[0.0 for i in range(n)]
for i in range(n):
```

```
    xx[i]=xmin+h*i
    yy[i]=func(c,xx[i])
print("xx=",xx)
print("yy=",yy)
plt.title('Vandermonde interpolation')
plt.xlabel('x ')
plt.ylabel('y ');
plt.plot(xx,yy,'k',linewidth=0.3)
```

```
runfile('E:/okul/SCO1/Vandermonde.py', wdir='E:/okul/SCO1')
Reloaded modules: f_x, gauss
x= [1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0]
y= [1.0, 4.0, 9.0, 16.0, 25.0, 36.0, 49.0, 64.0]
c= [0.0, 0.0, 1.0, -0.0, -0.0, -0.0, 0.0, -0.0]
xx= [1.0, 1.0707070707070707, 1.1414141414141414, 1.2121212121212122, 1.2828282828282829, 1.3535353535353536,
1.4242424242424243, 1.4949494949494948, 1.5656565656565657, 1.6363636363636362, 1.7070707070707072, 1.7777777777777777,
1.8484848484848484, 1.9191919191919191, 1.9898989898989898, 2.0606060606060606, 2.1313131313131315, 2.202020202020202,
2.2727272727272725, 2.3434343434343434, 2.4141414141414144, 2.484848484848485, 2.5555555555555554, 2.6262626262626263,
2.696969696969697, 2.7676767676767673, 2.8383838383838382, 2.909090909090909, 2.9797979797979797, 3.0505050505050506,
3.121212121212121, 3.191919191919192, 3.2626262626262625, 3.333333333333333, 3.404040404040404, 3.4747474747474745,
3.5454545454545454, 3.616161616161616, 3.686868686868687, 3.7575757575757573, 3.8282828282828283, 3.898989898989899,
3.9696969696969697, 4.04040404040404, 4.111111111111111, 4.181818181818182, 4.252525252525253, 4.3232323232323235,
4.393939393939394, 4.4646464646464645, 4.535353535353535, 4.6060606060606055, 4.6767676767676765, 4.747474747474747,
4.818181818181818, 4.888888888888889, 4.959595959595959, 5.03030303030303, 5.101010101010101, 5.171717171717171,
5.242424242424242, 5.313131313131313, 5.383838383838384, 5.454545454545454, 5.525252525252525, 5.595959595959596,
5.666666666666666, 5.737373737373737, 5.808080808080808, 5.878787878787879, 5.949494949494949, 6.02020202020202,
6.090909090909091, 6.161616161616162, 6.232323232323232, 6.303030303030303, 6.373737373737374, 6.444444444444445,
6.515151515151515, 6.585858585858586, 6.656565656565657, 6.727272727272727, 6.797979797979798, 6.8686868686868685,
6.9393939393939394, 7.0101010101010095, 7.08080808080808, 7.151515151515151, 7.222222222222222, 7.292929292929292,
7.363636363636363, 7.434343434343434, 7.505050505050505, 7.575757575757575, 7.646464646464646, 7.717171717171717,
7.787878787878787, 7.858585858585858, 7.929292929292929, 8.0]
yy= [1.0, 1.146413631262116, 1.3028262422201817, 1.4692378328741966, 1.645648403224161, 1.8320579532700747,
2.028466483011938, 2.2348739924497494, 2.4512804815835123, 2.6776859504132227, 2.914090398938884, 3.1604938271604937,
3.416896235078053, 3.683297622691562, 3.95969799000102, 4.246097337006428, 4.542495663707785, 4.848892970105092,
5.165289256198346, 5.491684521987552, 5.828078767472708, 6.174471992653811, 6.530864197530863, 6.897255382103867,
7.273645546372818, 7.660034690337718, 8.056422813998571, 8.462809917355372, 8.87919600040812, 9.305581063156822,
9.74196510560147, 10.188348127742067, 10.644730129578614, 11.111111111111109, 11.587491072339557, 12.073870013263951,
12.570247933884298, 13.07662483420059, 13.593000714212835, 14.119375573921026, 14.655749413325172, 15.20212223242526,
15.758494031221304, 16.32486480971329, 16.90123456790123, 17.48760330578512, 18.083971023364963, 18.690337720640752,
19.306703397612484, 19.933068054280174, 20.569431690643803, 21.215794306703394, 21.87215590245893, 22.538516477910417,
23.214876033057852, 23.901234567901238, 24.597592082440563, 25.30394857667585, 26.020304050607084, 26.746658504234258,
27.48301193755739, 28.22936435057647, 28.985715743291504, 29.752066115702476, 30.528415467809406, 31.314763799612287,
32.11111111111111, 32.917457402305885, 33.733802673196614, 34.560146923783286, 35.39649015406591, 36.24283236404448,
37.099173553719005, 37.96551372308948, 38.8418528721559, 39.72819100091827, 40.62452810937659, 41.53086419753087,
42.44719926538108, 43.37353331292725, 44.30986634016937, 45.25619834710743, 46.21252933374145, 47.178859300071416,
48.15518824609734, 49.14151617181919, 50.13784307723701, 51.144168962350776, 52.160493827160494, 53.18681767166615,
54.22314049586777, 55.26946229976533, 56.32578308335884, 57.39210284664829, 58.46842158963371, 59.55473931231507,
60.651056014692365, 61.75737169676563, 62.87368635853484, 64.0]
```



10.9 USAGE OF PARAMETRIC VARIABLES IN CURVE FITTING

x and y values are given in Table 6.8-1. If it is tried to curve fit an equation to this data, it will fail no matter which method will be used. For the same x values two different y values are defined. In cases

like this in order to curve fit the data an artificial parameter is added and x and y curvefit seperately with this artificial parametric variable.

Table 10.8-1 input data a.txt

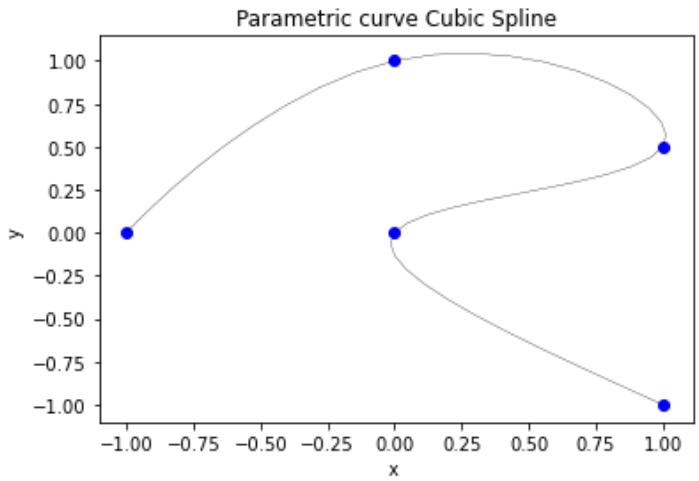
i	0	1	2	3	4
T	0	1	2	3	4
X	-1	0	1	0	1
Y	0	1	0.5	0	-1

```
# -*- coding: utf-8 -*-
"""
Created on Sun Jun  3 21:38:38 2018

@author: Mustafa Turhan Çoban
"""
from math import *
import numpy as np;
import matplotlib.pyplot as plt;
import os;

def thomas(f,e,g,r):
   n=len(f);
   x=[0.0 for i in range(n)];
   for k in range(0,n): e[k]=e[k]/f[k-1];f[k]=f[k]-e[k]*g[k-1];
   for k in range(1,n): r[k]=r[k]-e[k]*r[k-1];
   x[n-1]=r[n-1]/f[n-1];
   for k in range(n-2,-1,-1):  x[k]=(r[k]-g[k]*x[k+1])/f[k];
   return x;

def cubic_spline(xi,yi,c0,cn):
   n=len(xi);
   h=[0.0 for i in range(n)];
   w=[0.0 for i in range(n)];
   f=[0.0 for i in range(n)];
   e=[0.0 for i in range(n)];
   g=[0.0 for i in range(n)];
   d=[0.0 for i in range(n)];
   x=[0.0 for i in range(n)];
   S= [[0 for x in range(n)] for y in range(4)];
   for k in range(0,n-1):
       h[k]=xi[k+1]-xi[k];
       w[k]=(yi[k+1]-yi[k])/h[k];
   d[0]=c0;
   d[n-1]=cn;
   for k in range(0,n-1):
      d[k]=6.0*(w[k]-w[k-1]);
   f[0]=1.0;
   f[n-1]=1.0;
   g[0]=0.0;
   g[n-1]=0.0;
   e[0]=0.0;
   e[n-1]=0.0;
   for k in range(0,n-1):
       f[k]=2.0*(h[k]+h[k-1]);e[k]=h[k-1];g[k]=h[k];
   S[2]=thomas(f,e,g,d);
   S[3]=xi;
   for k in range(0,n-1):
      S[0][k]=(6.*yi[k+1]-h[k]*h[k]*S[2][k+1])/(6.0*h[k]);
      S[1][k]=(6.*yi[k]-h[k]*h[k]*S[2][k])/(6.0*h[k]);
   return S;


def funcSpline(S,x):
   n=len(S[0]);
   xx1=0;
   xx2=0;
   y=0;
   hk=0;
```

```
    for k in range(0,n-1):
        if S[3][k]<=x and x<=S[3][k+1]:
            hk=(S[3][k+1]-S[3][k]);
            xx1=(x-S[3][k]);
            xx2=(S[3][k+1]-x);
            y=S[0][k]*xx1+S[1][k]*xx2+(xx1*xx1*xx1*S[2][k+1]+xx2*xx2*xx2*S[2][k])/(6.0*hk);
            break;
    if y==0 and S[3][n-2]<=x:
        k=n-2;
        hk=(S[3][k+1]-S[3][k]);
        xx1=(x-S[3][k]);
        xx2=(S[3][k+1]-x);
        y=S[0][k]*xx1+S[1][k]*xx2+(xx1*xx1*xx1*S[2][k+1]+xx2*xx2*xx2*S[2][k])/(6.0*hk);
    return y;

def listSpline(xi,yi,numberofmidpoints,c0,cn):
    # numberofmidpoints :  in x--o--o--x--o--o--x chain if x's are esxperimental points
                # numberofmidpoints is 2
    n=len(xi);
    nn=(n-1)*(numberofmidpoints+1)+1;
    z=[[0 for x in range(nn)] for y in range(2)];
    S= [[0 for x in range(nn)] for y in range(4)];
    S=cubic_spline(xi,yi,c0,cn);
    dx=0;
    k=0;
    for i in range(0,n-1):
        z[0][k]=xi[i];z[1][k]=funcSpline(S,z[0][k]);k=k+1;
        for j in range(numberofmidpoints):
            dx=(xi[i+1]-xi[i])/(numberofmidpoints+1.0);
            z[0][k]=z[0][k-1]+dx;z[1][k]=funcSpline(S,z[0][k]);k=k+1;
    z[0][k]=xi[i+1];z[1][k]=funcSpline(S,z[0][k]);
    return z;

def read_array2D(d):
    path = d
    file1 = open(path,'r')
    file1.seek(0,2)  #jumps to the end
    endLocation=file1.tell()
    file1.seek(0) #jumps to the start
    nn=0
    m=0
    while True:
        line=file1.readline()
        ee=line.split()
        nn+=1
        if  file1.tell()==endLocation:
            break
    m=len(ee)
    print("m=",m)
    a=[[0.0 for i in range(nn)] for j in range(m)]
    file1.seek(0) #jumps to the start
    for i in range(nn):
        line=file1.readline()
        ee=line.split()
        for j in range(m):
            a[j][i]=float(ee[j])
    file1.close()
    return a

a=read_array2D('bb.txt')
x=a[0]
print("x=",x)
y=a[1]
print("y=",y)
n=len(x)
t=[0.0 for i in range(n)]
for i in range(n):
    t[i]=float(i)
z1=listSpline(t,x,10,0,0)
z2=listSpline(t,y,10,0,0)
plt.title('Parametric curve Cubic Spline')
```

```
plt.xlabel('x ')
plt.ylabel('y ');
plt.plot(x,y,'bo',z1[1],z2[1],'k',linewidth=0.3)
```

```
runfile('E:/okul/SCO1/parametric_cubic_spline.py', wdir='E:/okul/SCO1')
m= 2
x= [-1.0, 0.0, 1.0, 0.0, 1.0]
y= [0.0, 1.0, 0.5, 0.0, -1.0]
```



10.10 BERNSTEIN POLINOMIALS AND BEZIER CURVES

Figure 6.11.1 Felix Bernstein, German Mathematician

Bernstein polynomials are a serial function approximation using the following form in a range of (a,b)

$$B_n(f;x) = \sum_{i=0}^{n} \binom{n}{i} (x-a)^i (b-x)^{n-i} f(x_i) \qquad (6.11-1)$$

x_i in the equation can be written for constant point differences x_i=a+ih=a+(i/n)(b-a) i=0..n . If the same function is written in (0,1) range:

$$B_n(f;x) = \sum_{i=0}^{n} \binom{n}{i} x^i (1-x)^{n-i} f\left(\frac{i}{n}\right) = \sum_{i=0}^{n} f\left(\frac{i}{n}\right) B_{n,i}(x) \quad (6.11-2)$$

$$B_{n,i}(x) = \sum_{i=0}^{n} \binom{n}{i} x^i (1-x)^{n-i} \qquad (6.11-3)$$

When Bernstein polynomials are used for curve fitting approximations, it is seen that convertion rates are very low. Very high number of iterations are required for relatively good approximations. They will pass from starting and end points exactly but for the rest of the points it approximate slowly. But

if mid-points are only taken as a reference. A practical method for drawing complicated shapes is obtained. This type of Bernstein polynomial applications have a special name: Bezier curves. Bezier curves uses parametric variables. For any (x_i , y_i) set with the parametric variable t

$$x(t) = \sum_{i=0}^{n} x_i \, B_{n,i}(t) \qquad (6.11-4)$$

$$y(t) = \sum_{i=0}^{n} y_i \, B_{n,i}(t) \qquad (6.11-5)$$

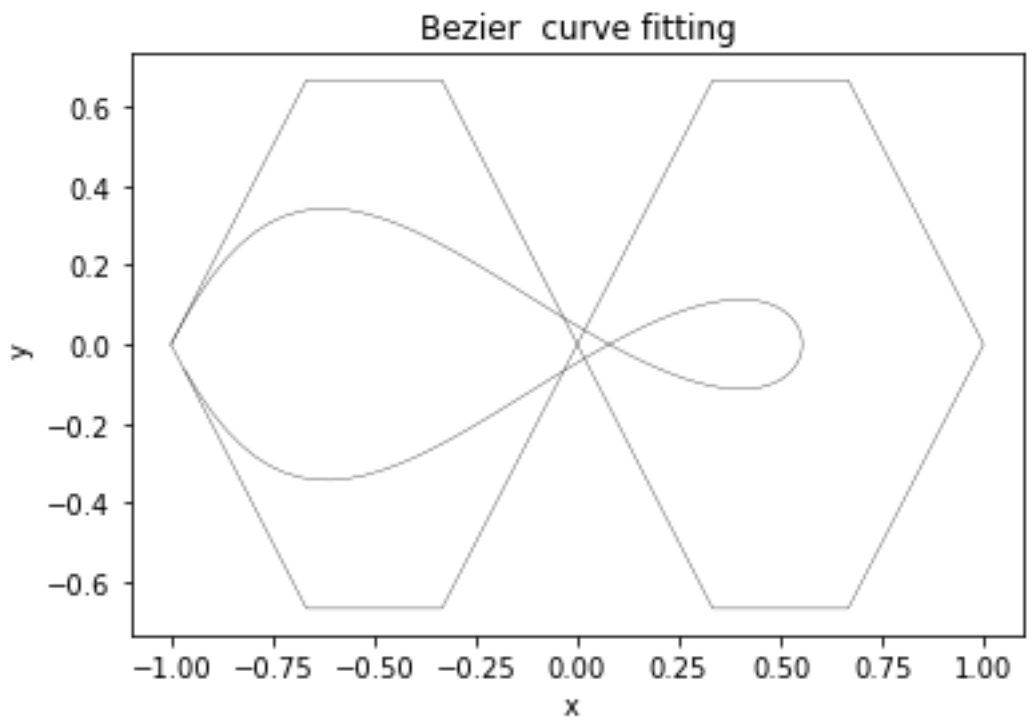
Let us use the following Example data to create th Bezier curve:

-1.0000000000	0.0000000000
-0.6666666667	0.6666666667
-0.3333333333	0.6666666667
0.3333333333	-0.6666666667
0.6666666667	-0.6666666667
1.0000000000	0.0000000000
0.6666666667	0.6666666667
0.3333333333	0.6666666667
-0.3333333333	-0.6666666667
-0.6666666667	-0.6666666667
-1.0000000000	0.0000000000

```
# Bernstein polinomials and Bezier curves
from math import *
import numpy as np;
import matplotlib.pyplot as plt;
import os;
def B(n,t):
    #   0<=t<=1
    b=[0 for x in range(n+1)]
    b[0]=1.0;
    for i in range(1,n+1):
        b[i]=t*b[i-1]
        j=i-1
        while j>=1:
            b[j]=(1.0-t)*b[j]+t*b[j-1]
            j=j-1
        b[0]=(1.0-t)*b[0];
    return b

def funcB(a,t):
    xx=0
    n=len(a)-1
    b=B(n,t)
    for i in range(n+1):
        xx=xx+a[i]*b[i]
    return xx


def Bezier(xi,yi,noktasayisi):
    n=len(xi)
    z=[[0 for x in range(noktasayisi)] for y in range(2)];
    dt=1.0/float(noktasayisi)
    t=0.0
    k=0
    for i in range(noktasayisi):
        z[0][i]=funcB(xi,t)
        z[1][i]=funcB(yi,t)
        t=t+dt
    return z

def read_array2D(d):
    path = d
    file1 = open(path,'r')
    file1.seek(0,2)  #jumps to the end
    endLocation=file1.tell()
    file1.seek(0) #jumps to the start
    nn=0
    m=0
    while True:
        line=file1.readline()
```

```
        ee=line.split()
        nn+=1
        if  file1.tell()==endLocation:
            break
    m=len(ee)
    print("m=",m)
    a=[[0.0 for i in range(nn)] for j in range(m)]
    file1.seek(0) #jumps to the start
    for i in range(nn):
        line=file1.readline()
        ee=line.split()
        for j in range(m):
            a[j][i]=float(ee[j])
    file1.close()
    return a

a=read_array2D('a.txt')
x=a[0]
print("x=",x)
y=a[1]
print("y=",y)
n=len(x)
z1=Bezier(x,y,100)
plt.title("Bezier  curve fitting")
plt.xlabel('x ')
plt.ylabel('y ');
plt.plot(x,y,'k',z1[0],z1[1],'k',linewidth=0.3)
```

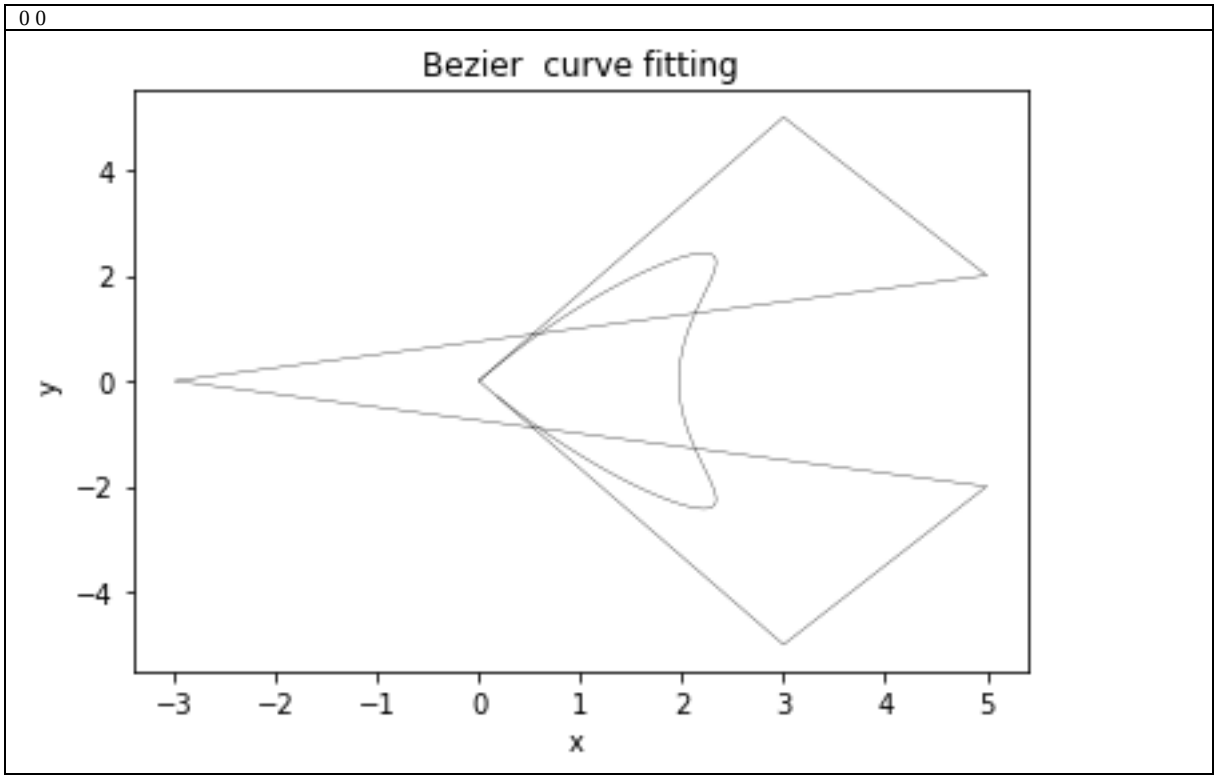
```
runfile('E:/okul/SCO1/bernstein_polynomials.py', wdir='E:/okul/SCO1')
m= 2
x= [-1.0, -0.6666666667, -0.3333333333, 0.3333333333, 0.6666666667, 1.0, 0.6666666667, 0.3333333333, -0.3333333333, -
0.6666666667, -1.0]
y= [0.0, 0.6666666667, 0.6666666667, -0.6666666667, -0.6666666667, 0.0, 0.6666666667, 0.6666666667, -0.6666666667, -
0.6666666667, 0.0]

In [226]:
```



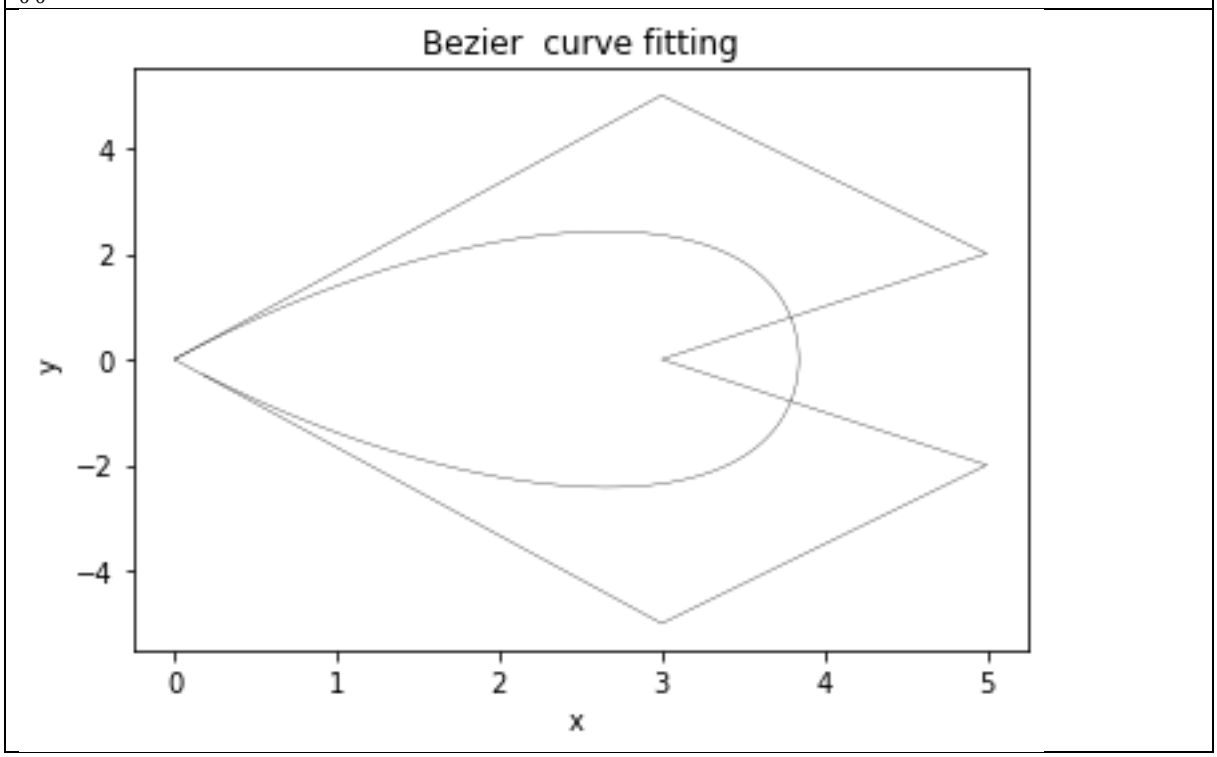
By changing reference points very complicated shapes can be drawn For example

```
0 0
3 5
5 2
-3 0
5 -2
3 -5
```

```
0 0
```




```
0 0
3 5
5 2
3 0
5 -2
3 -5
0 0
```



10.11 RATIONAL FUNCTION APPROXIMATION : PADE APPROXIMATION METHOD

Henri Pade

The class of algebraic polynomials has some distinct advantages for use in approximations. The disadvantage of using polynomials for approximation is their tendency to oscilate. This often causes error bounds in polynomial approximation to significantly exceed the avarage approximation error. Another family of functions existed that spread the approximation error more evenly over the approximation interval. This family of functions are called **rational functions.** A rational function r of degree N has the form :

$r(x) = \frac{p(x)}{q(x)}$ (6.13-1)

Where p(x) and q(x) are polynomials whose degrees sum to N.

$$r(x) = \frac{p(x)}{q(x)} = \frac{p_0 + p_1 x + p_2 x^2 + \cdots + p_n x^n}{q_0 + q_1 x + q_2 x^2 + \cdots + q_n x^n} \quad (6.13-2)$$

Pade′ rational approximation is based on Taylor expansion of the functions.

$$f(x) = f(x_0) + f'^{(x_0)}(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \frac{f^{(3)}(x_0)}{3!}(x - x_0)^3 + .. + \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n \quad (6.13-3)$$

The special form of the Taylor expansion when x_0=0 was called MacLaurin expansion.

$$f(x) = f(0) + f'(0)x + \frac{f''(0)}{2!}x^2 + \frac{f^{(3)}(0)}{3!}x^3 + .. + \frac{f^{(n)}(0)}{n!}x^n \quad (6.13-4)$$

$$f(x) = \sum_{i=0}^{\infty} a_i x^i \quad (6.13-5)$$

Now, if the function f(x) is opened to a MacLaurin series of degree N

$$r(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots + a_N x^N$$

Consider the error term difference of the function f(x) and r(x) due to finite series approximation of the function.

$$f(x) - r(x) = f(x) - \frac{p(x)}{q(x)} = \frac{f(x)q(x) - p(x)}{q(x)} = \frac{\sum_{i=0}^{\infty} a_i x^i \sum_{i=0}^{\infty} q_i x^i - \sum_{i=0}^{\infty} p_i x^i}{q(x)} \quad (6.13-6)$$

We can assume that q_0=1. The object is to chose the constants $q_1, q_2, \ldots, q_m$ and $p_1, p_2, \ldots, p_n$ so that the right hand term of the above equation

$$(a_0 + a_1 x + a_2 x^2 + \cdots)(1 + q_1 x + q_2 x^2 + \cdots + q_n x^m) - (p_0 + p_1 x + p_2 x^2 + \cdots + p_n x^n) \quad (6.13-7)$$

has no term of degree less than or equal to N. To simplify the notation, we define $p_{n+1}=p_{n+2}=\ldots p_N=0$ and define $q_{m+1}=q_{m+2}=\ldots q_m=0$. We can then express the coefficient of x^k in the expression as

$$\left(\sum_{i=0}^{k} a_i q_{k-i}\right) - p_k \quad (6.13-8)$$

So the rational function pade approximation results from the solution of n+1 linear equation

$$\left(\sum_{i=0}^{k} a_i q_{k-i}\right) = p_k \quad k = 0, \dots, N$$

In the N+1 unknowns $q_1, q_2, \dots, q_{m,} p_0, p_1, \dots, p_{m,}$
As a first example take function $f(x) = e^x$
MacLaurin expansion of the function $f(x) = e^x$ for error $O(x^5)$,
$f(x) = e^x, \quad f(0) = 1$
$f'(x) = e^x, \quad f'(0) = 1$

$f"(x) = e^x, \quad f"(0) = 1$

$f'''(x) = e^x, \quad f'''(0) = 1$

$f""(x) = e^x, \quad f""(0) = 1$

Maclaurin equation then becomes;
$f(x) = e^x = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} + O(x^5)$
$p(x) = p_0 + xp_1 + x^2 p_2$
$q(x) = 1 + xq_1 + x^2 q_2$
$f(x)q(x) - p(x) = 0$

$$(1 - p_0) + (1 - p_1 + q_1)x + \left(\frac{1}{2} - p_2 + q_1 + q_2\right)x^2 + \left(\frac{1}{6} + \frac{q_1}{2} + q_2\right)x^3 + \left(\frac{1}{24} + \frac{q_1}{6} + \frac{q_2}{2}\right)x^4 + O(x^4) = 0$$

$(1 - p_0) = 0$
$(1 - p_1 + q_1) = 0$
$\left(\frac{1}{2} - p_2 + q_1 + q_2\right) = 0$
$\left(\frac{1}{6} + \frac{q_1}{2} + q_2\right) = 0$
$\left(\frac{1}{24} + \frac{q_1}{6} + \frac{q_2}{2}\right) = 0$
Solution of the equation set gives:
$p_0 = 1 \;\; p_1 = 1/2 \;\; p_1 = 1/12 \;\; q_1 = -1/2 \;\; q_2 = 1/12$
And $r(x) = \frac{1 + \frac{1}{2x} + \frac{1}{12}x^2}{1 - \frac{1}{2x} + \frac{1}{12}x^2}$
by simplifying
$r(x) = \frac{12 + 6x + x^2}{12 - 6x + x^2}$
Maximum error for x=1 is found as

$$f(x) - r(x) = e^x - \frac{12 + 6x + x^2}{12 - 6x + x^2} = 0.00399611$$

Let us solve the same equation, by developing the computer code. In the code MacLauring derivative terms are calculated up to term 8. But also another input possibility of directly inputting the polynomial coefficients are given. In case of higher degree of polynomials, coefficients can directly be inputted into the program.

Function to be approximated $f(x) = e^x$ around x_a=0

Abstract class f_x to calculate derivatives in McLaurin series

```
from abc import ABC, abstractmethod
from math import *

class f_x(ABC):

    @abstractmethod
```

```
def func(self,x):
    pass;

def dfunc(self,x):
    h=1e-2
    n=1
    M=50
    return self.dnMfunc(x,n,M,h)

def d2func(self,x):
    h=1e-4
    n=2
    M=50
    return self.dnMfunc(x,n,M,h)

def d3func(self,x):
    h=1e-4
    n=3
    M=50
    return self.dnMfunc(x,n,M,h)

def dnfunc(self,x,n):
    h=1e-2+1e-2*float(n)
    M=15
    return self.dnMfunc(x,n,M,h)

def dnMfunc(self,x,N,Mi,hi):
    # order of the maximum derivative
    # N order of derivative
    # M degree of difference formula
    M=Mi;
    #double a[]=new double[0];
    h=hi;
    x0=0.0;
    alphai=[0,1,-1,2,-2,3,-3,4,-4,5,-5,6,-6,7,-7,8,-8,9,-9,10,-10,11,-11,12,-12,13,-13,14,-14,15,
    -15,16,-16,17,-17,18,-18,19,-19,20,-20,21,-21,22,-22,23,-23,24,-24,25,-25,26,-26,27,-27,28,-28,29,-29,30,-30,
    31,-31,32,-32,33,-33,34,-34,35,-35,36,-36,37,-37,38,-38,39,-39,40,-40,41,-41,42,-42,43,-43,44,-44,45,-45,46,
    -46,47,-47,48,-48,49,-49,50,-50,51,-51,52,-52,53,-53,54,-54,55,-55,56,-56,57,-57,58,-58,59,-59,60,-60,
    -61,61,-62,62,-63,63,-64,64,-65,65,-66,66,-67,67,-68,68,-70,70,-71,71,-72,72,-73,73,-74,74,-75,75,
    -76,76,-77,77,-78,78,-79,79,-80,80,-81,81,-82,82,-83,83,-84,84,-85,85,-86,86,-87,87,
    -88,88,-89,89,-90,90,-91,91,-92,92,-93,93,-94,94,-95,95,-96,96,-97,97,-98,98,-99,99,-100,100]
    alpha=alphai
    N1=len(alpha)-1;
    # M   degree of highest derivative
    #N+1 number of coefficients
    delta = [[[0 for x in range(M+1)] for y in range(N1+1)] for z in range(N1+1)];
    #double delta[][][]=new double[N1+1][N1+1][M+1];
    #double c1,c2,c3;
    delta[0][0][0]=1.0;
    c1=1.0
    for n in range(1,N1+1):
        c2=1
        for nu in range(0,n):
            c3=alpha[n]-alpha[nu]
            c2=c2*c3
            if n<=M: delta[n-1][nu][n]=0.0
            for m in range(0,(min(n,M)+1)):
                if m==0:
                    delta[n][nu][m]=((alpha[n]-x0)*delta[n-1][nu][m])/c3
                else:
                    delta[n][nu][m]=((alpha[n]-x0)*delta[n-1][nu][m]-m*delta[n-1][nu][m-1])/c3
        for m in range(0,(min(n,M)+1)):
            if m==0:delta[n][n][m]=c1/c2*(-(alpha[n-1]-x0)*delta[n-1][n-1][m])
            elif m!=0: delta[n][n][m]=c1/c2*(m*delta[n-1][n-1][m-1]-(alpha[n-1]-x0)*delta[n-1][n-1][m])
        c1=c2
    c=delta
    if Mi<N: M=N
    else: M=Mi
    h=hi;
    deriv=0
    h1=1.0/float(h)
    h2=1.0;
    for j in range(N):
      h2=h2*h1
    for i in range(0,len(c[0])):
      deriv=deriv+c[M][i][N]*self.func(x+alpha[i]*h)
```

```
        return deriv*h2

    def gauss_legendre_coefficients(self,x1,x2,n):
        #calculates legendre gauss-coefficients as coefficients of the integral
        #for n terms
        eps=3e-15
        m=int((n+1.0)/2.0)
        xm=0.5*(x2+x1)
        xl=0.5*(x2-x1)
        nmax=100
        a=[[0.0 for i in range(n)] for j in range(2)]
        for i in range(1,m+1):
            z=cos(pi*((i-0.25)/(n+0.5)))
            for ii in range(nmax):
                #while abs(z-z1) > eps:
                p1=1.0;
                p2=0.0;
                for j in range(1,n+1):
                    p3=p2
                    p2=p1
                    p1=(float)(((2.0*j-1.0)*z*p2-(j-1.0)*p3)/j)
                pp=(float)(n*(z*p1-p2)/(z*z-1.0))
                z1=z;
                z=z1-p1/pp
                if abs(z-z1) < eps: break
            a[0][i-1]=xm-xl*z
            a[0][n-i]=xm+xl*z
            a[1][i-1]=2.0*xl/((1.0-z*z)*pp*pp)
            a[1][n-i]=a[1][i-1]
        return a

    def integral(self,x1,x2,n):
        #integral func(x)dx
        #integral of a function by using gauss-legendre quadrature
        #between x1 and x2
        a=self.gauss_legendre_coefficients(x1,x2,n)
        z=0
        for i in range(n):
            z=z+a[1][i]*self.func(a[0][i])
```

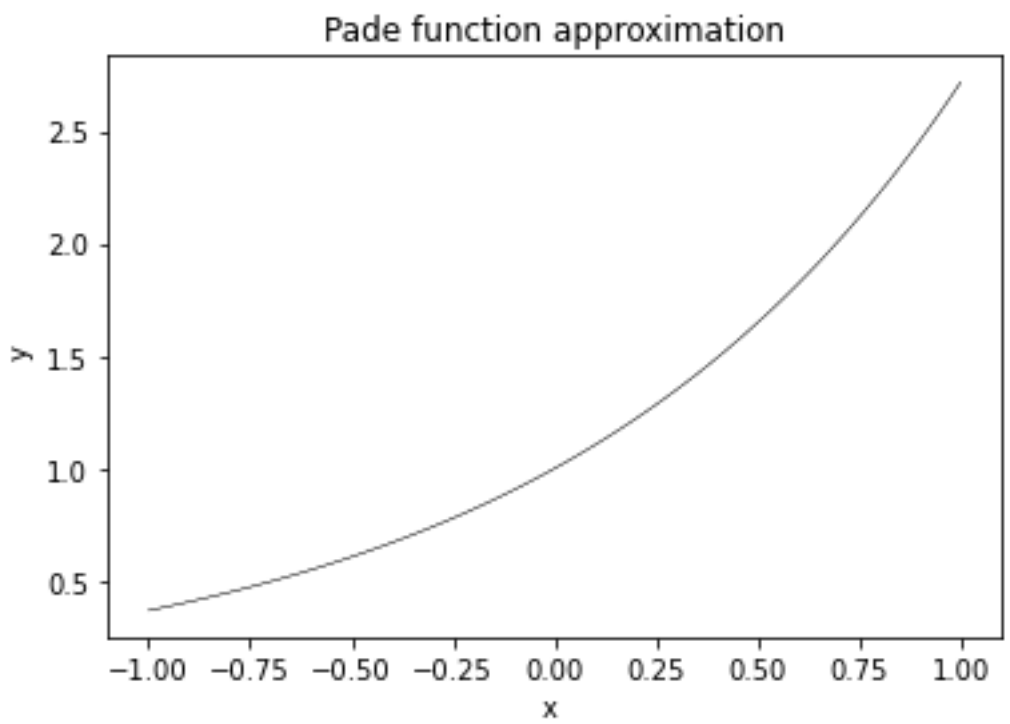
Pade approximation to functions

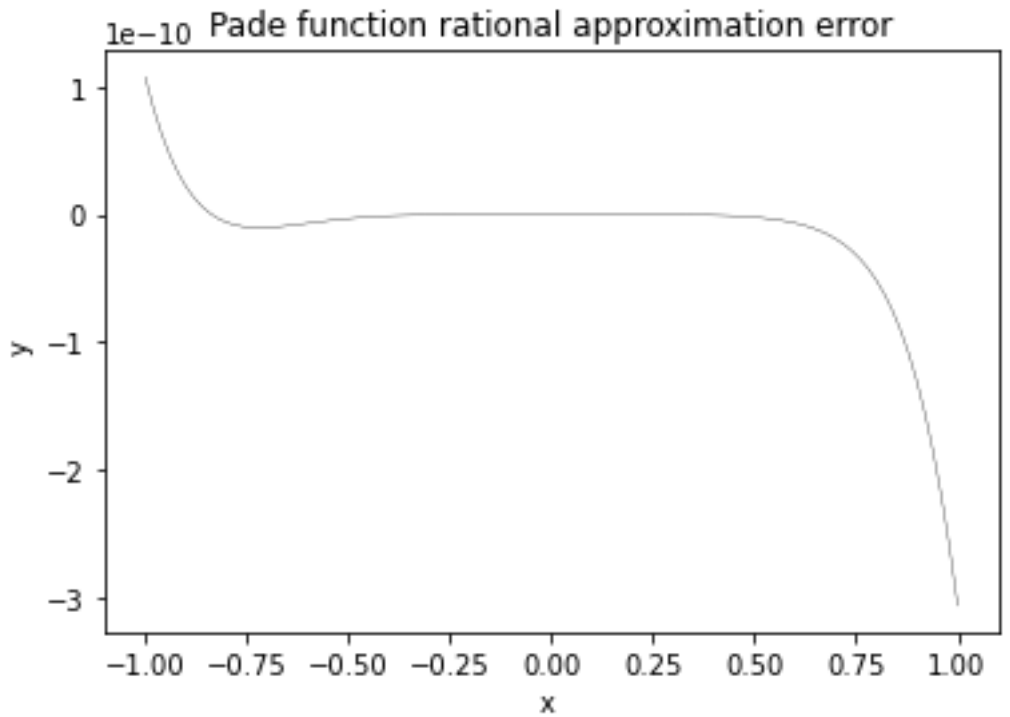
```
from math import *
from f_x import *
import gauss
import numpy as np;
import matplotlib.pyplot as plt;
import os;

def Maclaurin(f,x,n):
    #McLaurin series
    a=[0.0 for i in range(n+1)];
    fact=1
    for i in range(n+1):
        a[i]=f.dnfunc(x,i)/float(fact)
        fact=fact*(i+1)
    return a

def poly_func(e,x):
    # this function calculates the value of
    # least square curve fitting function
    n=len(e)
    ff=0.0
    if n!=0.0:
        ff=e[n-1]
        for i in range(n-2,-1,-1):
            ff=ff*x+e[i]
    else:
        ff=0.0
    return ff

def func(p,q,x):
    return poly_func(p,x)/poly_func(q,x)


def pade(ai,n,m):
```

```
    N=m+n
    NP1=N+1
    a=[0.0 for i in range(NP1)]
    for i in range(NP1):
        a[i]=ai[i]
    print("a=",a)
    q=[0.0 for i in range(m+1)]
    p=[0.0 for i in range(n+1)]
    q[0]=1.0
    p[0]=a[0]
    b=[[0.0 for j in range(NP1)] for i in range(NP1)]
    bb=[[0.0 for j in range(N)] for i in range(N)]
    c=[0.0 for i in range(N)]
    for i in range(1,NP1):
        for j in range(1,(i-1)):
            if j<=n:
                b[i][j]=0.0
        if i<=n:
            b[i][i]=1.0
        for j in range(i+1,NP1):
            b[i][j]=0.0
        for j in range(1,(i+1)):
            if j<=m:
                b[i][n+j]=-a[i-j]
        for j in range((n+i+1),NP1):
            b[i][j]=0
        c[i-1]=a[i]
    for i in range(N):
        for j in range(N):
            bb[i][j]=b[i+1][j+1]
    x=gauss.gauss(bb,c)
    for i in range(n):
        p[i+1]=x[i]
    for i in range(n,N):
        q[i+1-n]=x[i]
    aa=[p,q]
    return aa
def plot_func(f,p,q,a,b):
    nn=101
    xx=[0.0 for j in range(nn)]
    yy=[0.0 for j in range(nn)]
    yy1=[0.0 for j in range(nn)]
    for i in range(nn):
        xx[i]=a+i*(b-a)/(nn-1)
        yy[i]=func(p,q,xx[i])
        yy1[i]=f.func(xx[i])
    plt.title('Pade function approximation')
    plt.xlabel('x ')
    plt.ylabel('y ');
    plt.plot(xx,yy,'k',xx,yy1,'k',linewidth=0.3)

def plot_func_error(f,p,q,a,b):
    nn=101
    xx=[0.0 for j in range(nn)]
    yy=[0.0 for j in range(nn)]
    yy1=[0.0 for j in range(nn)]
    yy2=[0.0 for j in range(nn)]
    for i in range(nn):
        xx[i]=a+i*(b-a)/(nn-1)
        yy[i]=func(p,q,xx[i])
        yy1[i]=f.func(xx[i])
        yy1[i]=f.func(xx[i])
        yy2[i]=yy[i]-yy1[i]
    plt.title('Pade function rational approximation error')
    plt.xlabel('x ')
    plt.ylabel('y ');
    plt.plot(xx,yy2,'k',linewidth=0.3)

class f1(f_x):func=lambda self,x: exp(x)
f = f1()
x=0.0;
nn=10
a= Maclaurin(f,x,nn)
n=5
m=5
b=pade(a,n,m)
```

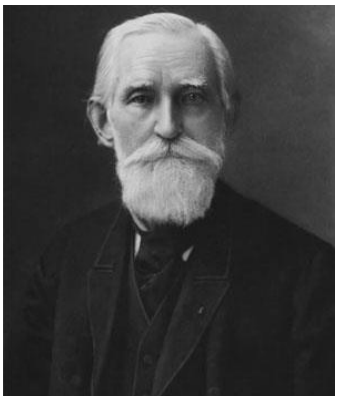
```
p=b[0]
q=b[1]
print("p=",p)
print("q=",q)
#plot_func(f,p,q,-1.0,1.0)
plot_func_error(f,p,q,-1.0,1.0)
```

```
runfile('E:/okul/SCO1/pade.py', wdir='E:/okul/SCO1')
Reloaded modules: f_x, gauss
a= [1.0, 1.0000000000000089, 0.5000000000003845, 0.16666666666769606, 0.041666666657391244, 0.00833333335205604,
0.0013888886774995873, 0.00019841275260375741, 2.48019370180262e-05, 2.7554250936454533e-06, 2.752535884189491e-07]
p= [1.0, 0.4964398301816557, 0.10927006227732894, 0.013476817335377644, 0.0009435159888604785, 3.043158784563592e-05]
q= [1.0, -0.5035601698183532, 0.11283023209530206, -0.014239996518251275, 0.0010284247722701014, -3.469322225524563e-05]
```





10.12 RATIONAL FUNCTION APPROXIMATION : CHEBYSCHEV APPROXIMATION METHOD



Pafnuty Chebyshev, Russian Mathematician

Pade approximation based on Taylor expansion of the functions. Taylor expansion has a wide variation of accuracy. To obtain more uniformly accurate rational-function approximation Chebyschev polynomials can be used. In chapter 6.2 Chebyschev polynomials were defined and used as orthogonal least square curve fitting functions.

$T_n(x)=\cos(n \arccos(x))$ (6.14-1)

If this equation is written in open form

$T_0(x)=1$
$T_1(x)=x$
$T_2(x)=2x^2-x$
…..
$T_{n+1}(x)=2x\, T_n(x)- T_{n-1}(x)$ (6.14-2)

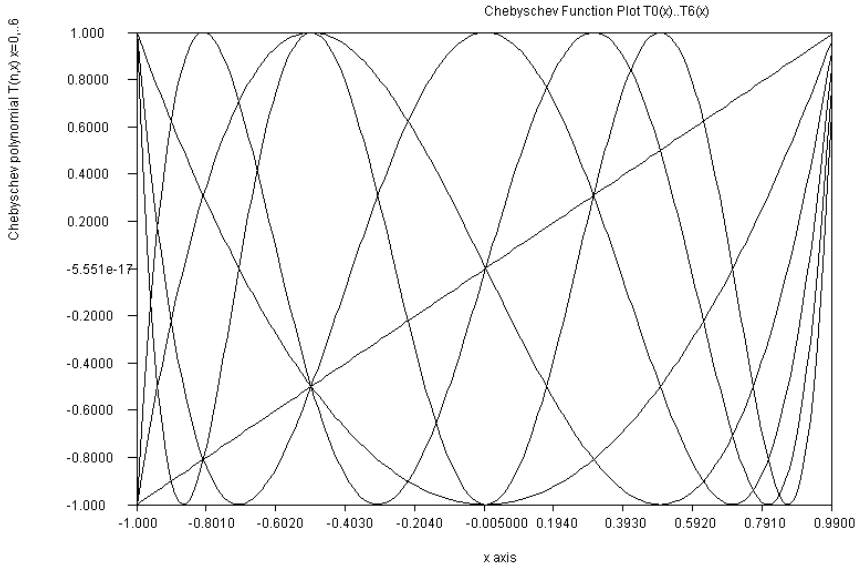


Figure 6.14-1 Chebyschev polynomials $T_0(x)$ through $T_6(x)$

Now if x^k terms of the pade approximations is replaced by Chebyschev polynomial $T_k(x)$ Chebyschev approximation is constructed. Suppose we want to approximate the function f by an N-th degree rational function r written in the form

$$r(x) = \frac{p(x)}{q(x)} = \frac{\sum_{k=0}^{n} p_k T_k(x)}{\sum_{k=0}^{n} q_k T_k(x)} \quad (6.14\text{-}3)$$

where N=n+m and $q_0=1$

Writing f(x) in a series involving Chebyschev polynomials as:

$f(x) = \sum_{k=0}^{n} a_k T_k(x)$ gives

$$f(x) - r(x) = f(x) - \frac{p(x)}{q(x)} = \frac{\sum_{k=0}^{n} a_k T_k(x) \sum_{k=0}^{n} q_k T_k\,(x) - \sum_{k=0}^{n} p_k T_k(x)}{\sum_{k=0}^{n} q_k T_k\,(x)} \quad (6.14-4)$$

The coefficients $q_1, q_2, \ldots, q_m$ and $p_1, p_2, \ldots, p_n$ are chosen so that the numerator on the right hand side of the equation has zero coefficients for $T_k(x)$ when k=0,1,..N. This implies that the series

$$(a_0 T_0(x) + a_1 T_1(x) + ..)\big(T_0(x) + q_1 T_1(x) + .. + q_m T_m(x)\big) - \big(p_0 T_0(x) + p_1 T_1(x) + .. + p_m T_m(x)\big) \quad (6.14-5)$$

Has no terms of degree less then or equal to N.

Solution of this equation involves few difficulties, the first one is the requirement of multiplication of Chebyschev terms. This problem is resolved by making use of the relationship

$$T_i(x) T_j(x) = 0.5\big[T_{i+1}(x) + T_{|i-j|}(x)\big] \quad (6.14-6)$$

Coefficients need to be calculated by using integration equations:

$$a_0 = \frac{1}{\pi} \int_{x=-1}^{1} \frac{f(x)}{\sqrt{1-x^2}} dx = \frac{1}{\pi} \int_{\theta=0}^{\pi} f(\cos(\theta)) d\theta \quad (6.14-7)$$

$$a_k = \frac{1}{\pi} \int_{x=-1}^{1} \frac{f(x) T_k(x)}{\sqrt{1-x^2}} dx = \frac{1}{\pi} \int_{\theta=0}^{\pi} f(\cos(\theta)) \cos(k\theta)\, d\theta \quad (6.14-8)$$

These are difficult integrals to solve analytically. Usually Numerical methods are involved to solve the integrals. Please refer related chapter for the details of the numerical integrals.

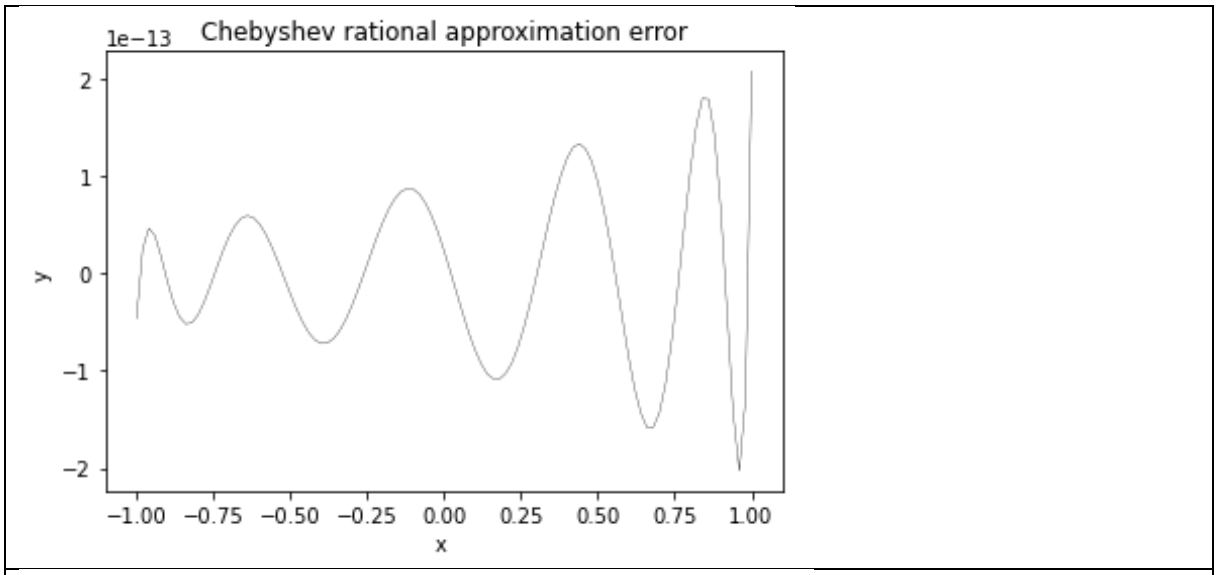
```
 from math import *
from f_x import *
import gauss
import numpy as np;
import matplotlib.pyplot as plt;
import os;

def  Ti(x,k):
   #Chebysev function
   T=[0 for j in range(k+1)]
   T[0]=1.0
   if k>=1:
      T[1]=x
   for i in range(2,k+1):
      T[i]=2.0*x*T[i-1]-T[i-2]
   return T[k]

def fk(f,k,x):
   a=0
   mult=2.0/pi
   a=mult*f.func(cos(x))*cos(k*x)
   return a

#def gauss_kronrod_integral(f,a,b,k,n,eqn_type):
def integral(ff,a,b,k,n):
   # eps hata miktarı
   eps=1.0e-10
   Aused=0
   integral=0
   toplamhata = 0
   nn = 0
   ng = 0
   i = 0
   j = 0
   h = 0
   v = 0
   k1 = 0
   k2 = 0
   intg = 0
   intk = 0
   ta = 0
   tb = 0
   Aw = 4
   A = [[0.0 for i in range(Aw-1+1)] for j in range(n-1+1)]    #[n-1+1][Aw-1+1];
   nn = 61;
   ng = 15;
   x =[0.0 for i in range(nn)]
   ck =[0.0 for i in range(nn)]
```

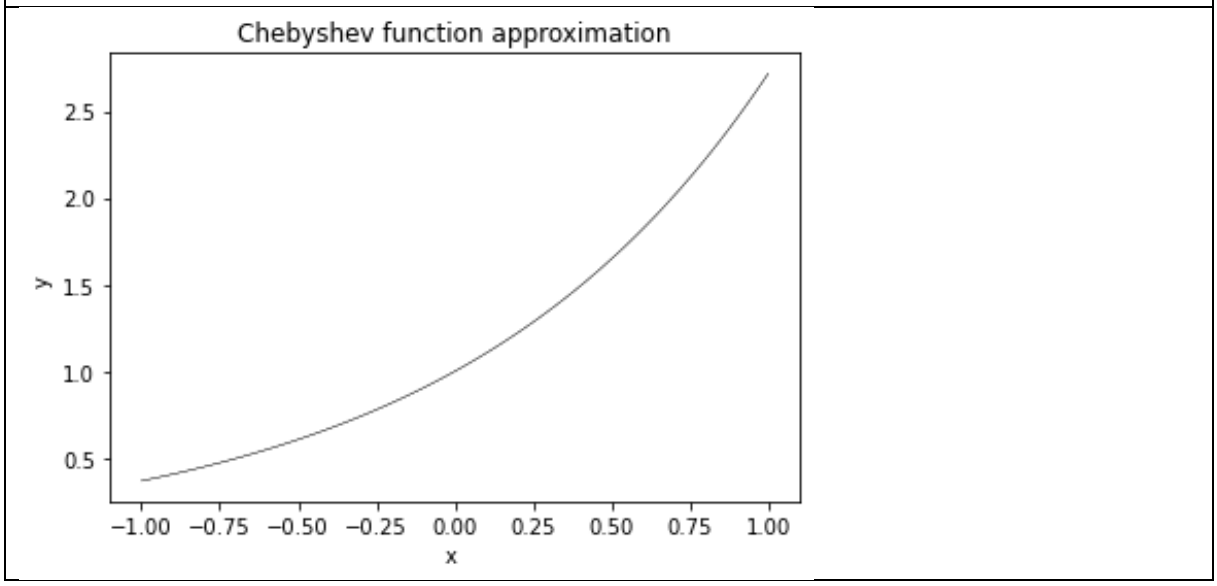
```
cg =[0.0 for i in range(nn)]
#Gauss coefficients
cg[0] = 0.007968192496166605615465883474674
cg[1] = 0.018466468311090959142302131912047
cg[2] = 0.028784707883323369349719179611292
cg[3] = 0.038799192569627049596801936446348
cg[4] = 0.048402672830594052902938140422808
cg[5] = 0.057493156217619066481721689402056
cg[6] = 0.065974229882180495128128515115962
cg[7] = 0.073755974737705206268243850022191
cg[8] = 0.080755895229420215354694938460530
cg[9] = 0.086899787201082979802387530715126
cg[10] = 0.092122522237786128717632707087619
cg[11] = 0.096368737174644259639468626351810
cg[12] = 0.099593420586795267062780282103569
cg[13] = 0.101762389748405504596428952168554
cg[14] = 0.102852652893558840341285636705415
x[0] = 0.999484410050490637571325895705811
x[1] = 0.996893484074649540271630050918695
x[2] = 0.991630996870404594858628366109486
x[3] = 0.983668123279747209970032581605663
x[4] = 0.973116322501126268374693868423707
x[5] = 0.960021864968307512216871025581798
x[6] = 0.944374444748559979415831324037439
x[7] = 0.926200047429274325879324277080474
x[8] = 0.905573307699907798546522558925958
x[9] = 0.882560535792052681543116462530226
x[10] = 0.857205233546061098958658510658944
x[11] = 0.829565762382768397442898119732502
x[12] = 0.799727835821839083013668942322683
x[13] = 0.767777432104826194917977340974503
x[14] = 0.733790062453226804726171131369528
x[15] = 0.697850494793315796932292388026640
x[16] = 0.660061064126626961370053668149271
x[17] = 0.620526182989242861140477556431189
x[18] = 0.579345235826361691756024932172540
x[19] = 0.536624148142019899264169793311073
x[20] = 0.492480467861778574993693061207709
x[21] = 0.447033769538089176780609900322854
x[22] = 0.400401254830394392535476211542661
x[23] = 0.352704725530878113471037207089374
x[24] = 0.304073202273625077372677107199257
x[25] = 0.254636926167889846439805129817805
x[26] = 0.204525116682309891438957671002025
x[27] = 0.153869913608583546963794672743256
x[28] = 0.102806937966737030147096751318001
x[29] = 0.051471842555317695833025213166723
x[30] = 0.000000000000000000000000000000000
#Kronrod coefficients
ck[0] = 0.001389013698677007624551591226760
ck[1] = 0.003890461127099884051267201844516
ck[2] = 0.006630703915931292173319826369750
ck[3] = 0.009273279659517763428441146892024
ck[4] = 0.011823015253496341742232898853251
ck[5] = 0.014369729507045804812451432443580
ck[6] = 0.016920889189053272627572289420322
ck[7] = 0.019414141193942381173408951050128
ck[8] = 0.021828035821609192297167485738339
ck[9] = 0.024191162078080601365686370725232
ck[10] = 0.026509954882333101610601709335075
ck[11] = 0.028754048765041292843978785354334
ck[12] = 0.030907257562387762472884252943092
ck[13] = 0.032981447057483726031814191016854
ck[14] = 0.034979338028060024137499670731468
ck[15] = 0.036882364651821229223911065617136
ck[16] = 0.038678945624727592950348651532281
ck[17] = 0.040374538951535959111995279752468
ck[18] = 0.041969810215164246147147541285970
ck[19] = 0.043452539701356069316831728117073
ck[20] = 0.044814800133162663192355551616723
ck[21] = 0.046059238271006988116271735559374
ck[22] = 0.047185546569299153945261478181099
ck[23] = 0.048185861757087129140779492298305
ck[24] = 0.049055434555029778887528165367238
ck[25] = 0.049795683427074206357811569379942
ck[26] = 0.050405921402782346840893085653585
```

```
    ck[27] = 0.050881795898749606492297473049805
    ck[28] = 0.051221547849258772170656282604944
    ck[29] = 0.051426128537459025933862879215781
    ck[30] = 0.051494729429451567558340433647099
    for i in range(nn-1,int(nn/2-1),-1):
        x[i] = -x[nn-1-i]
    for i in range(nn-1,int(nn/2-1),-1):
        ck[i] = ck[nn-1-i]
    for i in range(ng-1,-1,-1):
        cg[nn-2-2*i] = cg[i]
        cg[1+2*i] = cg[i]
    for i in range(0,int(nn/2+1)):
        cg[2*i] = 0
    k1 = 0.5*(b-a);
    k2 = 0.5*(b+a);
    intg = 0;
    intk = 0;
    for i in range(0,nn):
        xx=k1*x[i]+k2
        v=fk(f,k,xx)
        intk = intk+v*ck[i];
        if  i%2==1:
            intg = intg+v*cg[i]
    intk = intk*(b-a)*0.5
    intg = intg*(b-a)*0.5
    A[0][0] = abs(intg-intk);
    A[0][1] = intk;
    A[0][2] = a;
    A[0][3] = b;
    toplamhata = A[0][0]
    if toplamhata<eps:
        sonuc = True
        integral = intk
        Aused = 1
        return integral
    Aused = 1
    for h in range(1,n):
        Aused = h+1
        gir(A, h, Aw)
        toplamhata = toplamhata-A[h-1][0]
        ta = A[h-1][2]
        tb = A[h-1][3]
        A[h-1][2] = ta
        A[h-1][3] = 0.5*(ta+tb)
        A[h][2] = 0.5*(ta+tb)
        A[h][3] = tb
        for j in range(h-1,h+1):
            k1 = 0.5*(A[j][3]-A[j][2])
            k2 = 0.5*(A[j][3]+A[j][2])
            intg = 0.0
            intk = 0.0
            for i in range(0,nn):
                xx=k1*x[i]+k2
                v=fk(f,k,xx)
                intk = intk+v*ck[i];
                if  i%2==1:
                    intg = intg+v*cg[i]
            intk = intk*(A[j][3]-A[j][2])*0.5
            intg = intg*(A[j][3]-A[j][2])*0.5
            A[j][0] = abs(intg-intk)
            A[j][1] = intk
            toplamhata = toplamhata+A[j][0]

        cik(A, h-1, Aw);
        cik(A, h, Aw);
        if toplamhata<eps:
            break
    sonuc = toplamhata<eps
    integral = 0
    for j in range(0,Aused):
        integral = integral+A[j][1]
    return integral

def gir(A,n,Awidth):
    i = 0
    p = 0
```

```
    t = 0
    maxcp = 0
    if n==1:
        return
    for i in range(0,Awidth):
        t = A[n-1][i]
        A[n-1][i] = A[0][i]
        A[0][i] = t
    p = 0
    while 2*p+1<n-1:
        maxcp = 2*p+1;
        if 2*p+2<n-1:
            if A[2*p+2][0]>A[2*p+1][0]:
                maxcp = 2*p+2
        if  A[p][0]<A[maxcp][0]:
            for i in range(0,Awidth):
                t = A[p][i]
                A[p][i] = A[maxcp][i]
                A[maxcp][i] = t
            p = maxcp
        else:
            break



def cik(A,n,Awidth):
    i = 0
    p = 0
    t = 0
    kk = 0
    if n==0:
        return
    p = n
    while p!=0:
        kk =int( (p-1)/2)
        if A[p][0]>A[kk][0]:
            for i in range(0,Awidth):
                t = A[p][i];
                A[p][i] = A[kk][i]
                A[kk][i] = t
            p = kk
        else:
            break

def Chebyshev(f1,n,m):
    N=m+n
    Npm=N+m
    ai=[0.0 for i in range(Npm+1)]
    a=[0.0 for i in range(Npm+1)]
    for k in range(Npm+1):
        ai[k]=integral(f,0.0,pi,k,20)
    N=m+n;
    NP1=N+1
    NPm=N+m
    mp1=m+1
    np1=n+1
    for i in range(Npm+1):
        a[i]=ai[i]
    q=[0.0 for i in range(mp1)]
    p=[0.0 for i in range(np1)]
    q[0]=1.0
    b=[[0.0 for i in range(NP1)] for j in range(NP1)]
    c=[0.0 for i in range(NP1)]
    for i in range(NP1):
        for j in range(i+1):
            if j<=n:
                b[i][j]=0.0
        if i<=n:
            b[i][i]=1.0
        for j in range(i+1,np1):
            b[i][j]=0.0
        for j in range(np1,NP1):
            k1=i+j-n
            k2=abs(i-j+n)
            if i!=0:
                b[i][j]=-0.5*(a[k1]+a[k2])
```

```
            else:
                b[i][j]=-0.5*a[j-n]
        if i!=0:
            c[i]=a[i]
        else:
            c[i]=0.5*a[i]
    x=gauss.gauss(b,c)
    for i in range(np1):
        p[i]=x[i]
    for i in range(np1,NP1):
        q[i-n]=x[i]
    aa=[p,q]
    return aa

def func(p,q,x):
    return poly_func(p,x)/poly_func(q,x)


def poly_func(a,x):
    # this function calculates the value of
    # least square curve fitting function
    n=len(a)
    b=0.0
    for k in range(n):
        b=b+a[k]*Ti(x,k)
    return b


def plot_func(f,p,q,a,b):
    nn=101
    xx=[0.0 for j in range(nn)]
    yy=[0.0 for j in range(nn)]
    yy1=[0.0 for j in range(nn)]
    for i in range(nn):
        xx[i]=a+i*(b-a)/(nn-1)
        yy[i]=func(p,q,xx[i])
        yy1[i]=f.func(xx[i])
    plt.title('Chebyshev function approximation')
    plt.xlabel('x ')
    plt.ylabel('y ');
    plt.plot(xx,yy,'k',xx,yy1,'k',linewidth=0.3)

def plot_func_error(f,p,q,a,b):
    nn=101
    xx=[0.0 for j in range(nn)]
    yy=[0.0 for j in range(nn)]
    yy1=[0.0 for j in range(nn)]
    yy2=[0.0 for j in range(nn)]
    for i in range(nn):
        xx[i]=a+i*(b-a)/(nn-1)
        yy[i]=func(p,q,xx[i])
        yy1[i]=f.func(xx[i])
        yy2[i]=yy[i]-yy1[i]
    plt.title('Chebyshev function rational approximation error')
    plt.xlabel('x ')
    plt.ylabel('y ');
    plt.plot(xx,yy2,'k',linewidth=0.3)

class f1(f_x):func=lambda self,x: exp(x)
f = f1()
n=5
m=5
a= Chebyshev(f,n,m)
p=a[0]
q=a[1]
print("p=",p)
print("q=",q)
#plot_func(f,p,q,-1.0,1.0)
plot_func_error(f,p,q,-1.0,1.0)
```





TRIGONOMETRIC FUNCTION (FOURIER) APPROXIMATION

Jean Baptiste Joseph Fourier 1768-1830, French Mathematician

Using of sine and cosine functions to represent data has a long history. For a function in the range of [-π,π] An approximation polynomial can be written as

$S_n(x) = \frac{a_0}{2} + a_n \cos(nx) + \sum_{k=1}^{n-1}[a_k \cos(kx) + b_k \sin(kx)]$ (6.15-1)

Since this set is orthogonal on [-π,π] range with respect to weight function w(x)=1, the selection of coefficients will be:

$$a_k = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\cos(kx)\,dx \quad k = 0,1,2,..,n \qquad (6.15-2)$$

$$b_k = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\sin(kx)\,dx \quad k = 0,1,2,..,n \qquad (6.15-3)$$

The limit of $S_n(x)$ when $n \to \infty$ is called **Fourier series** of function f. A more general definition of the fourier series can be given in the range of [-l,l]

$$S(x) = \frac{a_0}{2} + \sum_{k=1}^{\infty}\left[a_k \cos\left(\frac{k\pi x}{l}\right) + b_k \sin\left(\left(\frac{k\pi x}{l}\right)\right)\right] \qquad (6.15-4)$$

$$a_k = \frac{1}{l}\int_{-l}^{l} f(x)\cos\left(\frac{k\pi x}{l}\right)dx = \frac{1}{l}\int_{\alpha}^{\alpha+2l} f(x)\cos\left(\frac{k\pi x}{l}\right)dx \quad k = 0,1,2,..,n \quad (6.15-5)$$

$$b_k = \frac{1}{l}\int_{-l}^{l} f(x)\sin\left(\frac{k\pi x}{l}\right)dx = \frac{1}{l}\int_{\alpha}^{\alpha+2l} f(x)\sin\left(\frac{k\pi x}{l}\right)dx \quad k = 0,1,2,..,n \quad (6.15-6)$$

A Computer program developed to calculate Trigonometric polynomials. Integrations are again taken numerically in this program similar to Chebyschev polynomial rational approximations. The first sample function to be investigated will be **f(x)=|x|**

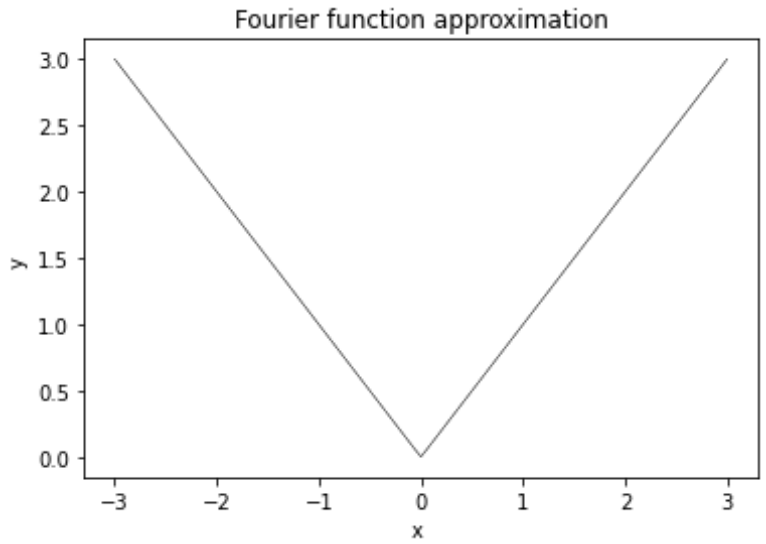
```
 from math import *
from f_x import *
import gauss
import numpy as np;
import matplotlib.pyplot as plt;
import os;


def fak(f,k,x):
    a=0
    mult=1.0/pi
    a=mult*f.func(x)*cos(k*x)
    return a

def fbk(f,k,x):
    a=0
    mult=1.0/pi
    a=mult*f.func(x)*sin(k*x)
    return a

#def gauss_kronrod_integral(f,x1,x2,k,n,eqn_type):
def integral(ff,a,b,k,n,eqn_type):
    # eps hata miktarı
    eps=1.0e-10
    Aused=0
    integral=0
    toplamhata = 0
    nn = 0
    ng = 0
    i = 0
    j = 0
    h = 0
    v = 0
    k1 = 0
    k2 = 0
    intg = 0
    intk = 0
    ta = 0
    tb = 0
    Aw = 4
```

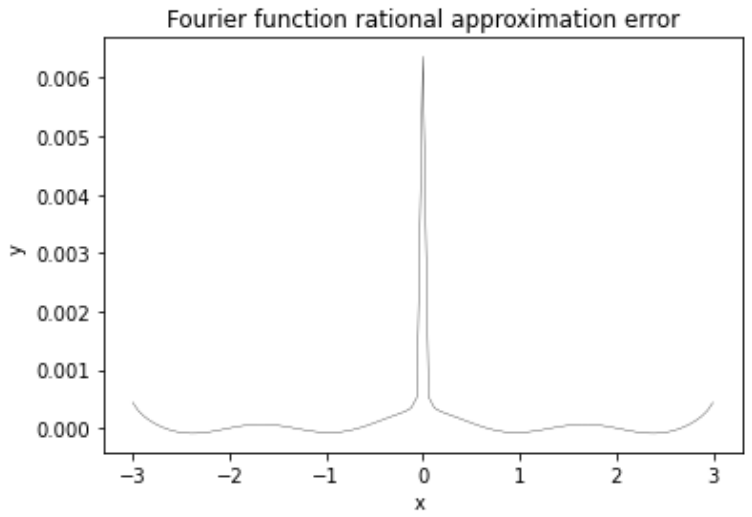
```
A = [[0.0 for i in range(Aw-1+1)] for j in range(n-1+1)]    #[n-1+1][Aw-1+1];
nn = 61;
ng = 15;
x =[0.0 for i in range(nn)]
ck =[0.0 for i in range(nn)]
cg =[0.0 for i in range(nn)]
#Gauss coefficients
cg[0] = 0.007968192496166605615465883474674
cg[1] = 0.018466468311090959142302131912047
cg[2] = 0.028784707883323369349719179611292
cg[3] = 0.038799192569627049596801936446348
cg[4] = 0.048402672830594052902938140422808
cg[5] = 0.057493156217619066481721689402056
cg[6] = 0.065974229882180495128128515115962
cg[7] = 0.073755974737705206268243850022191
cg[8] = 0.080755895229420215354694938460530
cg[9] = 0.086899787201082979802387530715126
cg[10] = 0.092122522237786128717632707087619
cg[11] = 0.096368737174644259639468626351810
cg[12] = 0.099593420586795267062780282103569
cg[13] = 0.101762389748405504596428952168554
cg[14] = 0.102852652893558840341285636705415
x[0] = 0.999484410050490637571325895705811
x[1] = 0.996893484074649540271630050918695
x[2] = 0.991630996870404594858628366109486
x[3] = 0.983668123279747209970032581605663
x[4] = 0.973116322501126268374693868423707
x[5] = 0.960021864968307512216871025581798
x[6] = 0.944374444748559979415831324037439
x[7] = 0.926200047429274325879324277080474
x[8] = 0.905573307699907798546522558925958
x[9] = 0.882560535792052681543116462530226
x[10] = 0.857205233546061098958658510658944
x[11] = 0.829565762382768397442898119732502
x[12] = 0.799727835821839083013668942322683
x[13] = 0.767777432104826194917977340974503
x[14] = 0.733790062453226804726171131369528
x[15] = 0.697850494793315796932292388026640
x[16] = 0.660061064126626961370053668149271
x[17] = 0.620526182989242861140477556431189
x[18] = 0.579345235826361691756024932172540
x[19] = 0.536624148142019899264169793311073
x[20] = 0.492480467861778574993693061207709
x[21] = 0.447033769538089176780609900322854
x[22] = 0.400401254830394392535476211542661
x[23] = 0.352704725530878113471037207089374
x[24] = 0.304073202273625077372677107199257
x[25] = 0.254636926167889846439805129817805
x[26] = 0.204525116682309891438957671002025
x[27] = 0.153869913608583546963794672743256
x[28] = 0.102806937966737030147096751318001
x[29] = 0.051471842555317695833025213166723
x[30] = 0.000000000000000000000000000000000
#Kronrod coefficients
ck[0] = 0.001389013698677007624551591226760
ck[1] = 0.003890461127099884051267201844516
ck[2] = 0.006630703915931292173319826369750
ck[3] = 0.009273279659517763428441146892024
ck[4] = 0.011823015253496341742232898853251
ck[5] = 0.014369729507045804812451432443580
ck[6] = 0.016920889189053272627572289420322
ck[7] = 0.019414141193942381173408951050128
ck[8] = 0.021828035821609192297167485738339
ck[9] = 0.024191162078080601365686370725232
ck[10] = 0.026509954882333101610601709335075
ck[11] = 0.028754048765041292843978785354334
ck[12] = 0.030907257562387762472884252943092
ck[13] = 0.032981447057483726031814191016854
ck[14] = 0.034979338028060024137499670731468
ck[15] = 0.036882364651821229223911065617136
ck[16] = 0.038678945624727592950348651532281
ck[17] = 0.040374538951535959111995279752468
ck[18] = 0.041969810215164246147147541285970
ck[19] = 0.043452539701356069316831728117073
ck[20] = 0.044814800133162663192355551616723
ck[21] = 0.046059238271006988116271735559374
```

```
ck[22] = 0.047185546569299153945261478181099
ck[23] = 0.048185861757087129140779492298305
ck[24] = 0.049055434555029778887528165367238
ck[25] = 0.049795683427074206357811569379942
ck[26] = 0.050405921402782346840893085653585
ck[27] = 0.050881795898749606492297473049805
ck[28] = 0.051221547849258772170656282604944
ck[29] = 0.051426128537459025933862879215781
ck[30] = 0.051494729429451567558340433647099
for i in range(nn-1,int(nn/2-1),-1):
    x[i] = -x[nn-1-i]
for i in range(nn-1,int(nn/2-1),-1):
    ck[i] = ck[nn-1-i]
for i in range(ng-1,-1,-1):
    cg[nn-2-2*i] = cg[i]
    cg[1+2*i] = cg[i]
for i in range(0,int(nn/2+1)):
    cg[2*i] = 0
k1 = 0.5*(b-a);
k2 = 0.5*(b+a);
intg = 0;
intk = 0;
for i in range(0,nn):
    xx=k1*x[i]+k2
    if eqn_type==0:
        v=fak(ff,k,xx)
    else:
        v=fbk(ff,k,xx)
    intk = intk+v*ck[i];
    if  i%2==1:
        intg = intg+v*cg[i]
intk = intk*(b-a)*0.5
intg = intg*(b-a)*0.5
A[0][0] = abs(intg-intk);
A[0][1] = intk;
A[0][2] = a;
A[0][3] = b;
toplamhata = A[0][0]
if toplamhata<eps:
    sonuc = True
    integral = intk
    Aused = 1
    return integral
Aused = 1
for h in range(1,n):
    Aused = h+1
    gir(A, h, Aw)
    toplamhata = toplamhata-A[h-1][0]
    ta = A[h-1][2]
    tb = A[h-1][3]
    A[h-1][2] = ta
    A[h-1][3] = 0.5*(ta+tb)
    A[h][2] = 0.5*(ta+tb)
    A[h][3] = tb
    for j in range(h-1,h+1):
        k1 = 0.5*(A[j][3]-A[j][2])
        k2 = 0.5*(A[j][3]+A[j][2])
        intg = 0.0
        intk = 0.0
        for i in range(0,nn):
            xx=k1*x[i]+k2
            if eqn_type==0:
                v=fak(ff,k,xx)
            else:
                v=fbk(ff,k,xx)
            intk = intk+v*ck[i];
            if  i%2==1:
                intg = intg+v*cg[i]
        intk = intk*(A[j][3]-A[j][2])*0.5
        intg = intg*(A[j][3]-A[j][2])*0.5
        A[j][0] = abs(intg-intk)
        A[j][1] = intk
        toplamhata = toplamhata+A[j][0]

    cik(A, h-1, Aw);
    cik(A, h, Aw);
```

```
        if toplamhata<eps:
            break
    sonuc = toplamhata<eps
    integral = 0
    for j in range(0,Aused):
        integral = integral+A[j][1]
    return integral

def gir(A,n,Awidth):
    i = 0
    p = 0
    t = 0
    maxcp = 0
    if n==1:
        return
    for i in range(0,Awidth):
        t = A[n-1][i]
        A[n-1][i] = A[0][i]
        A[0][i] = t
    p = 0
    while 2*p+1<n-1:
        maxcp = 2*p+1;
        if 2*p+2<n-1:
            if A[2*p+2][0]>A[2*p+1][0]:
                maxcp = 2*p+2
        if  A[p][0]<A[maxcp][0]:
            for i in range(0,Awidth):
                t = A[p][i]
                A[p][i] = A[maxcp][i]
                A[maxcp][i] = t
            p = maxcp
        else:
            break



def cik(A,n,Awidth):
    i = 0
    p = 0
    t = 0
    kk = 0
    if n==0:
        return
    p = n
    while p!=0:
        kk =int( (p-1)/2)
        if A[p][0]>A[kk][0]:
            for i in range(0,Awidth):
                t = A[p][i];
                A[p][i] = A[kk][i]
                A[kk][i] = t
            p = kk
        else:
            break

def Trigonometric(f1,n):
    a=[0.0 for i in range(n+1)]
    b=[0.0 for i in range(n+1)]
    for k in range(n):
        a[k]=integral(f,-pi,pi,k,80,0)
    b[0]=0.0
    for k in range(1,n):
        b[k]=integral(f,-pi,pi,k,80,1)
    aa=[a,b]
    return aa


def func(ak,bk,x):
    n=len(a)-1
    cc=ak[0]/2.0+ak[n]*cos(n*x)+bk[n]*sin(n*x)
    for k in range(1,n):
        cc=cc+ak[k]*cos(k*x)+bk[k]*sin(k*x)
    return cc


def plot_func(f,ak,bk,xmin,xmax):
```

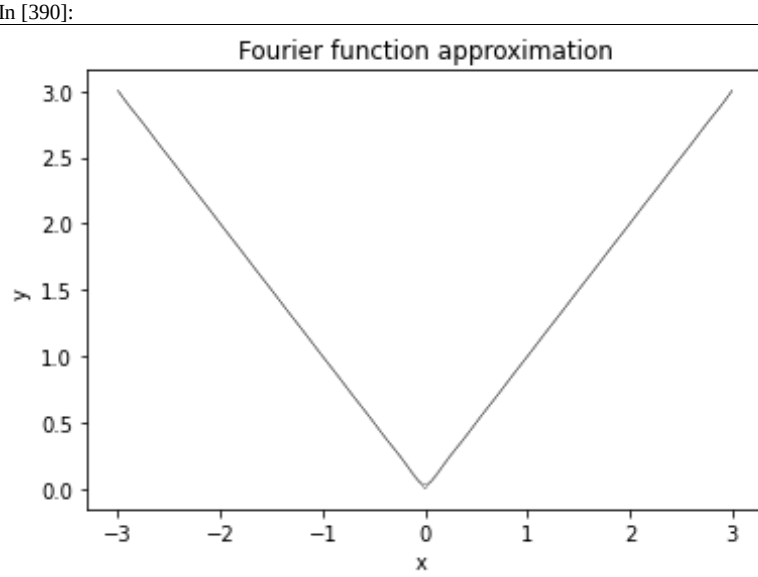
```
    nn=101
    xx=[0.0 for j in range(nn)]
    yy=[0.0 for j in range(nn)]
    yy1=[0.0 for j in range(nn)]
    for i in range(nn):
        xx[i]=xmin+i*(xmax-xmin)/(nn-1)
        yy[i]=func(ak,bk,xx[i])
        yy1[i]=f.func(xx[i])
    plt.title('Fourier function approximation')
    plt.xlabel('x ')
    plt.ylabel('y ');
    plt.plot(xx,yy,'k',xx,yy1,'k',linewidth=0.3)

def plot_func_error(f,ak,bk,xmin,xmax):
    nn=101
    xx=[0.0 for j in range(nn)]
    yy=[0.0 for j in range(nn)]
    yy1=[0.0 for j in range(nn)]
    yy2=[0.0 for j in range(nn)]
    for i in range(nn):
        xx[i]=xmin+i*(xmax-xmin)/(nn-1)
        yy[i]=func(ak,bk,xx[i])
        yy1[i]=f.func(xx[i])
        yy2[i]=yy[i]-yy1[i]
    plt.title('Fourier function rational approximation error')
    plt.xlabel('x ')
    plt.ylabel('y ');
    plt.plot(xx,yy2,'k',linewidth=0.3)

#main method
class f1(f_x):func=lambda self,x: abs(x)
f = f1()
n=100
c= Trigonometric(f1,n)
a=c[0]
b=c[1]
print("a=",a)
print("b=",b)
#plot_func(f,a,b,-3.0,3.0)
plot_func_error(f,a,b,-3.0,3.0)
```

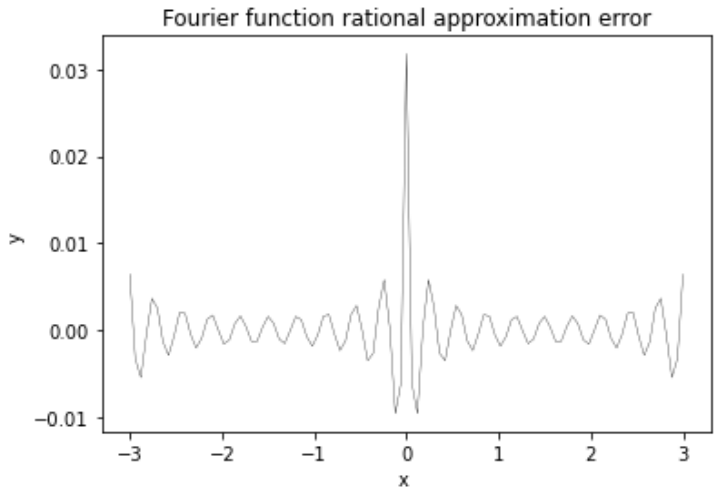




The same code for n=20

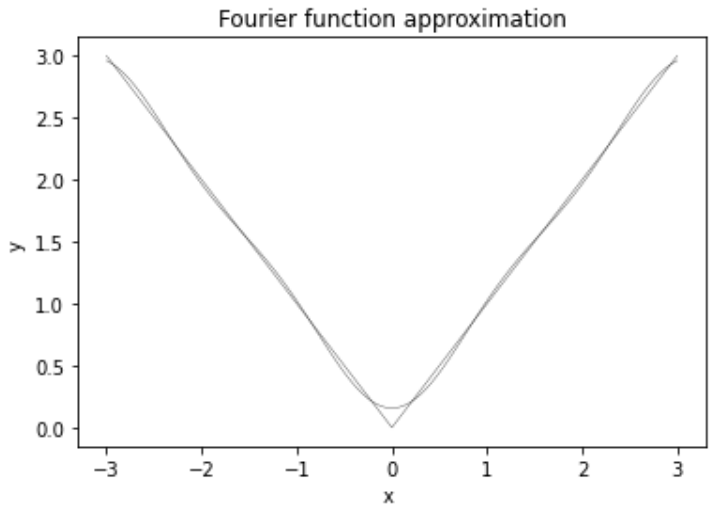
```
runfile('E:/okul/SCO1/Fourier1.py', wdir='E:/okul/SCO1')
Reloaded modules: f_x, gauss
a= [3.141592653589794, -1.273239544735162, -2.7107671815768533e-16, -0.1414710605261289, -1.8670427590337967e-16, -
0.050929581789406306, -4.750994924134084e-16, -0.02598448050479918, 8.009172245384794e-16, -0.015719006725126296,
7.840213314075715e-16, -0.0105226408655813, 1.3298597987703044e-15, -0.007533961803167354, 3.7682465225866257e-17, -
0.005658842421045016, -1.6358040255716413e-16, -0.004405673165172949, -5.156348339343926e-16, -0.003526979348298361, 0.0]
b= [0.0, -1.2266295340286705e-16, 2.190295837944721e-17, -2.516006214040498e-17, 5.3465464677421137e-17, -
9.341288433387376e-17, -2.6903038417714896e-17, -5.4976931128584544e-17, 7.914975021726131e-17, -7.355093786999404e-18,
1.2528140936755858e-17, -8.941069147727206e-18, -3.8638191674106856e-18, 3.5125628794642595e-17, 7.1975606639204e-17, -
3.246459631019997e-17, -1.4688899314123266e-18, 1.745637309794359e-18, 3.929812773024862e-17, 2.6844495703057158e-17,
0.0]

In [390]:
```

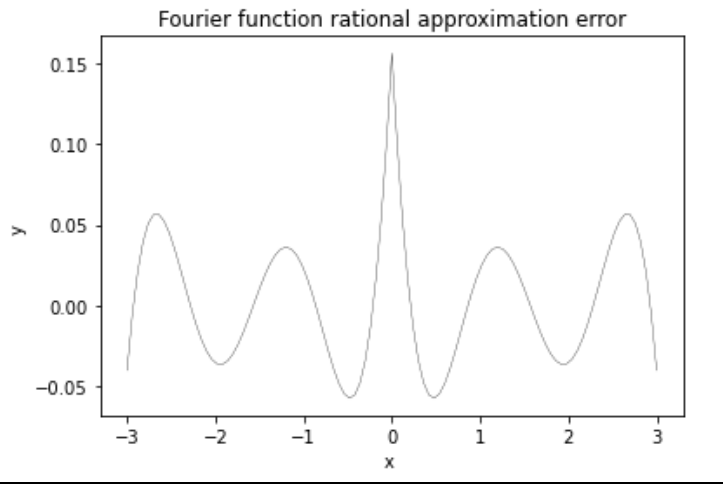




The same code **n=5**

```
runfile('E:/okul/SCO1/Fourier1.py', wdir='E:/okul/SCO1')
Reloaded modules: f_x, gauss
a= [3.141592653589794, -1.273239544735162, -2.7107671815768533e-16, -0.1414710605261289, -1.8670427590337967e-16, 0.0]
b= [0.0, -1.2266295340286705e-16, 2.190295837944721e-17, -2.516006214040498e-17, 5.3465464677421137e-17, 0.0]
```



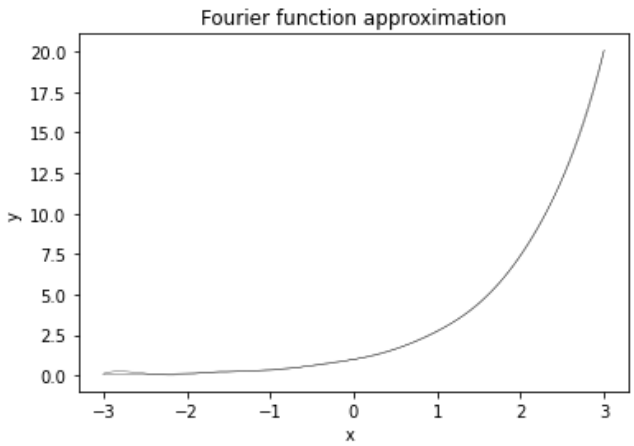


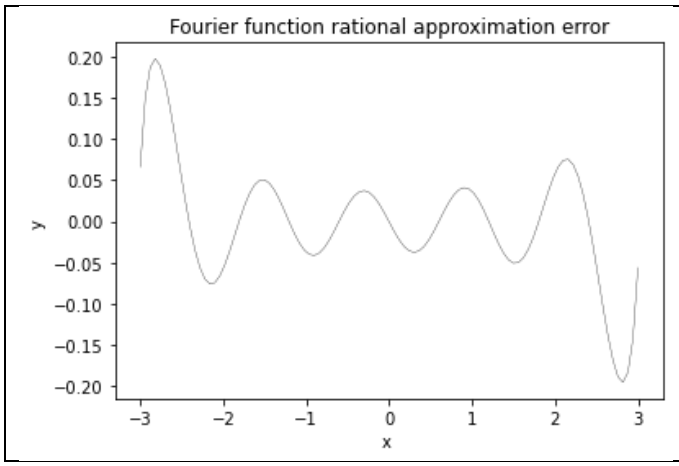
$$f(x) = \exp(x) \qquad -\pi \leq x \leq \pi$$

```
#main method
class f1(f_x):func=lambda self,x: exp(x)
f = f1()
```
n=100

```
c= Trigonometric(f1,n)
a=c[0]
b=c[1]
print("a=",a)
print("b=",b)
plot_func(f,a,b,-3.0,3.0)
#plot_func_error(f,a,b,-3.0,3.0)
```

```
runfile('E:/okul/SCO1/Fourier1.py', wdir='E:/okul/SCO1')
Reloaded modules: f_x, gauss
a= [7.352155820749955, -3.67607791037498, 1.4704311641499905, -0.7352155820749955, 0.43247975416176276, -
0.28277522387500004, 0.19870691407432245, -0.14704311641500056, 0.11311008954999914, -0.08966043683841284,
0.0727936219876225, -0.06026357230123277, 0.0507045229017273, -0.04324797541617821, 0.03732058792258898, -
0.032531662923671906, 0.02860761019747006, -0.025352261450860388, 0.022622017909999115, -0.020309822709257554,
0.018334553168956734, -0.016633836698530604, 0.015159084166494061, -0.013871992114626861, 0.012742037817591442, -
0.011744657860619522, 0.01085990520051472, -0.010071446329789856, 0.009365803593307787, -0.008731776509201442,
0.0081599953615404, -0.007642573618246035, 0.00717283494707311, -0.006745097083255873, 0.006354499412920247, -
0.005996864454120 89, 0.0056685858294168836, -0.005366537095437009, 0.005087997107792295, -0.0048305885813101634,
0.0045922272459486685, -0.00437107956049699, 0.004165527377195971, -0.003974138281486239, 0.003795640588924811, -
0.003628902182003617, 0.0034729125275103534, -0.0033267673397021184, 0.003189655453685236, -0.003060847552351006,
0.0029396864537090035, -0.002825578716662453, 0.002717987364418818, -0.0026164255589853566, 0.002520451086989418, -
0.002429661540238441, 0.00234369092685154, -0.0022622017910041508, 0.002184890288484924, -0.002111474962879117,
0.0020416983673232423, -0.0019753239711816214, 0.0019121341536377761, -0.001851928418329016, 0.001794521801499975, -
0.0017397434502452241, 0.0016874353501862226, -0.0016374511850193515, 0.0015896553125870978, -0.00154392184392459,
0.0015001338136586759, -0.0014581824317182407, 0.0014179664070890373, -0.0013793913359796686, 0.0013423691474822152, -
0.0013068176005610495, 0.0012726598270286354, -0.001239823915808499, 0.0012082425342171531, -0.001177852582619713,
0.0011485948790424915, -0.0011204138708810047, 0.0010932573711091432, -0.0010670763165098066, 0.0010418245459522, -
0.0010174585968431293, 0.0009939375180135802, -0.0009712226975722715, 0.0009492777044191383, -0.0009280681419811046,
0.0009075615134833872, -0.0008877270974096871, 0.0008685358323403199, -0.0008499602104928854, 0.0008319741791090257, -
0.00081453049055726, 0.0007976734100839171, -0.0007813130521535633, 0.0007654508923226556, -0.0007500669068287023,
0.0]
b= [0.0, 3.676077910374976, -2.940862328299981, 2.2056467462249865, -1.7299190166470486, 1.4138761193749914, -
1.1922414844459388, 1.0293018149049964, -0.9048807163999978, 0.8069439315457285, -0.7279362198762359,
0.6628992953135238, -0.6084542748206893, 0.5622236804102954, -0.5224882309162445, 0.48797494385508544, -
0.4577217631595316, 0.4309884446646532, -0.407196322379997, 0.38588663147582736, -0.3666910633790542,
0.3493105706691178, -0.3334998516628841, 0.319055818636317, -0.3058089076221852, 0.29361644651557767, -
0.28235753521344326, 0.27192905090445624, -0.2622425006127418, 0.2532215187669232, -0.2447998608462777,
0.23691978216554024, -0.22953071830633853, 0.22258820374747257, -0.2160529800393219, 0.2098902558941693, -
0.20406908985890657, 0.19856187253120167, -0.19334389009584493, 0.18839295467099249, -0.18368908983760984,
0.17921426198023674, -0.1749521498422164, 0.17088794610391939, -0.16700818591275002, 0.1633005981903995, -
0.1597539762657026, 0.1563580649661767, -0.15310346177700004, 0.14998153006525558, -0.14698432268592482,
0.14410451454966872, -0.14133534294972228, 0.13867055462624236, -0.13610435869746307, 0.13363138471290298, -
0.13124664519029758, 0.1289455020869876, -0.12672363673209422, 0.12457702280995349, -0.12250190203970737,
0.12049476224227897, -0.11855231752574864, 0.11667149035447234, -0.11484939529606882, 0.11308332426614214, -
0.11137073311211865, 0.10970922939648936, -0.10809656125642796, 0.10653060723052209, -0.10500936695623353,
0.10353095265236686, -0.1020935813103159, 0.1006955675262201, -0.09933531691354529, 0.09801132004199076, -
0.09672214685425415, 0.09546644151732593, -0.09424291766943053, 0.09305035402743556, -0.09188759032338371,
0.09075352354171745, -0.0896471044314444, 0.08856733427028177, -0.08751326185957514, 0.08648398073121538, -
0.08547862654921537, 0.08449637469025573, -0.08353643798915587, 0.08259806463604757, -0.0816805362137403,
0.08078316586431603, -0.07990529657519269, 0.07904629957569305, -0.0782055728358617, 0.07738253966001056, -
0.07657664736811726, 0.07578736605873779, -0.0750141874475188, 0.07425662377619346, 0.0]
```





If the function is defined in some other interval than [-π,π] then a range correction should apply
For example for the range $[x_0, x_n]$ ranges can be converted by using transfer functions

$$z = \frac{2\pi(x-x_0)}{(x_n-x_0)} - \pi \quad (6.15\text{-}12)$$

$$x = \frac{(x_n-x_0)(x+\pi)}{2\pi} + x_0 \quad (6.15\text{-}13)$$

```
 from math import *
from f_x import *
import gauss
import numpy as np;
import matplotlib.pyplot as plt;
import os;



def read_array2D(d):
    path = d
    file1 = open(path,'r')
    file1.seek(0,2)  #jumps to the end
    endLocation=file1.tell()
    file1.seek(0) #jumps to the start
    nn=0
    m=0
    while True:
        line=file1.readline()
        ee=line.split()
        nn+=1
        if  file1.tell()==endLocation:
            break
    m=len(ee)
    print("m=",m)
    a=[[0.0 for i in range(nn)] for j in range(m)]
    file1.seek(0) #jumps to the start
    for i in range(nn):
        line=file1.readline()
        ee=line.split()
        for j in range(m):
            a[j][i]=float(ee[j])
    file1.close()
    return a
# Fourier
def fak(f,k,x):
    a=0
    mult=1.0/pi
    a=mult*f.func(x)*cos(k*x)
    return a

def fbk(f,k,x):
    a=0
    mult=1.0/pi
    a=mult*f.func(x)*sin(k*x)
    return a

#def gauss_kronrod_integral(f,x1,x2,k,n,eqn_type):
```

```
def integral(ff,a,b,k,n,eqn_type):
    # eps hata miktarı
    eps=1.0e-10
    Aused=0
    integral=0
    toplamhata = 0
    nn = 0
    ng = 0
    i = 0
    j = 0
    h = 0
    v = 0
    k1 = 0
    k2 = 0
    intg = 0
    intk = 0
    ta = 0
    tb = 0
    Aw = 4
    A = [[0.0 for i in range(Aw-1+1)] for j in range(n-1+1)]    #[n-1+1][Aw-1+1];
    nn = 61;
    ng = 15;
    x =[0.0 for i in range(nn)]
    ck =[0.0 for i in range(nn)]
    cg =[0.0 for i in range(nn)]
    #Gauss coefficients
    cg[0] = 0.007968192496166605615465883474674
    cg[1] = 0.018466468311090959142302131912047
    cg[2] = 0.028784707883323369349719179611292
    cg[3] = 0.038799192569627049596801936446348
    cg[4] = 0.048402672830594052902938140422808
    cg[5] = 0.057493156217619066481721689402056
    cg[6] = 0.065974229882180495128128515115962
    cg[7] = 0.073755974737705206268243850022191
    cg[8] = 0.080755895229420215354694938460530
    cg[9] = 0.086899787201082979802387530715126
    cg[10] = 0.092122522237786128717632707087619
    cg[11] = 0.096368737174644259639468626351810
    cg[12] = 0.099593420586795267062780282103569
    cg[13] = 0.101762389748405504596428952168554
    cg[14] = 0.102852652893558840341285636705415
    x[0] = 0.999484410050490637571325895705811
    x[1] = 0.996893484074649540271630050918695
    x[2] = 0.991630996870404594858628366109486
    x[3] = 0.983668123279747209970032581605663
    x[4] = 0.973116322501126268374693868423707
    x[5] = 0.960021864968307512216871025581798
    x[6] = 0.944374444748559979415831324037439
    x[7] = 0.926200047429274325879324277080474
    x[8] = 0.905573307699907798546522558925958
    x[9] = 0.882560535792052681543116462530226
    x[10] = 0.857205233546061098958658510658944
    x[11] = 0.829565762382768397442898119732502
    x[12] = 0.799727835821839083013668942322683
    x[13] = 0.767777432104826194917977340974503
    x[14] = 0.733790062453226804726171131369528
    x[15] = 0.697850494793315796932292388026640
    x[16] = 0.660061064126626961370053668149271
    x[17] = 0.620526182989242861140477556431189
    x[18] = 0.579345235826361691756024932172540
    x[19] = 0.536624148142019899264169793311073
    x[20] = 0.492480467861778574993693061207709
    x[21] = 0.447033769538089176780609900322854
    x[22] = 0.400401254830394392535476211542661
    x[23] = 0.352704725530878113471037207089374
    x[24] = 0.304073202273625077372677107199257
    x[25] = 0.254636926167889846439805129817805
    x[26] = 0.204525116682309891438957671002025
    x[27] = 0.153869913608583546963794672743256
    x[28] = 0.102806937966737030147096751318001
    x[29] = 0.051471842555317695833025213166723
    x[30] = 0.000000000000000000000000000000000
    #Kronrod coefficients
    ck[0] = 0.001389013698677007624551591226760
    ck[1] = 0.003890461127099884051267201844516
    ck[2] = 0.006630703915931292173319826369750
```

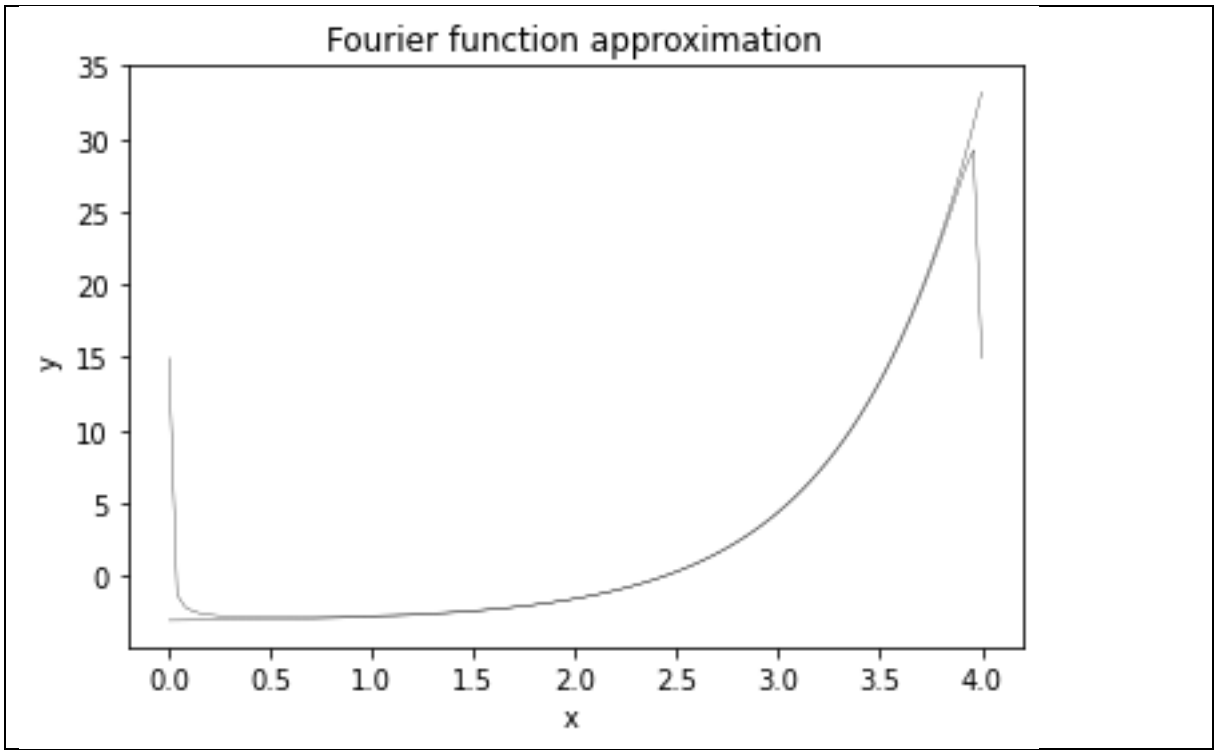
```
ck[3] = 0.009273279659517763428441146892024
ck[4] = 0.011823015253496341742232898853251
ck[5] = 0.014369729507045804812451432443580
ck[6] = 0.016920889189053272627572289420322
ck[7] = 0.019414141193942381173408951050128
ck[8] = 0.021828035821609192297167485738339
ck[9] = 0.024191162078080601365686370725232
ck[10] = 0.026509954882333101610601709335075
ck[11] = 0.028754048765041292843978785354334
ck[12] = 0.030907257562387762472884252943092
ck[13] = 0.032981447057483726031814191016854
ck[14] = 0.034979338028060024137499670731468
ck[15] = 0.036882364651821229223911065617136
ck[16] = 0.038678945624727592950348651532281
ck[17] = 0.040374538951535959111995279752468
ck[18] = 0.041969810215164246147147541285970
ck[19] = 0.043452539701356069316831728117073
ck[20] = 0.044814800133162663192355551616723
ck[21] = 0.046059238271006988116271735559374
ck[22] = 0.047185546569299153945261478181099
ck[23] = 0.048185861757087129140779492298305
ck[24] = 0.049055434555029778887528165367238
ck[25] = 0.049795683427074206357811569379942
ck[26] = 0.050405921402782346840893085653585
ck[27] = 0.050881795898749606492297473049805
ck[28] = 0.051221547849258772170656282604944
ck[29] = 0.051426128537459025933862879215781
ck[30] = 0.051494729429451567558340433647099
for i in range(nn-1,int(nn/2-1),-1):
    x[i] = -x[nn-1-i]
for i in range(nn-1,int(nn/2-1),-1):
    ck[i] = ck[nn-1-i]
for i in range(ng-1,-1,-1):
    cg[nn-2-2*i] = cg[i]
    cg[1+2*i] = cg[i]
for i in range(0,int(nn/2+1)):
    cg[2*i] = 0
k1 = 0.5*(b-a);
k2 = 0.5*(b+a);
intg = 0;
intk = 0;
for i in range(0,nn):
    xx=k1*x[i]+k2
    if eqn_type==0:
        v=fak(ff,k,xx)
    else:
        v=fbk(ff,k,xx)
    intk = intk+v*ck[i];
    if  i%2==1:
        intg = intg+v*cg[i]
intk = intk*(b-a)*0.5
intg = intg*(b-a)*0.5
A[0][0] = abs(intg-intk);
A[0][1] = intk;
A[0][2] = a;
A[0][3] = b;
toplamhata = A[0][0]
if toplamhata<eps:
    sonuc = True
    integral = intk
    Aused = 1
    return integral
Aused = 1
for h in range(1,n):
    Aused = h+1
    gir(A, h, Aw)
    toplamhata = toplamhata-A[h-1][0]
    ta = A[h-1][2]
    tb = A[h-1][3]
    A[h-1][2] = ta
    A[h-1][3] = 0.5*(ta+tb)
    A[h][2] = 0.5*(ta+tb)
    A[h][3] = tb
    for j in range(h-1,h+1):
        k1 = 0.5*(A[j][3]-A[j][2])
        k2 = 0.5*(A[j][3]+A[j][2])
```

```
            intg = 0.0
            intk = 0.0
            for i in range(0,nn):
                xx=k1*x[i]+k2
                if eqn_type==0:
                    v=fak(ff,k,xx)
                else:
                    v=fbk(ff,k,xx)
                intk = intk+v*ck[i];
                if  i%2==1:
                    intg = intg+v*cg[i]
            intk = intk*(A[j][3]-A[j][2])*0.5
            intg = intg*(A[j][3]-A[j][2])*0.5
            A[j][0] = abs(intg-intk)
            A[j][1] = intk
            toplamhata = toplamhata+A[j][0]

        cik(A, h-1, Aw);
        cik(A, h, Aw);
        if toplamhata<eps:
            break
    sonuc = toplamhata<eps
    integral = 0
    for j in range(0,Aused):
        integral = integral+A[j][1]
    return integral

def gir(A,n,Awidth):
    i = 0
    p = 0
    t = 0
    maxcp = 0
    if n==1:
        return
    for i in range(0,Awidth):
        t = A[n-1][i]
        A[n-1][i] = A[0][i]
        A[0][i] = t
    p = 0
    while 2*p+1<n-1:
        maxcp = 2*p+1;
        if 2*p+2<n-1:
            if A[2*p+2][0]>A[2*p+1][0]:
                maxcp = 2*p+2
        if  A[p][0]<A[maxcp][0]:
            for i in range(0,Awidth):
                t = A[p][i]
                A[p][i] = A[maxcp][i]
                A[maxcp][i] = t
            p = maxcp
        else:
            break



def cik(A,n,Awidth):
    i = 0
    p = 0
    t = 0
    kk = 0
    if n==0:
        return
    p = n
    while p!=0:
        kk =int( (p-1)/2)
        if A[p][0]>A[kk][0]:
            for i in range(0,Awidth):
                t = A[p][i];
                A[p][i] = A[kk][i]
                A[kk][i] = t
            p = kk
        else:
            break

def Trigonometric(f,n):
    a=[0.0 for i in range(n+1)]
```

```
    b=[0.0 for i in range(n+1)]
    for k in range(n):
        a[k]=integral(f,-pi,pi,k,80,0)
    b[0]=0.0
    for k in range(1,n):
        b[k]=integral(f,-pi,pi,k,80,1)
    aa=[a,b]
    return aa


def func(ak,bk,x):
    n=len(a)-1
    cc=ak[0]/2.0+ak[n]*cos(n*x)+bk[n]*sin(n*x)
    for k in range(1,n):
        cc=cc+ak[k]*cos(k*x)+bk[k]*sin(k*x)
    return cc


def plot_func(f,ak,bk,xmin,xmax):
    nn=101
    xx=[0.0 for j in range(nn)]
    zz=[0.0 for j in range(nn)]
    yy=[0.0 for j in range(nn)]
    yy1=[0.0 for j in range(nn)]
    for i in range(nn):
        xx[i]=xmin+i*(xmax-xmin)/(nn-1)
        zz[i]=z_x(xmin,xmax,xx[i])
        yy[i]=func(ak,bk,zz[i])
        yy1[i]=f.func(zz[i])
    plt.title('Fourier function approximation')
    plt.xlabel('x ')
    plt.ylabel('y ');
    plt.plot(xx,yy,'k',xx,yy1,'k',linewidth=0.3)

def plot_func_error(f,ak,bk,xmin,xmax):
    nn=101
    xx=[0.0 for j in range(nn)]
    yy=[0.0 for j in range(nn)]
    zz=[0.0 for j in range(nn)]
    yy1=[0.0 for j in range(nn)]
    yy2=[0.0 for j in range(nn)]
    for i in range(nn):
        xx[i]=xmin+i*(xmax-xmin)/(nn-1)
        zz[i]=z_x(xmin,xmax,xx[i])
        yy[i]=func(ak,bk,zz[i])
        yy1[i]=f.func(zz[i])
        yy2[i]=yy[i]-yy1[i]
    plt.title('Fourier function rational approximation error')
    plt.xlabel('x ')
    plt.ylabel('y ');
    plt.plot(xx,yy2,'k',linewidth=0.3)

#conversion function z=f(x)    x0<x<xn   -pi<z<pi
def  z_x(x0,xn,x):
    zz=2.0*pi*(x-x0)/(xn-x0)-pi;
    return zz

#conversion function x=f(z)    x0<x<xn   -pi<z<pi
def x_z(x0,xn,z):
    xx=(z+pi)/(2.0*pi)*(xn-x0)+x0
    return xx

def set_z_x(x0,xn,x):
    n=len(x)
    z=[0.0 for j in range(n)]
    for j in range(n):
        z[j]=z_x(x0,xn,x[j])
    return z

def set_x_z(x0,xn,z):
    n=len(z)
    x=[0.0 for j in range(n)]
    for j in range(n):
        x[j]=x_z(x0,xn,z[j])
    return x
```

```
#main method
xmin=0.0
xmax=4.0
class f2(f_x):func=lambda self,x:z_x(xmin,xmax,exp(x))
f = f2()
n=100
c= Trigonometric(f,n)
print("Fourier Function approximation n = ",n)
a=c[0]
b=c[1]
print("a = ",a)
print("b = ",a)

plot_func(f,a,b,xmin,xmax)
#plot_func_error(f,a,b,1.0,3.0)
```

```
runfile('E:/okul/SCO1/Fourier2a.py', wdir='E:/okul/SCO1')
Reloaded modules: f_x, gauss
Fourier Function approximation n =  100
a =  [5.265554050078161, -5.774369678628878, 2.3097478714515502, -1.1548739357257736, 0.6793376092504566, -
0.4441822829714538, 0.312128090736695, -0.23097478714515646, 0.1776729131885801, -0.14083828484460464,
0.11434395403225395, -0.09466179801031344, 0.07964647832591897, -0.06793376092504746, 0.05862304242262804, -
0.05110061662503418, 0.044936729016566215, -0.03982323916295617, 0.035534582637716, -0.031902594909555125,
0.02879984877122056, -0.02612836958655715, 0.023811833726305378, -0.021790074258981537, 0.020015146199752917, -
0.018448465426925263, 0.017058699198310517, -0.01582019090034844, 0.01471176988185205, -0.01371584246704926,
0.01281769074057233, -0.01200492656679935, 0.011267062787568307, -0.010595173722253992, 0.009981624336435412, -
0.00941985265681999, 0.008904193798969366, -0.008429736757121505, 0.007992207167661402, -0.007587870799778067,
0.007213453689738906, -0.006866075717754536, 0.00654319510326376, -0.006242561814732761, 0.005962178294917012, -
0.005700266217791001, 0.005455238241496265, -0.0052256739173065034, 0.005010299070390174, -0.004807968092112336,
0.004617648683418303, -0.004438408669202221, 0.0042694045683033975, -0.004109871657386625, 0.003959115309309038, -
0.0038165034227602354, 0.0036814597887357865, -0.0035534582637771605, 0.003432017639600842, -0.003316697115810202,
0.003207092295817246, -0.0031028316381630523, 0.003003573304874352, -0.002909002356996015, 0.0028188282541496065, -
0.0027327826212125046, 0.0026506172497755016, -0.0025721023067357155, 0.002497024725885095, -0.0024251867612929204,
0.0023564046841970023, -0.002290507607541411, 0.002227336423772561, -0.00216674284376745, 0.0021085885260662706, -
0.0020527442867506896, 0.0019990893815551966, -0.0019475108528236602, 0.0018979029346303591, -0.0018501665102868717,
0.0018042086169710208, -0.0017599419928727915, 0.001717284662783333, -0.0016761595583798632, 0.0016364941699329588, -
0.0015982202265829165, 0.0015612734023594012, -0.0015255930458533834, 0.001491121931211595, -0.0014578060284364328,
0.0014255942917196 23, -0.0013944384638054285, 0.0013642928951308466, -0.0013351143765637213, 0.00130686198453017, -
0.0012794969374355127, 0.0012529824625421783, -0.0012272836723996436, 0.00120236745000203, -0.0011782023420969963, 0.0]
b =  [5.265554050078161, -5.774369678628878, 2.3097478714515502, -1.1548739357257736, 0.6793376092504566, -
0.4441822829714538, 0.312128090736695, -0.23097478714515646, 0.1776729131885801, -0.14083828484460464,
0.11434395403225395, -0.09466179801031344, 0.07964647832591897, -0.06793376092504746, 0.05862304242262804, -
0.05110061662503418, 0.044936729016566215, -0.03982323916295617, 0.035534582637716, -0.031902594909555125,
0.02879984877122056, -0.02612836958655715, 0.023811833726305378, -0.021790074258981537, 0.020015146199752917, -
0.018448465426925263, 0.017058699198310517, -0.01582019090034844, 0.01471176988185205, -0.01371584246704926,
0.01281769074057233, -0.01200492656679935, 0.011267062787568307, -0.010595173722253992, 0.009981624336435412, -
0.00941985265681999, 0.008904193798969366, -0.008429736757121505, 0.007992207167661402, -0.007587870799778067,
0.007213453689738906, -0.006866075717754536, 0.00654319510326376, -0.006242561814732761, 0.005962178294917012, -
0.005700266217791001, 0.005455238241496265, -0.0052256739173065034, 0.005010299070390174, -0.004807968092112336,
0.004617648683418303, -0.004438408669202221, 0.0042694045683033975, -0.004109871657386625, 0.003959115309309038, -
0.0038165034227602354, 0.0036814597887357865, -0.0035534582637771605, 0.003432017639600842, -0.003316697115810202,
0.003207092295817246, -0.0031028316381630523, 0.003003573304874352, -0.002909002356996015, 0.0028188282541496065, -
0.0027327826212125046, 0.0026506172497755016, -0.0025721023067357155, 0.002497024725885095, -0.0024251867612929204,
0.0023564046841970023, -0.002290507607541411, 0.002227336423772561, -0.00216674284376745, 0.0021085885260662706, -
0.0020527442867506896, 0.0019990893815551966, -0.0019475108528236602, 0.0018979029346303591, -0.0018501665102868717,
0.0018042086169710208, -0.0017599419928727915, 0.001717284662783333, -0.0016761595583798632, 0.0016364941699329588, -
0.0015982202265829165, 0.0015612734023594012, -0.0015255930458533834, 0.001491121931211595, -0.0014578060284364328,
0.0014255942917196 23, -0.0013944384638054285, 0.0013642928951308466, -0.0013351143765637213, 0.00130686198453017, -
0.0012794969374355127, 0.0012529824625421783, -0.0012272836723996436, 0.00120236745000203, -0.0011782023420969963, 0.0]
```



We can use function approximation as a curvefitting method by combining with other curvefitting-interpolation method. In the next example interpolation function is created by using cubic polynomial then Forier function approximation is applied

```
from math import *
from f_x import *
import gauss
import numpy as np;
import matplotlib.pyplot as plt;
import os;

#======= Cubic spline ===================
def thomas(f,e,g,r):
    n=len(f);
    x=[0.0 for i in range(n)];
    for k in range(0,n): e[k]=e[k]/f[k-1];f[k]=f[k]-e[k]*g[k-1];
    for k in range(1,n): r[k]=r[k]-e[k]*r[k-1];
    x[n-1]=r[n-1]/f[n-1];
    for k in range(n-2,-1,-1):  x[k]=(r[k]-g[k]*x[k+1])/f[k];
    return x;

def cubic_spline(xi,yi,c0,cn):
    n=len(xi);
    h=[0.0 for i in range(n)];
    w=[0.0 for i in range(n)];
    f=[0.0 for i in range(n)];
    e=[0.0 for i in range(n)];
    g=[0.0 for i in range(n)];
    d=[0.0 for i in range(n)];
    x=[0.0 for i in range(n)];
    S= [[0 for x in range(n)] for y in range(4)];
    for k in range(0,n-1):
        h[k]=xi[k+1]-xi[k];
        w[k]=(yi[k+1]-yi[k])/h[k];
    d[0]=c0;
    d[n-1]=cn;
    for k in range(0,n-1):
       d[k]=6.0*(w[k]-w[k-1]);
    f[0]=1.0;
    f[n-1]=1.0;
    g[0]=0.0;
    g[n-1]=0.0;
    e[0]=0.0;
    e[n-1]=0.0;
```

```
    for k in range(0,n-1):
        f[k]=2.0*(h[k]+h[k-1]);e[k]=h[k-1];g[k]=h[k];
    S[2]=thomas(f,e,g,d);
    S[3]=xi;
    for k in range(0,n-1):
       S[0][k]=(6.*yi[k+1]-h[k]*h[k]*S[2][k+1])/(6.0*h[k]);
       S[1][k]=(6.*yi[k]-h[k]*h[k]*S[2][k])/(6.0*h[k]);
    return S;


def funcSpline(S,x):
    n=len(S[0]);
    xx1=0;
    xx2=0;
    y=0;
    hk=0;
    for k in range(0,n-1):
       if S[3][k]<=x and x<=S[3][k+1]:
           hk=(S[3][k+1]-S[3][k]);
           xx1=(x-S[3][k]);
           xx2=(S[3][k+1]-x);
           y=S[0][k]*xx1+S[1][k]*xx2+(xx1*xx1*xx1*S[2][k+1]+xx2*xx2*xx2*S[2][k])/(6.0*hk);
           break;
    if y==0 and S[3][n-2]<=x:
         k=n-2;
         hk=(S[3][k+1]-S[3][k]);
         xx1=(x-S[3][k]);
         xx2=(S[3][k+1]-x);
         y=S[0][k]*xx1+S[1][k]*xx2+(xx1*xx1*xx1*S[2][k+1]+xx2*xx2*xx2*S[2][k])/(6.0*hk);
    return y;

def listSpline(xi,yi,numberofmidpoints,c0,cn):
    # numberofmidpoints :  in x--o--o--x--o--o--x chain if x's are esxperimental points
                # numberofmidpoints is 2
    n=len(xi);
    nn=(n-1)*(numberofmidpoints+1)+1;
    z=[[0 for x in range(nn)] for y in range(2)];
    S= [[0 for x in range(nn)] for y in range(4)];
    S=cubic_spline(xi,yi,c0,cn);
    dx=0;
    k=0;
    for i in range(0,n-1):
       z[0][k]=xi[i];z[1][k]=funcSpline(S,z[0][k]);k=k+1;
       for j in range(numberofmidpoints):
          dx=(xi[i+1]-xi[i])/(numberofmidpoints+1.0);
          z[0][k]=z[0][k-1]+dx;z[1][k]=funcSpline(S,z[0][k]);k=k+1;
    z[0][k]=xi[i+1];z[1][k]=funcSpline(S,z[0][k]);
    return z;

def read_array2D(d):
    path = d
    file1 = open(path,'r')
    file1.seek(0,2)  #jumps to the end
    endLocation=file1.tell()
    file1.seek(0) #jumps to the start
    nn=0
    m=0
    while True:
       line=file1.readline()
       ee=line.split()
       nn+=1
       if  file1.tell()==endLocation:
          break
    m=len(ee)
    print("m=",m)
    a=[[0.0 for i in range(nn)] for j in range(m)]
    file1.seek(0) #jumps to the start
    for i in range(nn):
       line=file1.readline()
       ee=line.split()
       for j in range(m):
          a[j][i]=float(ee[j])
    file1.close()
    return a
# Fourier
def fak(f,k,x):
```

```
    a=0
    mult=1.0/pi
    a=mult*f.func(x)*cos(k*x)
    return a

def fbk(f,k,x):
    a=0
    mult=1.0/pi
    a=mult*f.func(x)*sin(k*x)
    return a

#def gauss_kronrod_integral(f,x1,x2,k,n,eqn_type):
def integral(ff,a,b,k,n,eqn_type):
    # eps hata miktarı
    eps=1.0e-10
    Aused=0
    integral=0
    toplamhata = 0
    nn = 0
    ng = 0
    i = 0
    j = 0
    h = 0
    v = 0
    k1 = 0
    k2 = 0
    intg = 0
    intk = 0
    ta = 0
    tb = 0
    Aw = 4
    A = [[0.0 for i in range(Aw-1+1)] for j in range(n-1+1)]    #[n-1+1][Aw-1+1];
    nn = 61;
    ng = 15;
    x =[0.0 for i in range(nn)]
    ck =[0.0 for i in range(nn)]
    cg =[0.0 for i in range(nn)]
    #Gauss coefficients
    cg[0] = 0.00796819249616660561546588347467 4
    cg[1] = 0.01846646831109095914230213191204 7
    cg[2] = 0.02878470788332336934971917961129 2
    cg[3] = 0.03879919256962704959680193644634 8
    cg[4] = 0.04840267283059405290293814042280 8
    cg[5] = 0.05749315621761906648172168940205 6
    cg[6] = 0.06597422988218049512812851511596 2
    cg[7] = 0.07375597473770520626824385002219 1
    cg[8] = 0.08075589522942021535469493846053 0
    cg[9] = 0.08689978720108297980238753071512 6
    cg[10] = 0.09212252223778612871763270708761 9
    cg[11] = 0.09636873717464425963946862635181 0
    cg[12] = 0.09959342058679526706278028210356 9
    cg[13] = 0.10176238974840550459642895216855 4
    cg[14] = 0.10285265289355884034128563670541 5
    x[0] = 0.99948441005049063757132589570581 1
    x[1] = 0.99689348407464954027163005091869 5
    x[2] = 0.99163099687040459485862836610948 6
    x[3] = 0.98366812327974720997003258160566 3
    x[4] = 0.97311632250112626837469386842370 7
    x[5] = 0.96002186496830751221687102558179 8
    x[6] = 0.94437444474855997941583132403743 9
    x[7] = 0.92620004742927432587932427708047 4
    x[8] = 0.90557330769990779854652255892595 8
    x[9] = 0.88256053579205268154311646253022 6
    x[10] = 0.85720523354606109895865851065894 4
    x[11] = 0.82956576238276839744289811973250 2
    x[12] = 0.79972783582183908301366894232268 3
    x[13] = 0.76777743210482619491797734097450 3
    x[14] = 0.73379006245322680472617113136952 8
    x[15] = 0.69785049479331579693229238802664 0
    x[16] = 0.66006106412662696137005366814927 1
    x[17] = 0.62052618298924286114047755643118 9
    x[18] = 0.57934523582636169175602493217254 0
    x[19] = 0.53662414814201989926416979331107 3
    x[20] = 0.49248046786177857499369306120770 9
    x[21] = 0.44703376953808917678060990032285 4
    x[22] = 0.40040125483039439253547621154266 1
```

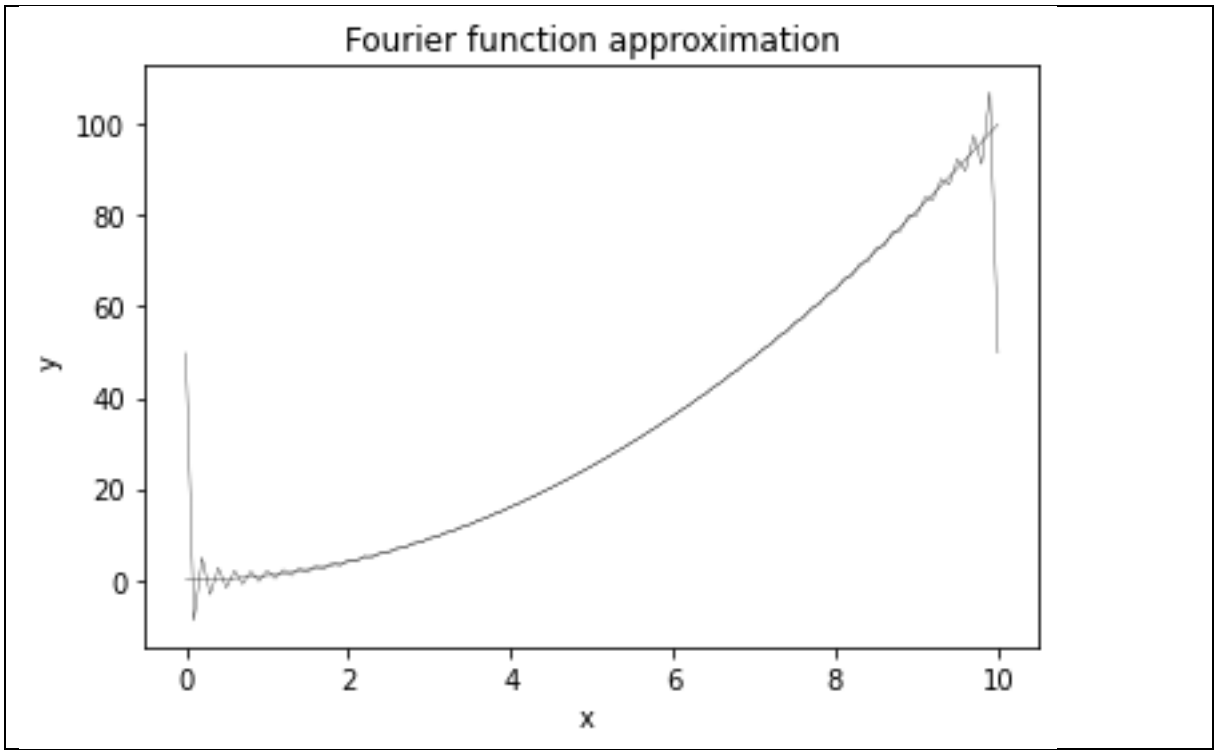
```
x[23] = 0.35270472553087811347103720708937 4
x[24] = 0.30407320227362507737267710719925 7
x[25] = 0.25463692616788984643980512981780 5
x[26] = 0.20452511668230989143895767100202 5
x[27] = 0.15386991360858354696379467274325 6
x[28] = 0.10280693796673703014709675131800 1
x[29] = 0.05147184255531769583302521316672 3
x[30] = 0.00000000000000000000000000000000
#Kronrod coefficients
ck[0] = 0.00138901369867700762455159122676 0
ck[1] = 0.00389046112709988405126720184451 6
ck[2] = 0.00663070391593129217331982636975 0
ck[3] = 0.00927327965951776342844114689202 4
ck[4] = 0.01182301525349634174223289885325 1
ck[5] = 0.01436972950704580481245143244358 0
ck[6] = 0.01692088918905327262757228942032 2
ck[7] = 0.01941414119394238117340895105012 8
ck[8] = 0.02182803582160919229716748573833 9
ck[9] = 0.02419116207808060136568637072523 2
ck[10] = 0.02650995488233310161060170933507 5
ck[11] = 0.02875404876504129284397878535433 4
ck[12] = 0.03090725756238776247288425294309 2
ck[13] = 0.03298144705748372603181419101685 4
ck[14] = 0.03497933802806002413749967073146 8
ck[15] = 0.03688236465182122922391106561713 6
ck[16] = 0.03867894562472759295034865153228 1
ck[17] = 0.04037453895153595911199527975246 8
ck[18] = 0.04196981021516424614714754128597 0
ck[19] = 0.04345253970135606931683172811707 3
ck[20] = 0.04481480013316266319235555161672 3
ck[21] = 0.04605923827100698811627173555937 4
ck[22] = 0.04718554656929915394526147818109 9
ck[23] = 0.04818586175708712914077949229830 5
ck[24] = 0.04905543455502977888752816536723 8
ck[25] = 0.04979568342707420635781156937994 2
ck[26] = 0.05040592140278234684089308565358 5
ck[27] = 0.05088179589874960649229747304980 5
ck[28] = 0.05122154784925877217065628260494 4
ck[29] = 0.05142612853745902593386287921578 1
ck[30] = 0.05149472942945156755834043364709 9
for i in range(nn-1,int(nn/2-1),-1):
    x[i] = -x[nn-1-i]
for i in range(nn-1,int(nn/2-1),-1):
    ck[i] = ck[nn-1-i]
for i in range(ng-1,-1,-1):
    cg[nn-2-2*i] = cg[i]
    cg[1+2*i] = cg[i]
for i in range(0,int(nn/2+1)):
    cg[2*i] = 0
k1 = 0.5*(b-a);
k2 = 0.5*(b+a);
intg = 0;
intk = 0;
for i in range(0,nn):
    xx=k1*x[i]+k2
    if eqn_type==0:
        v=fak(ff,k,xx)
    else:
        v=fbk(ff,k,xx)
    intk = intk+v*ck[i];
    if  i%2==1:
        intg = intg+v*cg[i]
intk = intk*(b-a)*0.5
intg = intg*(b-a)*0.5
A[0][0] = abs(intg-intk);
A[0][1] = intk;
A[0][2] = a;
A[0][3] = b;
toplamhata = A[0][0]
if toplamhata<eps:
    sonuc = True
    integral = intk
    Aused = 1
    return integral
Aused = 1
for h in range(1,n):
```

```
        Aused = h+1
        gir(A, h, Aw)
        toplamhata = toplamhata-A[h-1][0]
        ta = A[h-1][2]
        tb = A[h-1][3]
        A[h-1][2] = ta
        A[h-1][3] = 0.5*(ta+tb)
        A[h][2] = 0.5*(ta+tb)
        A[h][3] = tb
        for j in range(h-1,h+1):
            k1 = 0.5*(A[j][3]-A[j][2])
            k2 = 0.5*(A[j][3]+A[j][2])
            intg = 0.0
            intk = 0.0
            for i in range(0,nn):
                xx=k1*x[i]+k2
                if eqn_type==0:
                    v=fak(ff,k,xx)
                else:
                    v=fbk(ff,k,xx)
                intk = intk+v*ck[i];
                if  i%2==1:
                    intg = intg+v*cg[i]
            intk = intk*(A[j][3]-A[j][2])*0.5
            intg = intg*(A[j][3]-A[j][2])*0.5
            A[j][0] = abs(intg-intk)
            A[j][1] = intk
            toplamhata = toplamhata+A[j][0]

        cik(A, h-1, Aw);
        cik(A, h, Aw);
        if toplamhata<eps:
            break
    sonuc = toplamhata<eps
    integral = 0
    for j in range(0,Aused):
        integral = integral+A[j][1]
    return integral

def gir(A,n,Awidth):
    i = 0
    p = 0
    t = 0
    maxcp = 0
    if n==1:
        return
    for i in range(0,Awidth):
        t = A[n-1][i]
        A[n-1][i] = A[0][i]
        A[0][i] = t
    p = 0
    while 2*p+1<n-1:
        maxcp = 2*p+1;
        if 2*p+2<n-1:
            if A[2*p+2][0]>A[2*p+1][0]:
                maxcp = 2*p+2
        if  A[p][0]<A[maxcp][0]:
            for i in range(0,Awidth):
                t = A[p][i]
                A[p][i] = A[maxcp][i]
                A[maxcp][i] = t
            p = maxcp
        else:
            break



def cik(A,n,Awidth):
    i = 0
    p = 0
    t = 0
    kk = 0
    if n==0:
        return
    p = n
    while p!=0:
```

```
        kk =int( (p-1)/2)
        if A[p][0]>A[kk][0]:
            for i in range(0,Awidth):
                t = A[p][i];
                A[p][i] = A[kk][i]
                A[kk][i] = t
            p = kk
        else:
            break

def Trigonometric(f,n):
    a=[0.0 for i in range(n+1)]
    b=[0.0 for i in range(n+1)]
    for k in range(n):
        a[k]=integral(f,-pi,pi,k,80,0)
    b[0]=0.0
    for k in range(1,n):
        b[k]=integral(f,-pi,pi,k,80,1)
    aa=[a,b]
    return aa


def func(ak,bk,x):
    n=len(a)-1
    cc=ak[0]/2.0+ak[n]*cos(n*x)+bk[n]*sin(n*x)
    for k in range(1,n):
        cc=cc+ak[k]*cos(k*x)+bk[k]*sin(k*x)
    return cc


def plot_func(f,ak,bk,xmin,xmax):
    nn=101
    xx=[0.0 for j in range(nn)]
    zz=[0.0 for j in range(nn)]
    yy=[0.0 for j in range(nn)]
    yy1=[0.0 for j in range(nn)]
    for i in range(nn):
        xx[i]=xmin+i*(xmax-xmin)/(nn-1)
        zz[i]=z_x(xmin,xmax,xx[i])
        yy[i]=func(ak,bk,zz[i])
        yy1[i]=f.func(zz[i])
    plt.title('Fourier function approximation')
    plt.xlabel('x ')
    plt.ylabel('y ');
    plt.plot(xx,yy,'k',xx,yy1,'k',linewidth=0.3)

def plot_func_error(f,ak,bk,xmin,xmax):
    nn=101
    xx=[0.0 for j in range(nn)]
    yy=[0.0 for j in range(nn)]
    zz=[0.0 for j in range(nn)]
    yy1=[0.0 for j in range(nn)]
    yy2=[0.0 for j in range(nn)]
    for i in range(nn):
        xx[i]=xmin+i*(xmax-xmin)/(nn-1)
        zz[i]=z_x(xmin,xmax,xx[i])
        yy[i]=func(ak,bk,zz[i])
        yy1[i]=f.func(zz[i])
        yy2[i]=yy[i]-yy1[i]
    plt.title('Fourier function rational approximation error')
    plt.xlabel('x ')
    plt.ylabel('y ');
    plt.plot(xx,yy2,'k',linewidth=0.3)

#conversion function z=f(x)    x0<x<xn   -pi<z<pi
def  z_x(x0,xn,x):
    zz=2.0*pi*(x-x0)/(xn-x0)-pi;
    return zz

#conversion function x=f(z)    x0<x<xn   -pi<z<pi
def x_z(x0,xn,z):
    xx=(z+pi)/(2.0*pi)*(xn-x0)+x0
    return xx

def set_z_x(x0,xn,x):
    n=len(x)
```

```
    z=[0.0 for j in range(n)]
    for j in range(n):
        z[j]=z_x(x0,xn,x[j])
    return z

def set_x_z(x0,xn,z):
    n=len(z)
    x=[0.0 for j in range(n)]
    for j in range(n):
        x[j]=x_z(x0,xn,z[j])
    return x

#main method
a=read_array2D('c.txt')
x=a[0];
print("x=",x)
xmin=min(x)
xmax=max(x)
z=set_z_x(xmin,xmax,x)
print("z=",z)
y=a[1];
print("y=",y)
c0=0.0
cn=0.0
S=cubic_spline(z,y,c0,cn)
class f1(f_x):func=lambda self,z: funcSpline(S,z)
f = f1()
n=50
c= Trigonometric(f,n)
print("Fourier Function approximation n = ",n)
a=c[0]
b=c[1]
print("a = ",a)
print("b = ",a)

plot_func(f,a,b,xmin,xmax)
#plot_func_error(f,a,b,1.0,3.0)
```

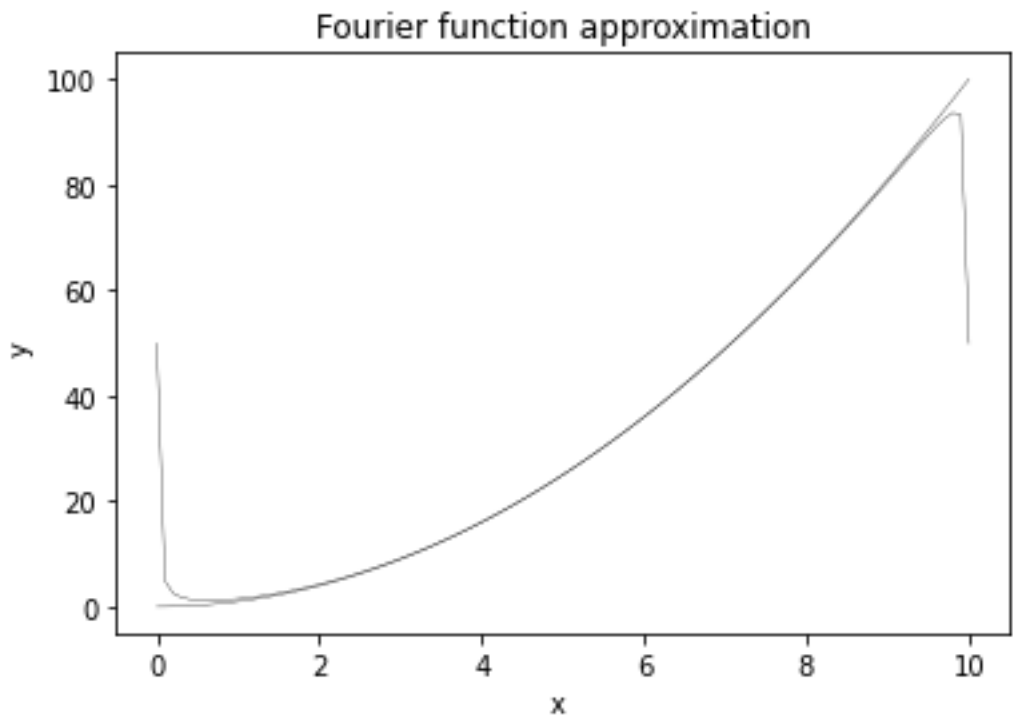
```
runfile('E:/okul/SCO1/Fourier2.py', wdir='E:/okul/SCO1')
m= 2
x= [0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0]
z= [-3.141592653589793, -2.5132741228718345, -1.8849555921538759, -1.2566370614359172, -0.6283185307179586, 0.0,
0.6283185307179586, 1.2566370614359172, 1.8849555921538759, 2.5132741228718345, 3.141592653589793]
y= [0.0, 1.0, 4.0, 9.0, 16.0, 25.0, 36.0, 49.0, 64.0, 81.0, 100.0]
Fourier Function approximation n =  50
a =  [66.67628917114205, -10.141862036816232, 2.5430495037027727, -1.1357758284083892, 0.6413521831844353, -
0.40781013430732865, 0.2785329821900868, -0.20224198840003016, 0.15406898196571728, -0.12152297027587206,
0.09839629299869285, -0.08133989332795988, 0.06839491195230347, -0.05834314381368921, 0.050377952739139364, -
0.04390730258465772, 0.03853908067862051, -0.034088361570065784, 0.030381898419286735, -0.02725891000668207,
0.02459907324106224, -0.0223136386343723, 0.020335491332364564, -0.018612730371802833, 0.01710304111397575, -
0.015766166805160214, 0.014570440234077497, -0.013503909073705411, 0.01255271183782114, -0.011700343257594836,
0.010932921433662174, -0.010239275543240105, 0.009610281642594014, -0.009038373967995172, 0.00851684274549322, -
0.008038275032385259, 0.007596326576736124, -0.007189293610971537, 0.006814785700305026, -0.00646931493694345,
0.0061497683020505884, -0.0058535449674257944, 0.005578448658041135, -0.005322601626189916, 0.005084254629509212, -
0.004861244290298883, 0.004651616383534257, -0.004455041698204543, 0.004270923041363807, -0.004098190419876399, 0.0]
b =  [66.67628917114205, -10.141862036816232, 2.5430495037027727, -1.1357758284083892, 0.6413521831844353, -
0.40781013430732865, 0.2785329821900868, -0.20224198840003016, 0.15406898196571728, -0.12152297027587206,
0.09839629299869285, -0.08133989332795988, 0.06839491195230347, -0.05834314381368921, 0.050377952739139364, -
0.04390730258465772, 0.03853908067862051, -0.034088361570065784, 0.030381898419286735, -0.02725891000668207,
0.02459907324106224, -0.0223136386343723, 0.020335491332364564, -0.018612730371802833, 0.01710304111397575, -
0.015766166805160214, 0.014570440234077497, -0.013503909073705411, 0.01255271183782114, -0.011700343257594836,
0.010932921433662174, -0.010239275543240105, 0.009610281642594014, -0.009038373967995172, 0.00851684274549322, -
0.008038275032385259, 0.007596326576736124, -0.007189293610971537, 0.006814785700305026, -0.00646931493694345,
0.0061497683020505884, -0.0058535449674257944, 0.005578448658041135, -0.005322601626189916, 0.005084254629509212, -
0.004861244290298883, 0.004651616383534257, -0.004455041698204543, 0.004270923041363807, -0.004098190419876399, 0.0]
```




```
runfile('E:/okul/SCO1/Fourier2.py', wdir='E:/okul/SCO1')
m= 2
x= [0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0]
z= [-3.141592653589793, -2.5132741228718345, -1.8849555921538759, -1.2566370614359172, -0.6283185307179586, 0.0,
0.6283185307179586, 1.2566370614359172, 1.8849555921538759, 2.5132741228718345, 3.141592653589793]
y= [0.0, 1.0, 4.0, 9.0, 16.0, 25.0, 36.0, 49.0, 64.0, 81.0, 100.0]
Fourier Function approximation n =  50
a =  [66.67628917114205, -10.141862036816232, 2.5430495037027727, -1.1357758284083892, 0.6413521831844353, -
0.40781013430732865, 0.2785329821900868, -0.20224198840003016, 0.15406898196571728, -0.12152297027587206,
0.09839629299869285, -0.08133989332795988, 0.06839491195230347, -0.05834314381368921, 0.050377952739139364, -
0.04390730258465772, 0.03853908067862051, -0.034088361570065784, 0.030381898419286735, -0.02725891000668207,
0.02459907324106224, -0.0223136386343723, 0.020335491332364564, -0.018612730371802833, 0.01710304111397575, -
0.015766166805160214, 0.014570440234077497, -0.013503909073705411, 0.01255271183782114, -0.011700343257594836,
0.010932921433662174, -0.010239275543240105, 0.009610281642594014, -0.009038373967995172, 0.00851684274549322, -
0.008038275032385259, 0.007596326576736124, -0.007189293610971537, 0.006814785700305026, -0.00646931493694345,
0.0061497683020505884, -0.0058535449674257944, 0.005578448658041135, -0.005322601626189916, 0.005084254629509212, -
0.004861244290298883, 0.004651616383534257, -0.004455041698204543, 0.004270923041363807, -0.004098190419876399, 0.0]
b =  [66.67628917114205, -10.141862036816232, 2.5430495037027727, -1.1357758284083892, 0.6413521831844353, -
0.40781013430732865, 0.2785329821900868, -0.20224198840003016, 0.15406898196571728, -0.12152297027587206,
0.09839629299869285, -0.08133989332795988, 0.06839491195230347, -0.05834314381368921, 0.050377952739139364, -
0.04390730258465772, 0.03853908067862051, -0.034088361570065784, 0.030381898419286735, -0.02725891000668207,
0.02459907324106224, -0.0223136386343723, 0.020335491332364564, -0.018612730371802833, 0.01710304111397575, -
0.015766166805160214, 0.014570440234077497, -0.013503909073705411, 0.01255271183782114, -0.011700343257594836,
0.010932921433662174, -0.010239275543240105, 0.009610281642594014, -0.009038373967995172, 0.00851684274549322, -
0.008038275032385259, 0.007596326576736124, -0.007189293610971537, 0.006814785700305026, -0.00646931493694345,
0.0061497683020505884, -0.0058535449674257944, 0.005578448658041135, -0.005322601626189916, 0.005084254629509212, -
0.004861244290298883, 0.004651616383534257, -0.004455041698204543, 0.004270923041363807, -0.004098190419876399, 0.0]

runfile('E:/okul/SCO1/Fourier2.py', wdir='E:/okul/SCO1')
Reloaded modules: f_x, gauss
m= 2
x= [0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0]
z= [-3.141592653589793, -2.5132741228718345, -1.8849555921538759, -1.2566370614359172, -0.6283185307179586, 0.0,
0.6283185307179586, 1.2566370614359172, 1.8849555921538759, 2.5132741228718345, 3.141592653589793]
y= [0.0, 1.0, 4.0, 9.0, 16.0, 25.0, 36.0, 49.0, 64.0, 81.0, 100.0]
Fourier Function approximation n =  100
a =  [66.67628917114205, -10.141862036816232, 2.5430495037027727, -1.1357758284083892, 0.6413521831844353, -
0.40781013430732865, 0.2785329821900868, -0.20224198840003016, 0.15406898196571728, -0.12152297027587206,
0.09839629299869285, -0.08133989332795988, 0.06839491195230347, -0.05834314381368921, 0.050377952739139364, -
0.04390730258465772, 0.03853908067862051, -0.034088361570065784, 0.030381898419286735, -0.02725891000668207,
0.02459907324106224, -0.0223136386343723, 0.020335491332364564, -0.018612730371802833, 0.01710304111397575, -
0.015766166805160214, 0.014570440234077497, -0.013503909073705411, 0.01255271183782114, -0.011700343257594836,
0.010932921433662174, -0.010239275543240105, 0.009610281642594014, -0.009038373967995172, 0.00851684274549322, -
0.008038275032385259, 0.007596326576736124, -0.007189293610971537, 0.006814785700305026, -0.00646931493694345,
```

0.0061497683020505884, -0.005853544967425794 4, 0.005578448658041135, -0.005322601626189916, 0.005084254629509212, -0.004861244290298883, 0.004651616383534257, -0.004455041698204543, 0.004270923041363807, -0.004098190419876399, 0.003935851708770477, -0.0037830624564985788, 0.0036390981056385607, -0.0035033301431848483, 0.0033751525478602617, -0.0032537415926545 41, 0.003138323463721798, -0.003028836088048684, 0.002925096539990457, -0.0028266923663036753, 0.0027332303583325612, -0.0026443726221774717, 0.002559827097172415, -0.002479338961655725, 0.0024026557020891655, -0.0023294043904382 93, 0.002259223913248843, -0.0021921131386875223, 0.0021280097343106263, -0.0020667281633577, 0.0020080876092551936, -0.0019519320093764258, 0.0018981261393628188, -0.0018465518950147 37, 0.0017970895412919 86, -0.0017495484092142667, 0.0017037393162101 95, -0.0016596744306123 72, 0.0016173320093470 31, -0.0015766187683837855, 0.0015374420753971713, -0.0014997218976556233, 0.0014463388925316 84, -0.0014283827585445524, 0.0013946409036447588, -0.0013620567233085523, 0.0013305222056923943, -0.0013000507259025018, 0.0012706354692569432, -0.0012422225407647013, 0.0012147690471574595, -0.0011882219303518 96, 0.0011625456604001 66, -0.0011377069297338976, 0.0011136706024246108, -0.0010903727790193968, 0.0010677476066320805, -0.0010458066760143173, 0.0010245489238348 63, -0.0010039445090656851, 0.0]

b = [66.67628917114205, -10.141862036816232, 2.5430495037027727, -1.1357758284083892, 0.6413521831844353, -0.40781013430732865, 0.2785329821900868, -0.20224198840003016, 0.15406898196571728, -0.12152297027587206, 0.09839629299869285, -0.08133989332795988, 0.06839491195230347, -0.05834314381368921, 0.050377952739139364, -0.04390730258465772, 0.03853908067862051, -0.034088361570065784, 0.030381898419286735, -0.02725891000668207, 0.02459907324106224, -0.0223136386343723, 0.020335491332364564, -0.018612730371802833, 0.01710304111397575, -0.015766166805160214, 0.014570440234077497, -0.013503909073705411, 0.01255271183782114, -0.011700343257594836, 0.010932921433662174, -0.010239275543240105, 0.009610281642594014, -0.009038373967995172, 0.00851684274549322, -0.008038275032385259, 0.007596326576736124, -0.007189293610971537, 0.006814785700305026, -0.00646931493694345, 0.0061497683020505884, -0.005853544967425794 4, 0.005578448658041135, -0.005322601626189916, 0.005084254629509212, -0.004861244290298883, 0.004651616383534257, -0.004455041698204543, 0.004270923041363807, -0.004098190419876399, 0.003935851708770477, -0.0037830624564985788, 0.0036390981056385607, -0.0035033301431848483, 0.0033751525478602617, -0.0032537415926545 41, 0.003138323463721798, -0.003028836088048684, 0.002925096539990457, -0.0028266923663036753, 0.0027332303583325612, -0.0026443726221774717, 0.002559827097172415, -0.002479338961655725, 0.0024026557020891655, -0.0023294043904382 93, 0.002259223913248843, -0.0021921131386875223, 0.0021280097343106263, -0.0020667281633577, 0.0020080876092551936, -0.0019519320093764258, 0.0018981261393628188, -0.0018465518950147 37, 0.0017970895412919 86, -0.0017495484092142667, 0.0017037393162101 95, -0.0016596744306123 72, 0.0016173320093470 31, -0.0015766187683837855, 0.0015374420753971713, -0.0014997218976556233, 0.0014463388925316 84, -0.0014283827585445524, 0.0013946409036447588, -0.0013620567233085523, 0.0013305222056923943, -0.0013000507259025018, 0.0012706354692569432, -0.0012422225407647013, 0.0012147690471574595, -0.0011882219303518 96, 0.0011625456604001 66, -0.0011377069297338976, 0.0011136706024246108, -0.0010903727790193968, 0.0010677476066320805, -0.0010458066760143173, 0.0010245489238348 63, -0.0010039445090656851, 0.0]



There is a discreate analog that is useful for the discreate least square approximation and interpolation of the data. Assume that a collection of 2m paired data points (x_j,y_j) j=0..2m-1 is given. We assume that the interval is $[-\pi,\pi]$. If any other range is given, it should be converted by using linear conversion. Standart linear least square curve fitting method for w=1 can be applied here. For the trigonometric function

$$S_n(x) = \frac{a_0}{2} + a_n \cos(nx) + \sum_{k=1}^{n-1} [a_k \cos(kx) + b_k \sin(kx)] \quad (6.15-7)$$

Error term will be:

$$E(S_n) = \sum_{j=0}^{2m-1} \left\{ y_j - \left[\frac{a_0}{2} + a_n \cos(nx_j) + \sum_{k=1}^{n-1} [a_k \cos(kx_j) + b_k \sin(kx_j)] \right] \right\} \quad (6.15-8)$$

We need to choose a_k and b_k coefficients so that error will be minimum. It should be note that cosine and sine functions are orthogonal and for orthogonal functions relation

$$\sum_{k=1}^{2m-1} \emptyset_k(x_j)\emptyset_l(x_j) = 0 \quad for\ k \neq l \quad (6.15-9)$$

Will held, So solution set will not require a matrix solution and the coeeficients will be:

$$a_k = \frac{1}{m} \sum_{j=0}^{2m-1} y_j \cos(kx_j) \quad (6.15-10)$$

$$b_k = \frac{1}{m} \sum_{j=0}^{2m-1} y_j \sin(kx_j) \quad (6.15-11)$$

If the function is defined in some other interval than [-π,π] then a range correction should apply
For example for the range $[x_0, x_n]$ ranges can be converted by using transfer functions

$z = \frac{2\pi(x-x_0)}{(x_n-x_0)} - \pi$ (6.15-12)

$x = \frac{(x_n-x_0)(x+\pi)}{2\pi} + x_0$ (6.15-13)

So basic equations becomes:

$S_n(z) = \frac{a_0}{2} + a_n \cos(nz) + \sum_{k=1}^{n-1}[a_k \cos(kz) + b_k \sin(kz)]$ (6.15-14)

$a_k = \frac{1}{m}\sum_{j=0}^{2m-1} y_j \cos(kz_j)$ (6.15-16)

$b_k = \frac{1}{m}\sum_{j=0}^{2m-1} y_j \sin(kz_j)$ (6.15-17)

Note that if a discreate set is given problem can be solved either as a function fitting or data fitting problem. As an example problem Let us find discreate least square approximation $S_3(x)$, for m=5 (2m-1=9). The function is $f(x) = x^4 - 3x^3 + 2x^2 \tan(x(x-1))$. The problem first solved as an excel datasheet problem

FOURIER DISCRETE LEAST SQUARE CURVE

	xi			*k*	*0*	*1*	*2*	*3*				
j	**j/5**	**z_i**		**f(x)**	**a_0**	**a_1**	**a_2**	**a_3**	**b_0**	**b_1**	**b_2**	**b_3**
0	0	-3.14159	-	0	0	0	0	0	0	0	0	0
1	0.2	-2.51327	-	0.43400285	0.0868006	-0.0702231	0.0268228	0.0268229	0	-0.0510201	0.0825523	-0.082552
2	0.4	-1.88496	-	0.89814382	0.1796288	-0.0555083	-0.1453227	0.1453227	0	-0.1708371	0.1055831	0.1055832
3	0.6	-1.25664	-	1.31723235	0.2634465	0.0814094	-0.2131327	-	0	-0.2505525	-0.15485	0.1548499
4	0.8	-0.62832	-	1.58195749	0.3163915	0.2559661	0.0977704	-	0	-0.1859703	-0.3009062	-0.300906
5	1	0	0	1.55740772	0.3114815	0.3114815	0.3114815	0.3114815	0	0	0	0
6	1.2	0.62832	0.62832	1.19795749	0.2395915	0.1938336	0.0740378	-	0	0.1408283	0.2278651	0.2278651
7	1.4	1.25664	1.25664	0.64523235	0.1290465	0.0398776	-0.1044008	-	0	0.1227305	0.0758516	-0.075852
8	1.6	1.88496	1.88496	0.13014382	0.0260288	-0.0080433	-0.0210577	0.0210577	0	0.0247548	-0.0152993	-0.015299
9	1.8	2.51327	2.51327	-0.1419971	-	0.0229756	-0.0087759	-	0	-0.0166928	0.0270095	-0.027009
10	2	3.14159	3.14159	0	0	0	0	0	0	0	0	0
				7.62008075	1.5524156	0.7717691	0.0174228	0.0065673	0	-0.386759	0.047806	-0.013321

x	**z**	**f(x)**	**$S_3(x)$**	**$\lvert f(x)-S_3(x)\rvert$**
0	-3.14159	0	0.01529	0.01529429
0.125	-2.74889	0.2644	0.2548	0.00959471
0.375	-1.9635	0.84081	0.86573	0.02492368
0.625	-1.1781	1.3615	1.37668	0.015174
0.875	-0.3927	1.61282	1.61826	0.00544731
1.125	0.392699	1.36672	1.38986	0.02313645
1.375	1.178097	0.71697	0.72965	0.01267567
1.625	1.963495	0.07909	0.08349	0.00439694

The same calculation is done in python code:

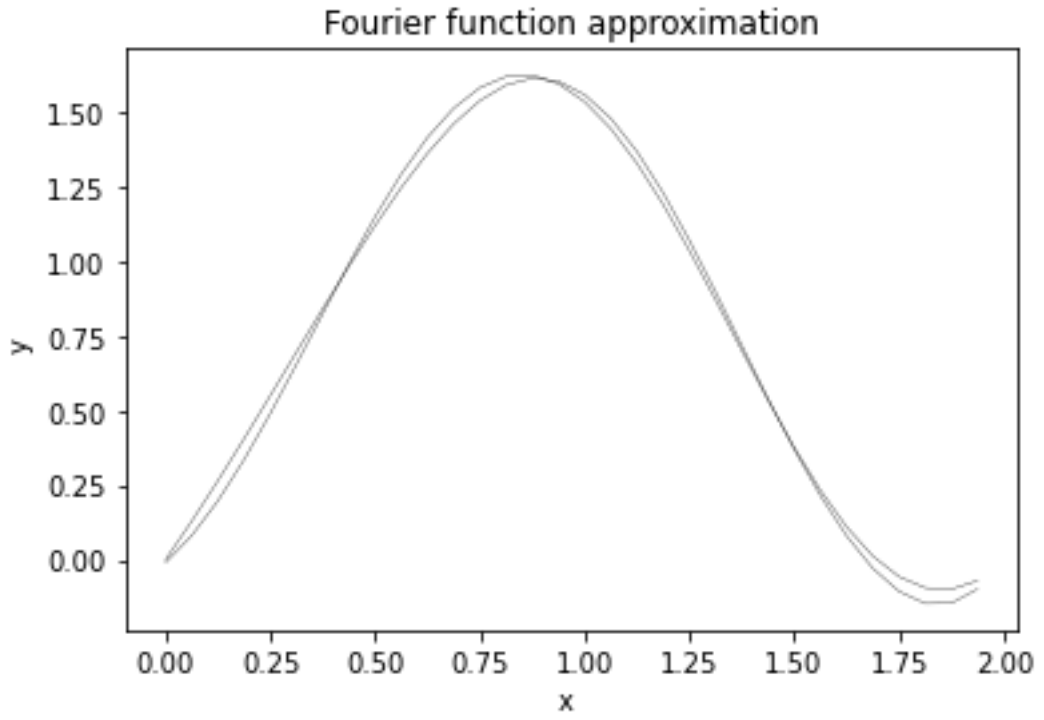
```
from math import *
from f_x import *
import gauss
import numpy as np;
import matplotlib.pyplot as plt;
import os;

#conversion function z=f(x)    x0<x<xn   -pi<z<pi
def  z_x(x0,xn,x):
    zz=2.0*pi*(x-x0)/(xn-x0)-pi;
    return zz

#conversion function x=f(z)    x0<x<xn   -pi<z<pi
def x_z(x0,xn,z):
    xx=(z+pi)/(2.0*pi)*(xn-x0)+x0
    return xx

def set_z_x(x0,xn,x):
    n=len(x)
    z=[0.0 for j in range(n)]
    for j in range(n):
        z[j]=z_x(x0,xn,x[j])
    return z

def set_x_z(x0,xn,z):
    n=len(z)
    x=[0.0 for j in range(n)]
    for j in range(n):
        x[j]=x_z(x0,xn,z[j])
    return x
def Trigonometric(z,y,n):
    a=[0.0 for i in range(n+1)]
    b=[0.0 for i in range(n+1)]
    mm=len(z)
    m=int(mm/2)
    print("mm=",mm,"m=",m)
    for k in range(n+1):
        a[k]=0.0
        b[k]=0.0
        for j in range(2*m):
            a[k]=a[k]+1.0/m*y[j]*cos(k*z[j])
            b[k]=b[k]+1.0/m*y[j]*sin(k*z[j])
    aa=[a,b]
    return aa


def func(ak,bk,xmin,xmax,x):
    n=len(a)-1
    z=z_x(xmin,xmax,x)
    cc=ak[0]/2.0+ak[n]*cos(n*z)+bk[n]*sin(n*z)
    for k in range(1,n):
        cc=cc+ak[k]*cos(k*z)+bk[k]*sin(k*z)
    return cc

def read_array2D(d):
    path = d
    file1 = open(path,'r')
    file1.seek(0,2)  #jumps to the end
    endLocation=file1.tell()
    file1.seek(0) #jumps to the start
    nn=0
    m=0
    while True:
        line=file1.readline()
        ee=line.split()
        nn+=1
        if  file1.tell()==endLocation:
            break
    m=len(ee)
    print("m=",m)
    a=[[0.0 for i in range(nn)] for j in range(m)]
    file1.seek(0) #jumps to the start
    for i in range(nn):
        line=file1.readline()
```

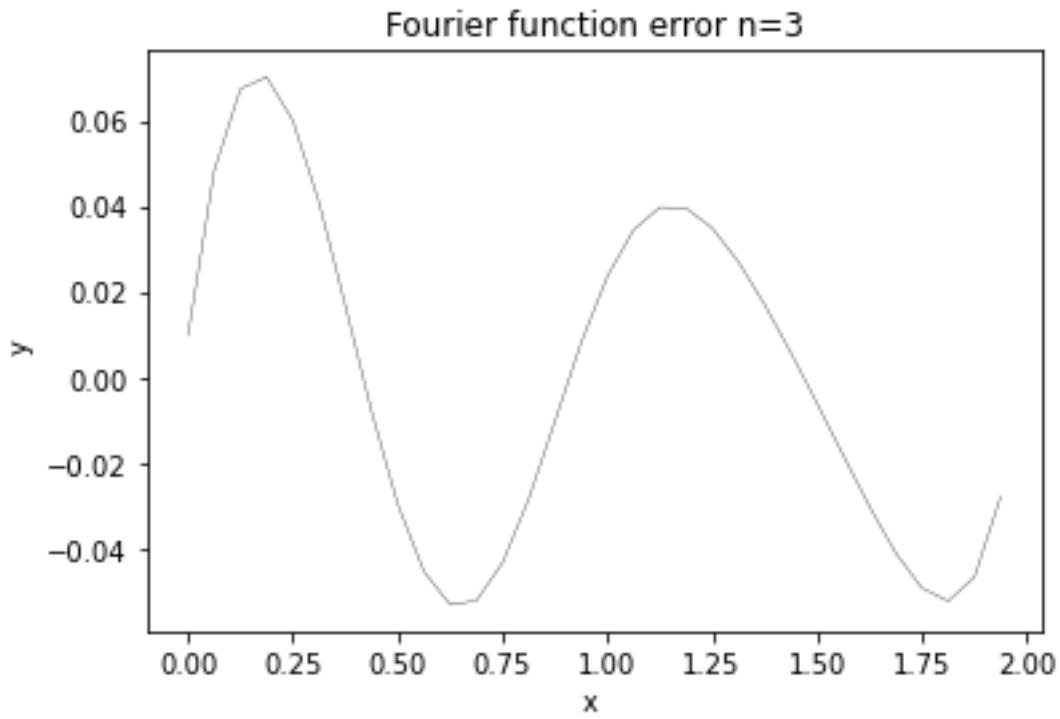
```
        ee=line.split()
        for j in range(m):
            a[j][i]=float(ee[j])
    file1.close()
    return a

def plot_func(f,ak,bk,xmin,xmax):
    nn=32
    xx=[0.0 for j in range(nn)]
    zz=[0.0 for j in range(nn)]
    yy=[0.0 for j in range(nn)]
    yy1=[0.0 for j in range(nn)]
    for i in range(nn):
        xx[i]=xmin+i*(xmax-xmin)/nn
        zz[i]=z_x(xmin,xmax,xx[i])
        yy[i]=func(ak,bk,xmin,xmax,xx[i])
        yy1[i]=f.func(xx[i])
    plt.title('Fourier function approximation')
    plt.xlabel('x ')
    plt.ylabel('y ');
    plt.plot(xx,yy,'k',xx,yy1,'k',linewidth=0.3)

def plot_error(f,ak,bk,xmin,xmax):
    nn=32
    xx=[0.0 for j in range(nn)]
    zz=[0.0 for j in range(nn)]
    yy=[0.0 for j in range(nn)]
    yy1=[0.0 for j in range(nn)]
    yy2=[0.0 for j in range(nn)]
    for i in range(nn):
        xx[i]=xmin+i*(xmax-xmin)/nn
        zz[i]=z_x(xmin,xmax,xx[i])
        yy[i]=func(ak,bk,xmin,xmax,xx[i])
        yy1[i]=f.func(xx[i])
        yy2[i]=yy1[i]-yy[i]
    plt.title('Fourier function error n=3')
    plt.xlabel('x ')
    plt.ylabel('y ');
    plt.plot(xx,yy2,'k',linewidth=0.3)

#main method
a=read_array2D('d.txt')
x=a[0];
mm=len(x)
m=int(mm/2)
xmin=min(x)
xmax=max(x)
print("x=",x)
xmin=min(x)
xmax=max(x)
z=set_z_x(xmin,xmax,x)
print("z=",z)
y=a[1];
print("y=",y)
n=3
b=Trigonometric(z,y,n)
ak=b[0]
bk=b[1]
class f2(f_x):func=lambda self,x:x*x*x*x-3.0*x*x*x+2.0*x*x-tan(x*(x-2.0))
f = f2()
print("ak=",ak,"bk=",bk)
for i in range(2*m+1):
    x=0.125+0.25*i
    y=f.func(x)
plot_error(f,ak,bk,xmin,xmax)
```

```
runfile('E:/okul/SCO1/discreet_Fourier.py', wdir='E:/okul/SCO1')
Reloaded modules: f_x, gauss
m= 2
x= [0.0, 0.2, 0.4, 0.6, 0.8, 1.0, 1.2, 1.4, 1.6, 1.8, 2.0]
z= [-3.141592653589793, -2.5132741228718345, -1.8849555921538759, -1.2566370614359172, -0.6283185307179586, 0.0,
0.6283185307179586, 1.2566370614359172, 1.8849555921538759, 2.5132741228718345, 3.141592653589793]
y= [0.0, 0.434002852, 0.898143822, 1.317232349, 1.581957491, 1.557407725, 1.197957491, 0.645232349, 0.130143822, -0.141997148,
0.0]
mm= 11 m= 5
```

ak= [1.5240161506, 0.7717690393612457, 0.017422790340254395, 0.006567273038754351] bk= [0.0, -0.38675904513315906, 0.0478060470697493 5, -0.013320698468437184]

In [73]:





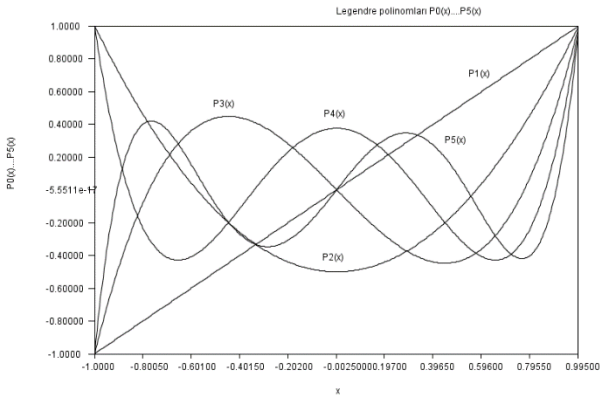
6.16 ORTHOGONAL FUNCTION APPROXIMATION: LEGENDRE POLYNOMIALS

Figure 6.16.1 Adrien Marie Legendre 1752-1833 French Mathematician

Utilising sine and cosine functions as orthogonal function approximation is investigated in the previous sub-section. Similar tasks can be achieved by using other orthogonal polynomials. In this section we will be using Legendre polynomials as a means of function approximation. Definition of LEgendre polynomials are as follows:

$$P_k(x) = \frac{(2k-1)}{k} x P_{k-1}(x) - \frac{(k-1)}{k} P_{k-2}(x) \quad (6.16\text{-}1)$$

In this recursive equation first two terms are defined as: $P_0(x)=1$ and $P_1(x)=x$. The first 6 Legndre functions are shown in the plot below.



(bolum6_049.jpg,Legendre polinomları)

If the first 10 Legendre polynomials are listed in open form:

n	$P_n(x)$ -1<= x <=1
0	1
1	x
2	$(3x^2-1)/2$
3	$(5x^3-3x)/2$
4	$(35x^4-30x^2+3)/8$
5	$(63x^5-70x^3+15x)/8$
6	$(231x^6-315x^4+105x^2-5)/16$
7	$(429x^7-693x^5+315x^3-35x)/16$
8	$(6435x^8-12012x^6+6930x^4-1260x^2+35)/128$
9	$(12155x^9-25740x^7+18018x^5-4620x^3+315x)/128$
10	$(46189x^{10}-109395x^8+90090x^6-30030x^4+3465x^2-63)/256$

Equivalent form of a function f(x) in as Legendre function in the range [-1,1]

$$f(x) = \sum_{k=0}^{m} b_k P_k(x) + error$$

Basic definition of orthogonal form of Legendre function is based on [-1,1] region, if a function definition of a different range is concerned, coordinate system transfer is needed. For the region of [a,b] the following transfer function can be used.

$$z = \left(\frac{b-a}{2}\right)x + \left(\frac{b+a}{2}\right) = \alpha x + \beta \qquad \alpha = \left(\frac{b-a}{2}\right) \quad \beta = \left(\frac{b+a}{2}\right) \quad (6.16\text{-}2)$$

For the new coordinate system equation will be transferred as

$f(z) = \sum_{k=0}^{m} b_k q_k(z) + error$ (6.16-3) In this equation q_k converted Legendre function in the form of

$q_k(z) = q_k(\alpha x + \beta) = P_k(x)$ (6.16-4)

Due to orthogonality properties:

$$\left[P_n(x)P_j(x)\right] = \int_{-1}^{1} P_n(x)P_j(x)dx \quad \text{where} \quad \left[P_n(x)P_j(x)\right] = \begin{Bmatrix} 0 \;\; if \;\; n \neq j \\ \frac{2}{2n+1} \;\; if \; n = j \end{Bmatrix} \quad (6.16\text{-}5)$$

so the coefficients of the function fitting will become $b_k = \frac{2k+1}{b-a}\int_a^b f(z)q_k(z)dz$ (6.16-6)

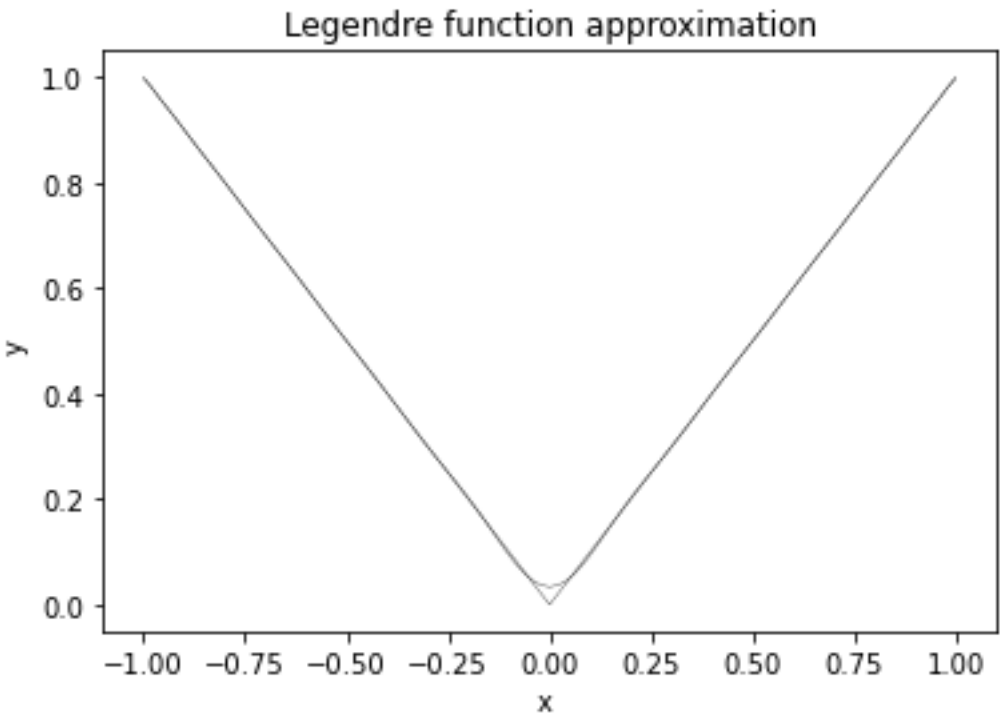
In order to achive numerical integration process, please referred to related section. As an example function let us assume **f(x)=|x|**

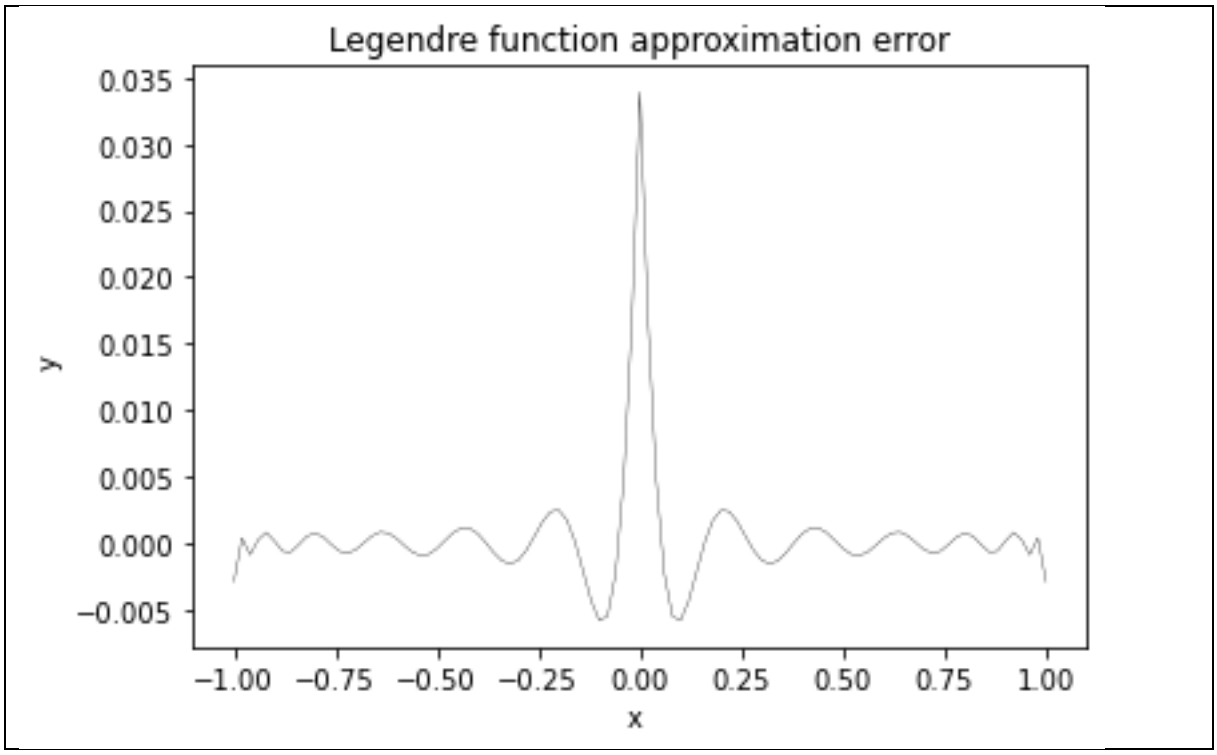
```
 #Legendre polynomial function fitting
from math import *
from f_x import *
import gauss
import numpy as np;
import matplotlib.pyplot as plt;
import os;

def P(k,x):
   #Legendre polynomial  -1<=x<=1
   y=1.0
   if k==0:
      y=1.0
   elif k==1:
      y=x
   else:
      y=((2.0*k-1.0)*P((k-1),x)*x-P((k-2),x)*float(k-1))/float(k)
   return y

def q(xmin,xmax,k,z):
   alpha=(xmax-xmin)/2.0
   beta=(xmax+xmin)/2.0
   x=(z-beta)/alpha
   return P(k,x)

def fak(f,xmin,xmax,k,z):
   # integral function
   ff=f.func(z)
   qq=q(xmin,xmax,k,z)
   a=(2.0*k+1)/(xmax-xmin)*ff*qq
   return a

def legendre(f,xmin,xmax,m):
     alpha=a=(xmax-xmin)/2.0
     beta=(xmax+xmin)/2.0
     b=[0.0 for i in range(m+1)];
     for j in range(m+1):
        b[j]=integral(f,xmin,xmax,30,j)
     return b

def gauss_legendre_coefficients(x1,x2,n):
   #calculates legendre gauss-coefficients as coefficients of the integral
   #for n terms
      eps=3e-15
      m=int((n+1.0)/2.0)
      xm=0.5*(x2+x1)
      xl=0.5*(x2-x1)
      nmax=100
      a=[[0.0 for i in range(n)] for j in range(2)]
      for i in range(1,m+1):
         z=cos(pi*((i-0.25)/(n+0.5)))
         for ii in range(nmax):
            #while abs(z-z1) > eps:
```

```
            p1=1.0;
            p2=0.0;
            for j in range(1,n+1):
                p3=p2
                p2=p1
                p1=(float)(((2.0*j-1.0)*z*p2-(j-1.0)*p3)/j)
            pp=(float)(n*(z*p1-p2)/(z*z-1.0))
            z1=z;
            z=z1-p1/pp
            if abs(z-z1) < eps: break
        a[0][i-1]=xm-xl*z
        a[0][n-i]=xm+xl*z
        a[1][i-1]=2.0*xl/((1.0-z*z)*pp*pp)
        a[1][n-i]=a[1][i-1]
    return a

def integral(f,x1,x2,n,k):
    #integral func(x)dx
    #integral of a function by using gauss-legendre quadrature
    #between x1 and x2
    a=gauss_legendre_coefficients(x1,x2,n)
    z=0.0
    yy=0.0
    for i in range(n):
        yy=fak(f,x1,x2,k,a[0][i])
        z=z+a[1][i]*yy
    return z


def funcLegendre(xmin,xmax,z,m):
    beta=(xmin+xmax)/2.0
    alpha=(xmax-xmin)/2.0
    x=(z-beta)/alpha;
    ff=0.0
    pk=0.0
    for j in range(m+1):
        pk=P(j,x);
        ff=ff+b[j]*pk;
    return ff

def plot_func(f,bk,xmin,xmax):
    nn=101
    m=len(bk)-1
    xx=[0.0 for j in range(nn)]
    zz=[0.0 for j in range(nn)]
    yy=[0.0 for j in range(nn)]
    yy1=[0.0 for j in range(nn)]
    alpha=(xmax-xmin)/2.0
    beta=(xmax+xmin)/2.0
    for i in range(nn):
        xx[i]=xmin+i*(xmax-xmin)/(nn-1)
        yy[i]=funcLegendre(xmin,xmax,xx[i],m)
        yy1[i]=f.func(xx[i])
    plt.title('Legendre function approximation')
    plt.xlabel('x ')
    plt.ylabel('y ');
    plt.plot(xx,yy,'k',xx,yy1,'k',linewidth=0.3)

def plot_error(f,bk,xmin,xmax):
    nn=101
    m=len(bk)-1
    xx=[0.0 for j in range(nn)]
    zz=[0.0 for j in range(nn)]
    yy=[0.0 for j in range(nn)]
    yy1=[0.0 for j in range(nn)]
    yy2=[0.0 for j in range(nn)]
    alpha=(xmax-xmin)/2.0
    beta=(xmax+xmin)/2.0
    for i in range(nn):
        xx[i]=xmin+i*(xmax-xmin)/(nn-1)
        yy[i]=funcLegendre(xmin,xmax,xx[i],m)
        yy1[i]=f.func(xx[i])
        yy2[i]=yy[i]-yy1[i]
    plt.title('Legendre function approximation ')
    plt.xlabel('x ')
    plt.ylabel('y ');
```

```
    plt.plot(xx,yy2,'k',linewidth=0.3)
n=25 #number of evaluation points
xmin=-1.0
xmax=1.0
class f1(f_x):func=lambda self,x: abs(x)
f = f1()
b=legendre(f,xmin,xmax,n)
print("b=",b)
plot_func(f,b,xmin,xmax)
#plot_error(f,b,xmin,xmax)
```

```
runfile('E:/okul/SCO1/legendre.py', wdir='E:/okul/SCO1')
Reloaded modules: f_x, gauss
b= [0.5004424966861193, 1.5612511283791264e-17, 0.6238875711657413, 3.469446951953614e-18, -0.18597854824309185,
2.0816681711721685e-17, 0.09969327644980225, 6.938893903907228e-18, -0.06420690744357288, -1.3877787807814457e-17,
0.045318387689030355, -1.3877787807814457e-17, -0.03373566285267805, 2.7755575615628914e-17, 0.025936642703607515, -
1.3877787807814457e-17, -0.020306723986429465, 0.0, 0.016004563758942075, 0.0, -0.012547099754515664, -
1.3877787807814457e-17, 0.009633136817453328, 0.0, -0.0070592684618817556, -6.938893903907228e-18]
```





The approximation can be applied by transforming data into an interpolation or curve fittimg program
Data (created from function **f(x)=|x|**)

-1	1
-0.95	0.95
-0.9	0.9
-0.85	0.85
-0.8	0.8
-0.75	0.75
-0.7	0.7
-0.65	0.65
-0.6	0.6
-0.55	0.55
-0.5	0.5
-0.45	0.45
-0.4	0.4
-0.35	0.35
-0.3	0.3
-0.25	0.25
-0.2	0.2
-0.15	0.15
-0.1	0.1
-0.05	0.05
3.19E-16	3.19E-16
0.05	0.05
0.1	0.1
0.15	0.15
0.2	0.2
0.25	0.25
0.3	0.3
0.35	0.35
0.4	0.4
0.45	0.45
0.5	0.5
0.55	0.55
0.6	0.6
0.65	0.65
0.7	0.7
0.75	0.75
0.8	0.8
0.85	0.85
0.9	0.9
0.95	0.95
1	1

```
#Legendre polynomial function fitting
from math import *
```

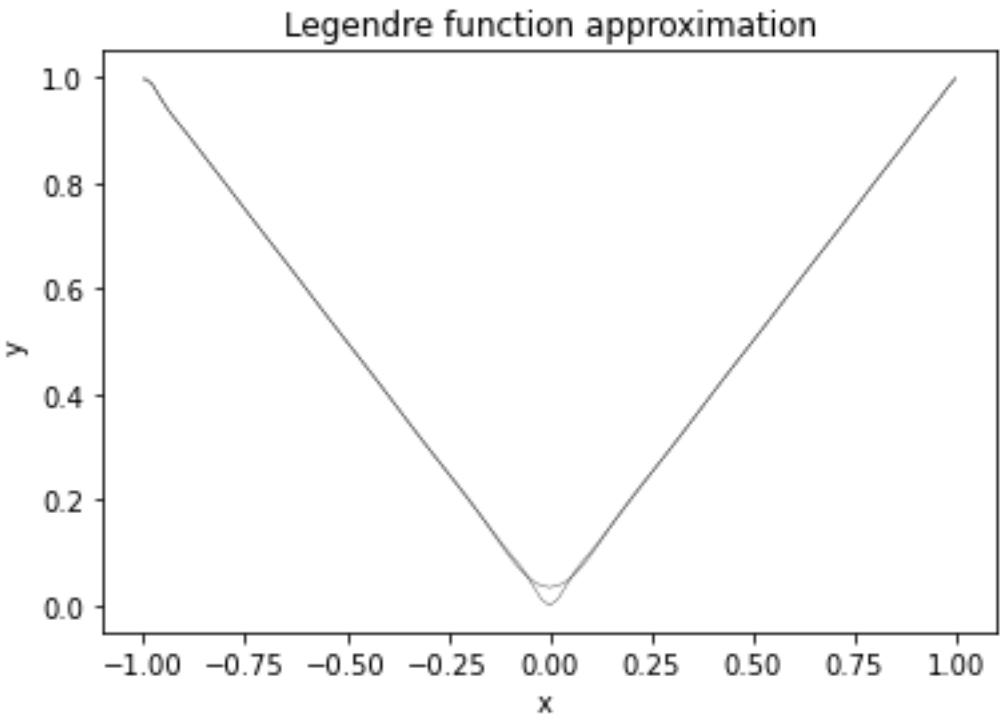
```
from f_x import *
import gauss
import numpy as np;
import matplotlib.pyplot as plt;
import os;

def P(k,x):
    #Legendre polynomial  -1<=x<=1
    y=1.0
    if k==0:
        y=1.0
    elif k==1:
        y=x
    else:
        y=((2.0*k-1.0)*P((k-1),x)*x-P((k-2),x)*float(k-1))/float(k)
    return y

def q(xmin,xmax,k,z):
    alpha=(xmax-xmin)/2.0
    beta=(xmax+xmin)/2.0
    x=(z-beta)/alpha
    return P(k,x)

def fak(f,xmin,xmax,k,z):
    # integral function
    ff=f.func(z)
    qq=q(xmin,xmax,k,z)
    a=(2.0*k+1)/(xmax-xmin)*ff*qq
    return a

def legendre(f,xmin,xmax,m):
      alpha=a=(xmax-xmin)/2.0
      beta=(xmax+xmin)/2.0
      b=[0.0 for i in range(m+1)];
      for j in range(m+1):
          b[j]=integral(f,xmin,xmax,30,j)
      return b

def gauss_legendre_coefficients(x1,x2,n):
    #calculates legendre gauss-coefficients as coefficients of the integral
    #for n terms
        eps=3e-15
        m=int((n+1.0)/2.0)
        xm=0.5*(x2+x1)
        xl=0.5*(x2-x1)
        nmax=100
        a=[[0.0 for i in range(n)] for j in range(2)]
        for i in range(1,m+1):
            z=cos(pi*((i-0.25)/(n+0.5)))
            for ii in range(nmax):
                #while abs(z-z1) > eps:
                p1=1.0;
                p2=0.0;
                for j in range(1,n+1):
                    p3=p2
                    p2=p1
                    p1=(float)(((2.0*j-1.0)*z*p2-(j-1.0)*p3)/j)
                pp=(float)(n*(z*p1-p2)/(z*z-1.0))
                z1=z;
                z=z1-p1/pp
                if abs(z-z1) < eps: break
            a[0][i-1]=xm-xl*z
            a[0][n-i]=xm+xl*z
            a[1][i-1]=2.0*xl/((1.0-z*z)*pp*pp)
            a[1][n-i]=a[1][i-1]
        return a

def integral(f,x1,x2,n,k):
        #integral func(x)dx
        #integral of a function by using gauss-legendre quadrature
        #between x1 and x2
        a=gauss_legendre_coefficients(x1,x2,n)
        z=0.0
        yy=0.0
        for i in range(n):
            yy=fak(f,x1,x2,k,a[0][i])
```

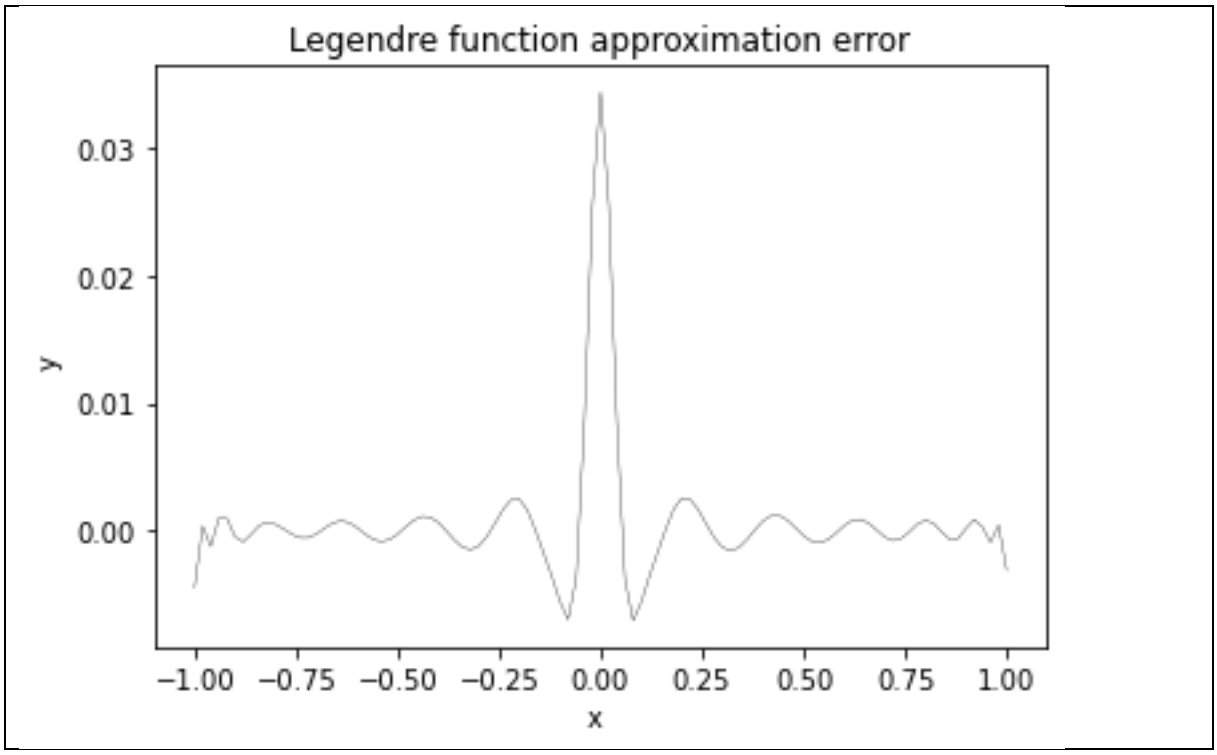
```
            z=z+a[1][i]*yy
        return z


def funcLegendre(xmin,xmax,z,m):
    beta=(xmin+xmax)/2.0
    alpha=(xmax-xmin)/2.0
    x=(z-beta)/alpha;
    ff=0.0
    pk=0.0
    for j in range(m+1):
        pk=P(j,x);
        ff=ff+b[j]*pk;
    return ff


#======= Cubic spline ===================
def thomas(f,e,g,r):
    n=len(f);
    x=[0.0 for i in range(n)];
    for k in range(0,n): e[k]=e[k]/f[k-1];f[k]=f[k]-e[k]*g[k-1];
    for k in range(1,n): r[k]=r[k]-e[k]*r[k-1];
    x[n-1]=r[n-1]/f[n-1];
    for k in range(n-2,-1,-1):  x[k]=(r[k]-g[k]*x[k+1])/f[k];
    return x;

def cubic_spline(xi,yi,c0,cn):
    n=len(xi);
    h=[0.0 for i in range(n)];
    w=[0.0 for i in range(n)];
    f=[0.0 for i in range(n)];
    e=[0.0 for i in range(n)];
    g=[0.0 for i in range(n)];
    d=[0.0 for i in range(n)];
    x=[0.0 for i in range(n)];
    S= [[0 for x in range(n)] for y in range(4)];
    for k in range(0,n-1):
         h[k]=xi[k+1]-xi[k];
         w[k]=(yi[k+1]-yi[k])/h[k];
    d[0]=c0;
    d[n-1]=cn;
    for k in range(0,n-1):
        d[k]=6.0*(w[k]-w[k-1]);
    f[0]=1.0;
    f[n-1]=1.0;
    g[0]=0.0;
    g[n-1]=0.0;
    e[0]=0.0;
    e[n-1]=0.0;
    for k in range(0,n-1):
         f[k]=2.0*(h[k]+h[k-1]);e[k]=h[k-1];g[k]=h[k];
    S[2]=thomas(f,e,g,d);
    S[3]=xi;
    for k in range(0,n-1):
        S[0][k]=(6.*yi[k+1]-h[k]*h[k]*S[2][k+1])/(6.0*h[k]);
        S[1][k]=(6.*yi[k]-h[k]*h[k]*S[2][k])/(6.0*h[k]);
    return S;


def funcSpline(S,x):
    n=len(S[0]);
    xx1=0;
    xx2=0;
    y=0;
    hk=0;
    for k in range(0,n-1):
        if S[3][k]<=x and x<=S[3][k+1]:
            hk=(S[3][k+1]-S[3][k]);
            xx1=(x-S[3][k]);
            xx2=(S[3][k+1]-x);
            y=S[0][k]*xx1+S[1][k]*xx2+(xx1*xx1*xx1*S[2][k+1]+xx2*xx2*xx2*S[2][k])/(6.0*hk);
            break;
    if y==0 and S[3][n-2]<=x:
          k=n-2;
          hk=(S[3][k+1]-S[3][k]);
          xx1=(x-S[3][k]);
```

```
        xx2=(S[3][k+1]-x);
        y=S[0][k]*xx1+S[1][k]*xx2+(xx1*xx1*xx1*S[2][k+1]+xx2*xx2*xx2*S[2][k])/(6.0*hk);
    return y;

def listSpline(xi,yi,numberofmidpoints,c0,cn):
    # numberofmidpoints :  in x--o--o--x--o--o--x chain if x's are esxperimental points
                # numberofmidpoints is 2
    n=len(xi);
    nn=(n-1)*(numberofmidpoints+1)+1;
    z=[[0 for x in range(nn)] for y in range(2)];
    S= [[0 for x in range(nn)] for y in range(4)];
    S=cubic_spline(xi,yi,c0,cn);
    dx=0;
    k=0;
    for i in range(0,n-1):
        z[0][k]=xi[i];z[1][k]=funcSpline(S,z[0][k]);k=k+1;
        for j in range(numberofmidpoints):
            dx=(xi[i+1]-xi[i])/(numberofmidpoints+1.0);
            z[0][k]=z[0][k-1]+dx;z[1][k]=funcSpline(S,z[0][k]);k=k+1;
    z[0][k]=xi[i+1];z[1][k]=funcSpline(S,z[0][k]);
    return z;

def read_array2D(d):
    path = d
    file1 = open(path,'r')
    file1.seek(0,2)  #jumps to the end
    endLocation=file1.tell()
    file1.seek(0) #jumps to the start
    nn=0
    m=0
    while True:
        line=file1.readline()
        ee=line.split()
        nn+=1
        if  file1.tell()==endLocation:
            break
    m=len(ee)
    a=[[0.0 for i in range(nn)] for j in range(m)]
    file1.seek(0) #jumps to the start
    for i in range(nn):
        line=file1.readline()
        ee=line.split()
        for j in range(m):
            a[j][i]=float(ee[j])
    file1.close()
    return a

def plot_func(f,bk,xmin,xmax):
    nn=101
    m=len(bk)-1
    xx=[0.0 for j in range(nn)]
    zz=[0.0 for j in range(nn)]
    yy=[0.0 for j in range(nn)]
    yy1=[0.0 for j in range(nn)]
    alpha=(xmax-xmin)/2.0
    beta=(xmax+xmin)/2.0
    for i in range(nn):
        xx[i]=xmin+i*(xmax-xmin)/(nn-1)
        yy[i]=funcLegendre(xmin,xmax,xx[i],m)
        yy1[i]=f.func(xx[i])
    plt.title('Legendre function approximation')
    plt.xlabel('x ')
    plt.ylabel('y ');
    plt.plot(xx,yy,'k',xx,yy1,'k',linewidth=0.3)

def plot_error(f,bk,xmin,xmax):
    nn=101
    m=len(bk)-1
    xx=[0.0 for j in range(nn)]
    zz=[0.0 for j in range(nn)]
    yy=[0.0 for j in range(nn)]
    yy1=[0.0 for j in range(nn)]
    yy2=[0.0 for j in range(nn)]
    alpha=(xmax-xmin)/2.0
    beta=(xmax+xmin)/2.0
    for i in range(nn):
```

```
        xx[i]=xmin+i*(xmax-xmin)/(nn-1)
        yy[i]=funcLegendre(xmin,xmax,xx[i],m)
        yy1[i]=f.func(xx[i])
        yy2[i]=yy[i]-yy1[i]
    plt.title('Legendre function approximation ')
    plt.xlabel('x ')
    plt.ylabel('y ');
    plt.plot(xx,yy2,'k',linewidth=0.3)


n=25 #number of evaluation points

a=read_array2D('d.txt')
x=a[0];
print("x=",x)
xmin=min(x)
xmax=max(x)
y=a[1];
print("y=",y)
c0=0.0
cn=0.0
S=cubic_spline(x,y,c0,cn)
class f1(f_x):func=lambda self,x: funcSpline(S,x)
f = f1()
b=legendre(f,xmin,xmax,n)
print("b=",b)
plot_func(f,b,xmin,xmax)
#plot_error(f,b,xmin,xmax)
```

```
runfile('E:/okul/SCO1/legendre_discrete.py', wdir='E:/okul/SCO1')
Reloaded modules: f_x, gauss
x= [-1.0, -0.95, -0.9, -0.85, -0.8, -0.75, -0.7, -0.65, -0.6, -0.55, -0.5, -0.45, -0.4, -0.35, -0.3, -0.25, -0.2, -0.15, -0.1, -0.05, 3.19189e-16,
0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95, 1.0]
y= [1.0, 0.95, 0.9, 0.85, 0.8, 0.75, 0.7, 0.65, 0.6, 0.55, 0.5, 0.45, 0.4, 0.35, 0.3, 0.25, 0.2, 0.15, 0.1, 0.05, 3.19189e-16, 0.05, 0.1, 0.15, 0.2,
0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95, 1.0]
b= [0.5005931355221859, -0.00031113156010302914, 0.6242815160968539, -0.0006814249464642576, -0.18500471624280213, -
0.0009412274800383522, 0.10053955717536847, -0.00102224713415755, -0.06306194616791921, -0.0008669643693506451,
0.04584539043818744, -0.0004597927286600634, -0.03340183515682857, 0.00014205037863022907, 0.025316861436946086,
0.0008008551274151232, -0.02126956966480424, 0.0013165895809865297, 0.014443077363651968, 0.0014732619307835926, -
0.013824604222774622, 0.0011410107168812605, 0.008749507331599585, 0.00041828861402271045, -0.007024376303811079, -
0.00032178719273900924]
```





PROBLEMS

PROBLEM 1

Fit a linear line by using polynomial least square curve fitting

X	**2**	**3**	**4**	**7**	**8**	**9**	**5**	**5**
Y	**9**	**6**	**5**	**10**	**9**	**11**	**2**	**3**

PROBLEM 2

For R22 refrigerant liquid viscosity values are required. The values are defined as the table below.

Fit a curve to input viscosity into computer as a function. Calculate the amount of error.

-101.1111111	8.6519780084E-04	-28.88888889	3.0093836551E-04	32.22222222	1.5253606713E-04
-95.55555556	7.7466826506E-04	-26.11111111	2.9143069737E-04	35	1.4798892150E-04
-90	6.9943367368E-04	-23.33333333	2.8233640610E-04	37.77777778	1.4302839899E-04
-84.44444444	6.3536025795E-04	-20.55555556	2.7365549171E-04	40.55555556	1.3848125336E-04
-78.88888889	5.8079451036E-04	-17.77777778	2.6538795420E-04	43.33333333	1.3393410773E-04
-73.33333333	5.3325616965E-04	-15	2.5712041668E-04	46.11111111	1.2938696209E-04
-70.55555556	5.1176057211E-04	-12.22222222	2.4926625605E-04	48.88888889	1.2483981646E-04
-67.77777778	4.9150510520E-04	-9.444444444	2.4182547228E-04	51.66666667	1.2070604770E-04
-65	4.7290314580E-04	-6.666666667	2.3479806540E-04	54.44444444	1.1615890207E-04
-62.22222222	4.5512794014E-04	-3.888888889	2.2777065851E-04	57.22222222	1.1202513331E-04
-59.44444444	4.3817948824E-04	-1.111111111	2.2074325162E-04	60	1.0747798768E-04
-56.66666667	4.2205779009E-04	1.666666667	2.1412922161E-04	62.77777778	1.0334421892E-04
-53.88888889	4.0717622256E-04	4.444444444	2.0792856848E-04	65.55555556	9.9210450167E-05
-51.11111111	3.9312140879E-04	7.222222222	2.0172791534E-04	68.33333333	9.5076681410E-05
-48.33333333	3.7947997189E-04	10	1.9552726220E-04	71.11111111	9.0529535778E-05
-45.55555556	3.6666528874E-04	12.77777778	1.8973998595E-04	73.88888889	8.6395767021E-05
-42.77777778	3.5426398247E-04	15.55555556	1.8395270969E-04	76.66666667	8.1848621388E-05
-40.81111111	3.4599644496E-04	18.33333333	1.7857881030E-04	79.44444444	7.7714852631E-05
-40	3.4268942995E-04	21.11111111	1.7320491092E-04	82.22222222	7.2754330123E-05
-37.22222222	3.3152825431E-04	23.88888889	1.6783101153E-04	85	6.8207184490E-05

-34.44444444	3.2078045554E-04	26.66666667	1.6287048902E-04	87.77777778	6.2833285106E-05
-31.66666667	3.1044603365E-04	29.44444444	1.5749658964E-04	90.55555556	5.7046008846E-05
				93.33333333	5.0018601959E-05

PROBLEM 3

Saturated vapor pressure for refrigerant R134a is required. These values are given below as table format. Fit a curve to this values that will minimize error.

-99	0.01	-49	2.06	0	32.73	50	212.61	100	786.93	150	2100.93
-98	0.01	-48	2.21	1	34.24	51	219.24	101	804.64	151	2137.75
-97	0.01	-47	2.37	2	35.81	52	226.02	102	822.65	152	2175.06
-96	0.01	-46	2.54	3	37.43	53	232.96	103	840.96	153	2212.87
-95	0.01	-45	2.72	4	39.12	54	240.07	104	859.56	154	2251.19
-94	0.02	-44	2.91	5	40.86	55	247.33	105	878.47	155	2290.01
-93	0.02	-43	3.11	6	42.67	56	254.77	106	897.69	156	2329.35
-92	0.02	-42	3.32	7	44.54	57	262.38	107	917.22	157	2369.21
-91	0.03	-41	3.54	8	46.48	58	270.15	108	937.07	158	2409.61
-90	0.03	-40	3.78	9	48.49	59	278.1	109	957.23	159	2450.54
-89	0.04	-39	4.03	10	50.57	60	286.23	110	977.71	160	2492.01
-88	0.04	-38	4.29	11	52.72	61	294.54	111	998.52	161	2534.04
-87	0.05	-37	4.57	12	54.94	62	303.03	112	1019.66	162	2576.63
-86	0.06	-36	4.86	13	57.23	63	311.7	113	1041.12	163	2619.8
-85	0.07	-35	5.17	14	59.61	64	320.57	114	1062.92	164	2663.54
-84	0.08	-34	5.5	15	62.06	65	329.62	115	1085.05	165	2707.88
-83	0.09	-33	5.84	16	64.59	66	338.86	116	1107.52	166	2752.81
-82	0.1	-32	6.2	17	67.2	67	348.3	117	1130.35	167	2798.36
-81	0.11	-31	6.58	18	69.9	68	357.93	118	1153.52	168	2844.54
-80	0.13	-30	6.98	19	72.68	69	367.77	119	1177.04	169	2891.36
-79	0.14	-29	7.4	20	75.55	70	377.81	120	1200.91	170	2938.83
-78	0.16	-28	7.84	21	78.52	71	388.05	121	1225.15	171	2986.97
-77	0.18	-27	8.3	22	81.57	72	398.5	122	1249.75	172	3035.81
-76	0.2	-26	8.78	23	84.71	73	409.17	123	1274.71	173	3085.29
-75	0.23	-25	9.29	24	87.95	74	420.04	124	1300.04	174	3135.59
-74	0.25	-24	9.82	25	91.29	75	431.13	125	1325.74	175	3186.64
-73	0.28	-23	10.38	26	94.73	76	442.45	126	1351.82	176	3238.47
-72	0.31	-22	10.96	27	98.27	77	453.98	127	1378.28	177	3291.12
-71	0.34	-21	11.58	28	101.92	78	465.74	128	1405.13	178	3344.62
-70	0.37	-20	12.22	29	105.66	79	477.72	129	1432.36	179	3399.02
-69	0.41	-19	12.88	30	109.52	80	489.93	130	1459.98	180	3454.38
-68	0.45	-18	13.58	31	113.49	81	502.38	131	1488	181	3510.78
-67	0.49	-17	14.32	32	117.57	82	515.06	132	1516.41	182	3568.29
-66	0.54	-16	15.08	33	121.76	83	527.98	133	1545.23	183	3627.05
-65	0.59	-15	15.88	34	126.07	84	541.15	134	1574.46		
-64	0.64	-14	16.71	35	130.49	85	554.55	135	1604.1		
-63	0.7	-13	17.58	36	135.04	86	568.21	136	1634.15		
-62	0.76	-12	18.48	37	139.71	87	582.11	137	1664.62		
-61	0.83	-11	19.43	38	144.51	88	596.27	138	1695.52		
-60	0.9	-10	20.41	39	149.43	89	610.68	139	1726.84		
-59	0.97	-9	21.44	40	154.48	90	625.35	140	1758.6		
-58	1.05	-8	22.5	41	159.66	91	640.28	141	1790.79		
-57	1.14	-7	23.61	42	164.98	92	655.47	142	1823.42		
-56	1.23	-6	24.77	43	170.43	93	670.94	143	1856.5		
-55	1.33	-5	25.97	44	176.03	94	686.67	144	1890.03		
-54	1.43	-4	27.22	45	181.76	95	702.68	145	1924.01		
-53	1.54	-3	28.52	46	187.63	96	718.96	146	1958.46		
-52	1.66	-2	29.87	47	193.65	97	735.53	147	1993.37		
-51	1.79	-1	31.27	48	199.82	98	752.38	148	2028.74		
-50	1.92			49	206.14	99	769.51	149	2064.6		

PROBLEM 4

Fit a parametric curve to the values given in the table P4

Table P.4 Girdi verisi

İ	0	1	2	3	4
T	0	1	2	3	4
X	-1	0	1	0	1
Y	0	1	0.5	0	-1

PROBLEM 5

X	y
0	-2.20
0.1	-1.90
0.2	0
0.3	-0.1
0.4	-0.01
0.5	-0.3
0.6	-0.2
0.7	-0.09
0.8	0.4
0.9	-0.1
1.0	-1.1
1.1	-1.3
1.2	-2
1.3	-1.8
1.4	-0.02
1.5	1

Data in the above table is given

a) Fit a polynomial least square curve
b) Use interpolation polynomials to fit the data

PROBLEM 6

X	y
0	0
1	1
2	4
3	8

Fit cubic spline to the data.

PROBLEM 7

X	y
0	0
1	1
2	0
1	-1

Fit cubic spline to the data.

PROBLEM 8

Ethylene gylicol-water mixture density values are given in the table below for different mixture percentages. By using formula

$$\rho(x,t)= a_0+a_1*x+a_2*t+a_3*x^2+a_4*t^2+a_5*xt+a_6*x^3+a_7*t^3+a_8*x^2t$$
$$+a_9*xt^2+a_{10}*x^4+a_{11}*t^4+a_{12}*x^2t^2+a_{13}x^3t+a_{14}*xt^3$$

	x %volume ratio ethylene gylicol concentration									
t	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%
-35							1090	1105	1119	1132
-30							1089	1104	1117	1131
-25						1088	1102	1116	1129	1142
-20					1072	1087	1101	1115	1128	1140
-15					1071	1086	1100	1113	1126	1138
-10				1054	1070	1084	1098	1111	1124	1136
-5			1037	1053	1068	1083	1096	1109	1122	1134
0	999.8	1019	1036	1052	1067	1081	1095	1108	1120	1132
5	999.9	1018	1034	1050	1065	1079	1093	1105	1118	1129
10	999.7	1016	1033	1049	1063	1077	1091	1103	1115	1127
15	999.1	1015	1031	1047	1062	1075	1089	1101	1113	1124
20	998.2	1013	1030	1045	1060	1073	1086	1098	1110	1121
25	997	1012	1028	1043	1058	1071	1084	1096	1107	1118
30	995.6	1010	1026	1041	1055	1069	1081	1093	1105	1115
35	994	1008	1024	1039	1053	1066	1079	1090	1102	1112
40	992.2	1006	1022	1037	1051	1064	1076	1088	1098	1109
45	990.2	1004	1020	1034	1048	1061	1073	1084	1095	1106
50	988	1002	1017	1032	1045	1058	1070	1081	1092	1102
55	985.7	999.2	1015	1029	1043	1055	1067	1078	1088	1098
60	983.2	996.7	1012	1026	1040	1052	1064	1075	1085	1095
65	980.5	994.1	1009	1023	1037	1049	1060	1071	1081	1091
70	977.7	991.4	1006	1020	1033	1045	1057	1067	1077	1087
75	974.8	988.5	1003	1017	1030	1042	1053	1064	1073	1083
80	971.8	985.5	1000	1014	1027	1038	1049	1060	1069	1078
85	968.6	982.4	997	1011	1023	1035	1046	1056	1065	1074
90	965.3	979.2	993.6	1007	1019	1031	1042	1052	1061	1070
95	961.9	975.8	990.1	1003	1016	1027	1038	1047	1056	1065
100	958.3	972.3	986.5	999.7	1012	1023	1033	1043	1052	1060
105	954.7	968.7	982.8	995.8	1008	1019	1029	1038	1047	1055
110	950.9	965	978.9	991.8	1004	1014	1025	1034	1042	1050
115	947.1	961.2	974.9	987.7	999.3	1010	1020	1029	1038	1045
120	943.1	957.2	970.8	983.4	994.9	1005	1015	1024	1032	1040
125	939	953.1	966.6	979.1	990.4	1001	1010	1019	1027	1035

PROBLEM 9

Liquid flows through a sphere. Friction drag coefficient C_D is given as a function of Reynolds number. Fit a cubic spline interpolation polynomial to the data. Find C_D values for *Re* = 5, 50, 500 and 5000 . Note: Logarithmic scale can be used.

Re	0.2	2	20	200	2000	20 000
C_D	103	13.9	2.72	0.800	0.401	0.433

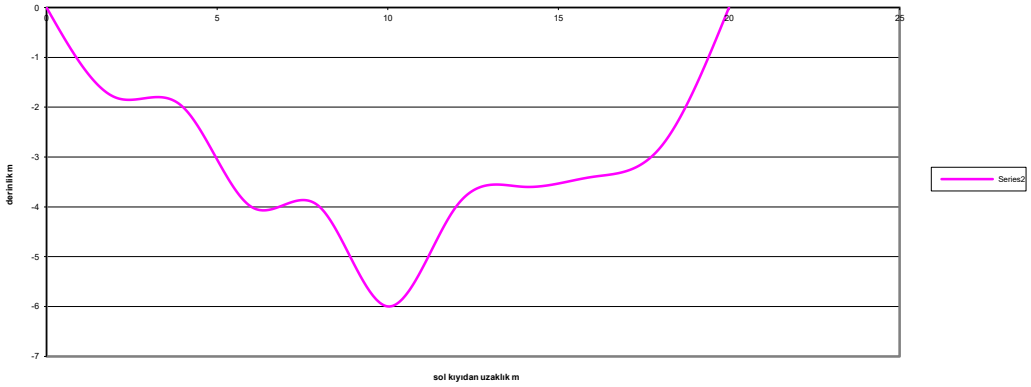
PROBLEM 10

Solve problem 9 by using 4 point Lagrange and Newton interpolation formula

PROBLEM 11

The depth and profile of a channel is given below

x, m	0	2	4	6	8	10	12	14	16	18	20	
H, m	0	1.8	2	4	4	6	4	3.6	3.4	2.8	0	
U m/s	0	0.03	0.045	0.055	0.065	0.12	0.07	0.06	0.05	0.04	0	



Find H(x) and U(x) curve fitting functions by using

a) Cubic spline interpolation formula
b) Polynomial least square
c) Lagrange interpolation formula

PROBLEM 12

Temperature distribution of a solid substance is given as

Time , t, s	**0**	**5**	**10**	**15**	**20**	**25**	**30**	**35**
Temperature T C	**90**	**55**	**40**	**28**	**24**	**22**	**21.5**	**20.6**

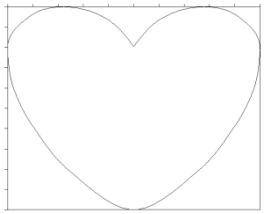
Find the temperature distribution at t=13 sn and t=23 sn t=33 sn, t=43sn ve t=53 sn deki sicaklik dağilimini bulunuz.

PROBLEM 13

A hard shape coordinates are given below, curvefit data to these coordinates

Xi	Yi	Xi	yi	Xi	yi	xi	Yi
241	127	331	157	215	246	149	120
258	107	323	178	199	232	160	106
283	96	312	197	184	217	175	97
307	97	298	217	170	197	199	96
322	106	283	232	159	178	224	107
333	120	267	246	151	157	241	127
334	138	241	258	148	138		

Note: the same values are repeated.



PROBLEM 14

Distance required to stop a car is function of velocity. This relation is given in the table below

Velocity (km/h)	**24**	**32**	**40**	**48**
Stopping distance (m)	**4.8**	**6.0**	**10**	**12**

If the vehicle travels with a velocity of 38 km/h, calculate stopping distance by using Newton interpolation formula.

PROBLEM 15

Thermal conductivity coefficient as a function of temperature is given in the table below.

Sicaklik ***T*** . [K]	275	290	305	320	340	360
Thermal conductivity coefficient ***k***. [W/(m·K)]	0.574	0.598	0.620	0.640	0.660	0.674

a-) Curve fit a quadratic polynomial to data

$$k = A \cdot T^2 + BT + C$$

b-) Estimate the thermal conductivity coefficient at 300 K

PROBLEM 16

For Etylenegylicol Prandtl (Pr) is given as a function of temperature in the below table. Derive a polynomial least square curve fitting and calculate Pr number for T= 42 °C

Temperature (K)	280	300	320	340
Pr number	400	151	73.5	42.8

PROBLEM 17

Saturated water temperature of water as a function of temperature t ºC will be curvefit to the following equation

$$h_f = A + Bt + Ct^2 + Dt^3$$

Following table is taken from the thermodynamic property table

t, ºC	10	30	50	70	90
h_f , kJ/kg	41.99	125.66	209.26	292.97	376.94

Calculate the coefficients by using the least square curve fitting

PROBLEM 18

According to Hooke's law Force applied to a linear spring is F=kx. In this equation k is spring constant, x is the displacement in the spring. Displacement versus force in a real spring is given by the below table find the k spring constant

X, m	0.02	0.04	0.06	0.08	0.1
F, kN	3.1	6.1	9.2	12	15.1

PROBLEM 19

Fit below values into function $f(x) = (Ax+B)^{-2}$

X	-1	0	1	2
Y	13.46	3.02	0.68	0.14

PROBLEM 20

2.7×10^7 W/m^3 heat is generated in 5 mm thick plate by a heat souce. Thermal conductivity of the plate is k=25 W/m K. Surfce temperatures of the plate (at x=0 mm) 161.5 °C, and (x=5 mm) 166.9 °C. Temperature distribution insde the plate is given in the table below. Give o polinomial function for the temperature distribution inside the plate. Use Newton interpolation Formula. Calculate temperature at x=4 mm.

x (mm)	0	0.5	2.5	4.5	5.0
T (°C)	161.5	166.9	177.7	171.22	166.9

PROBLEM 21

For very low pressure values (P<10 kPa) establish a Formula for the enthalpy for dry steam as h=A+BT

where A and B constants , h ,(kJ/kg) enthalpy and T , (ºC) is temperature.

h (kJ/kg)	2510.6	2519.8	2528.9	2538.1	2547.2	2556.3	2565.3
T (ºC)	5	10	15	20	25	30	35

PROBLEM 22

In a direct current electrical motor, Moment(F) and electrical current(I) is related with the relation

$I=A\,(F^B)$

Experimental measurements of F and I is given in the table below. Calculate A and B coefficients.

F (Nm)	100	200	300	400
I (Amper)	8	11.31	13.85	16

Bu ilişkiyi en küçük kareler yöntemi ile belirleyiniz.

PROBLEM 23

Derive and equation given the density of dry air ρ (kg/ m^3) at one athmosphere pressure by using Lagrange interpolation formula and then calculate the density at 22.5 °C.

t (°C)	**0**	**20**	**40**	**60**
ρ (kg/ m^3)	**1,2930**	**1,2045**	**1,1267**	**1,0595**

PROBLEM 24

For variable z the required function is given as

z=ax+by+cxy

By using least square method find a,b,c values and the function.

z	x	y
0.1	**1**	**1**
-0.9	**1**	**2**
2.0	**2**	**2**
-1.8	**3**	**1**

PROBLEM 25

Saturation pressure of water vapor(steam) changes as a function of temperature. One of the relation used for this rlation is known as "*Dupperet Formula*" is

$$P_{saturation} = A\left(\frac{t_{saturation}}{100}\right)^4$$

In this equation $t_{saturation}$:ºC is saturation temperature, $P_{saturation}$, kPa is saturation pressure. Find coefficient A.

t, °C	10	30	60	90	110	140	180	200	250	300	340	370
P kPa	1.2276	4.246	19.940	70.14	143.27	361.3	1002.1	1553.8	3973	8581	14586	21030

PROBLEM 26

The vibration(x) of a machine as a function of time(t) is measured and results are given as the table below.

i	**1**	**2**	**3**	**4**
t_i	0	2	4	6
x_i	5,0	3,7	2,7	2,0

For the vibration of this machine find an equation is in the form of

$$x(t) = ae^{-bt}$$

Where a and b are constants.

PROBLEM 27

In order to develop a new machine, shear stress (τ) is required to be measured. In the experiment the plate above is moved with a velocity of 2 m/s while the plate below remained constant. The distance between the plates are (h) 0.025 m . The shear stres is related to viscosity as follows:

$$\tau = \mu \times (V / h)$$

Viskosity (μ) N-s/m^2, is function of temperature and viscosity temperature relation is obtained in the laboratory as :

T (°C)	5	20	30	50	55
μ (N-s/m^2)	0,08	0,015	0,009	0,006	0,005

a) By using Lagrange interpolation find the shear stres at 38 °

b) By using a polynomial least square, obtain an eqaution for sheat stress

PROBLEM 28

A pump manufacturer is measured for the new pump the following data between volumetric flow rate(Q) and pump pressure (P)

Q (lt/s)	0,5	1,0	1,5	2,0	2,5	3,0	3,5
P (kPa)	275	290	280	255	229	205	178

d) Fit a second degree least square polynomial [$P=f(Q)=aQ^2+bQ+c$] b-) Pompa 2,2 lt/s debide su basarken oluşacak basinci tahmin ediniz.

b) Find the flow rate fort he maximum pressure?

PROBLEM 29

Yanma gazlari yoğunluğunun (ρ: kg/m^3), sicaklikla (T:°C) değişimi aşağida verilmiştir.

ρ: kg/m^3	0,950	0,748	0,617	0,525
T:°C	100	200	300	400

Newton enterpolasyon yöntemini kullanarak 1'den 4'ü dereceye kadar polinom

Denklemlerini bulunuz. 285 °C'da yanma gazlarinin yoğunluğunu hesaplayiniz.

PROBLEM 30

An orifice is used to measure volumetric flow rate. Such a device is shown in the figure. Following data is used to design the orifice-meter.

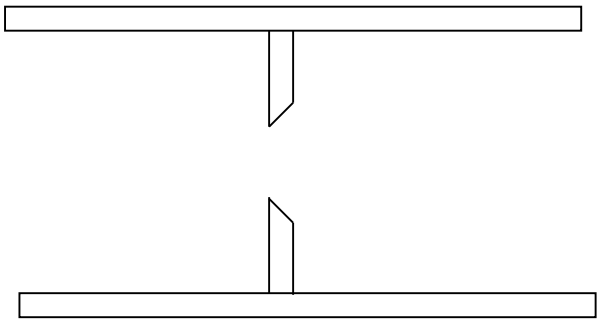
A : Orifice cross sectional area.

A_1 : pipe cross sectional area<.

A_2 : C_cA (shrinked area behind the orifice).

C_c is shrinkage coefficient is a function of (A/A_1) ratio. Experimental results fort his coefficient is given below: Find a least sqaure cirve fiitng equation for [$C_c=f(A/A_1)$]

A/A_1	C_c
0.1	0.62
0.2	0.63
0.3	0.64
0.4	0.66
0.5	0.68
0.6	0.71
0.7	0.76
0.8	0.81
0.9	0.89
1.00	1.00



PROBLEM 31

The life of the cutting tool for a lathe is T(minutes) and cutting speed V(m/minutes). The life test is carried out and the following data is obtained.

Cutting speed, V (m/minutes)	Life of the cutting tool,T (minutes)
140	80
160	70
180	55
200	50
220	35

If the general relation between cutting speed and life of the cutting tool is:

$$\mathbf{V=(b/T)^{(1/a)}}$$

Find the coefficients by using least square curve fitting

PROBLEM 32

Fit the data below the following equation by using least square curve fitting

$$y=ax^b$$

by using least square curve fitting

X	1	2	3	4	5
Y	1	1	2	4	4

PROBLEM 33

By using data in the following table answer the quaestions below.

X	1	2	3	5	6
f(X)	4,75	4	5,25	19,75	36

a) Obtain fourth order polynomial equation by using Newton interpolation formula

b) Obtain cubic spline for the second subregion

c) PROBLEM 34

Fit a line to following data by using least square method.

x	5	10	15	20	25	30	35	40	45	50
y	16	25	32	33	38	36	3	40	42	42

PROBLEM 35

Kinematic viscosities of oil SAE30 is given in the table below.

Sicaklik T(C)	5	10	15	20	25	30	35	40	45	50
Vizkozite μ (m^2/s)	16	25	32	33	38	36	3	40	42	42

a) Fit the data into $\mu=ae^{(b/T)}$ equation

b) Fit the data into $\mu = \dfrac{a\sqrt{T}}{1+\dfrac{b}{T}}$ equation

PROBLEM 36

For $f(x) = \dfrac{20}{1+2x^2}$ function in the range of -3<=x<=3 by using points x_i=-3,-2,-1,0,1,2,3

a) Find cubic spline interpolation formulas

e) Find quadratic B-spline formulas

PROBLEM 37

In a smooth sphere, the following relation is existed between Reynolds number(Re) and friction coefficient (C)

Re	0.1	1	10	100	1000	10000
C	210	30	4	1	0.5	0.4

Find an equation to represent the data properly

PROBLEM 38

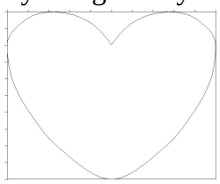
By using bezier curve construct the letter α.

PROBLEM 39

By using bezier curve construct the letter γ .

PROBLEM 40

By using two symmetric bezier curve construct the hard shape.



PROBLEM 41

The tuition for 12 units at *St. Anford* University has been increasing from 1998 to 2008 as shown in the table below:

Year Tuition per year

1998 $21,300
1999 $23,057
2000 $24,441
2001 $25,917
2002 $27,204
2003 $28,564

2004 $29,847
2005 $31,200
2006 $32,994
2007 $34,800
2008 $36,030

(a) Plot the given data points and intuitively interpolate (draw) a smooth curve through them.

(b) Interpolate the data with the Lagrange polynomial. Plot the polynomial and the data points. Use the polynomial to predict the tuition in 2010. This is an extrapolation problem; discuss the utility of Lagrange polynomials for extrapolation.

(c) Repeat with a cubic spline interpolation and compare your results.

PROBLEM 42

The concentration of a certain toxin in a system of lakes downwind of an industrial area has been monitored very accurately at intervals from 1993 to 2007 as shown in the table below. It is believed that the concentration has varied smoothly between these data points.

Year Toxin Concentration

1993 12.0
1995 12.7
1997 13.0
1999 15.2
2001 18.2
2003 19.8
2005 24.1
2007 28.1
2009 ???

(a) Interpolate the data with the Lagrange polynomial. Plot the polynomial and the data points. Use the polynomial to predict the condition of the lakes in 2009. Discuss this prediction.

(b) Interpolation may also be used to fill "holes" in the data. Say the data from 1997 and 1999 disappeared. Predict these values using the Lagrange polynomial fitted through the other known data points.

(c) Repeat (b) with a cubic spline interpolation. Compare and discuss your results.

PROBLEM 43

Fit Pade rational approximation to the function sin(x) between [-π,π] with polynomial

$$r(x) = \frac{p(x)}{q(x)} = \frac{p_0 + p_1 x + p_2 x^2 + p_3 x^3}{q_0 + q_1 x + q_2 x^2}$$

p(x) is 3th degree polynomial and q(x) is 2nd degree polynomial.

PROBLEM 44

Fit Pade rational approximation to the function |x| between [-1,1] with polynomial

$$r(x) = \frac{p(x)}{q(x)} = \frac{p_0 + p_1 x + p_2 x^2 + p_3 x^3}{q_0 + q_1 x + q_2 x^2 + q_3 x^3}$$

p(x) is 3th degree polynomial and q(x) is 3rd degree polynomial.

PROBLEM 45

Fit Chebyschev rational approximation to the function sin(x) between [-π,π] with polynomial

$$r(x) = \frac{p(x)}{q(x)} = \frac{p_0 To(x) + p_1 T_1(x) + p_2 T_2(x) + p_3 T_3 x^3}{q_0 To(x) + q_1 T_1(x) + q_2 T_2(x)}$$

p(x) is 3th degree polynomial and q(x) is 2nd degree polynomial.

PROBLEM 46

Fit Chebyschev rational approximation to the function sin(x) between [-1,1] with polynomial

$$r(x) = \frac{p(x)}{q(x)} = \frac{p_0 To(x) + p_1 T_1(x) + p_2 T_2(x) + p_3 T_3 x^3}{q_0 To(x) + q_1 T_1(x) + q_2 T_2(x) + q_3 T_3(x)}$$

p(x) is 3th degree polynomial and q(x) is 3th degree polynomial.

PROBLEM 47

Fit Trigonometric(Fourier) approximation to the function $f(x) = x^2 - 2 * x + 5$ between [-π,π] with degree 5 function $S_5(x)$

PROBLEM 48

Fit Trigonometric(Fourier) approximation to the function $f(x) = |x|$ between [-1,1] with degree 10 function $S_{10}(x)$

PROBLEM 49

Average temperature in a city is measured as follows

Months x_i	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Average Temperature T_i	**-6**	**-3**	**2**	**9**	**15**	**21**	**23**	**22**	**18**	**12**	**5**	**-2**

Fit a discrete Trigonometric (Fourier) least square curve to this data.

PROBLEM 50

For CO2 Specific heat , Cp (KJ/kmolK) data is given as a function of temperature (degree K) as

T (K)	**Cp(KJ/kmolK)**
298.15	37.132
300	37.217
400	41.326
500	44.625
600	47.323
700	49.563
800	51.434
900	52.999
1000	54.308
1100	55.412
1200	56.342
1300	57.13
1400	57.803
1500	58.381
1600	58.883
1700	59.321
1800	59.705
1900	60.046
2000	60.349
2100	60.62
2200	60.864
2300	61.086
2400	61.288
2500	61.473
2600	61.644
2700	61.804
2800	61.953
2900	62.094
3000	62.228

Find an equation for Cp. Investigate different kind of curve fitting equations for a best result.

PROBLEM 51

Saturation vapor pressure equation for HCFC-123 Refrigerant(2,2 dichloro-1,1,1-trifloroethane) is given as: $\log_{10} P_{sat} = A + B/T + C\log_{10} T + DT + E((F-T)/T)\log_{10}(F-T)$

Curve fit values are given in the book as:

A=1.656333e03 B=-2.480583e06,C=1.792522e1,D=-8.86838e-2,E=4.61781e2,F=1.666667e3

The data in the tables are given as T (degree K) and P_{sat}(kPa):

-80	0.13	18	69.95	50	212.46	84	540.19
-70	0.34	20	75.61	52	225.84	86	567.2
-60	0.81	22	81.63	54	239.85	88	595.2
-50	1.77	24	88.02	56	254.51	90	624.23
-40	3.58	26	94.8	58	269.85	92	654.3
-30	6.75	27.82	101.33	60	285.89	94	685.44
-20	12	28	101.98	62	302.64	96	717.68
-10	20.25	30	109.58	64	320.13	98	751.03
0	32.65	32	117.62	66	338.38	100	785.53
2	35.74	34	126.11	68	357.4	110	976.03
4	39.07	36	135.07	70	377.22	120	1199
6	42.64	38	144.52	72	397.87	130	1457.8
8	46.47	40	154.47	74	419.36	140	1756.3
10	50.57	42	164.95	76	441.71	150	2098.7
12	54.95	44	175.97	78	464.94	160	2490.1
14	59.63	46	187.55	80	489.09	170	2937.2
16	64.63	48	199.71	82	514.16	180	3450.6
						183.68	3661.8

Make a non-linear curve fit for a given data and check if the given coefficients are correct.

11.0 DIFFERENTIAL EQUATIONS

11.1 General definition/analytic solutions

$$\frac{dy}{dx} = f(x, y) = yx^2$$

If x=0 y=1 (initial conditions), we can solve this equation analytically as:

$$\frac{dy}{y} = x^2 dx$$

By integrating both sides

$$\ln(y) = \frac{x^3}{3} + C$$

Applying initial condition gives:

$$C = \ln(1) = 0$$

So

$$y = \exp\left(\frac{x^3}{3}\right)$$

Euler method for numerical solutions:

Figure Leonhard Euler

It is desired the find solution of (y values) differential equation

$$\frac{dy}{dx} = f(x, y)$$

The first simplest method to solve differential equation numerically is the euler equation. Solution set in euler equation can be written as:

$$\frac{y_{i+1} - y_i}{h} \approx f(x, y)$$

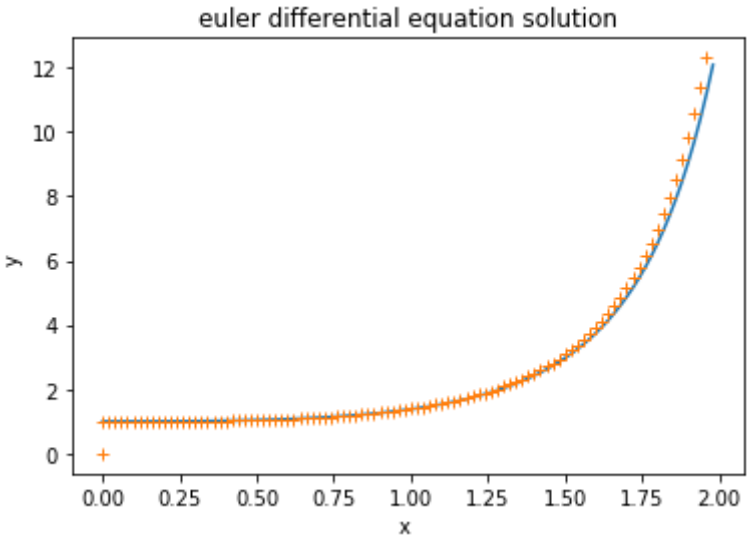
$y_{i+1} = y_i + f(x_i, y_i)h$

In the equation h is step size. H can be calculated as $h = \Delta x = x_{i+1} - x_i$

```
from math import *
import matplotlib.pyplot as plt;

def f(x,y):
    z=y*x*x
    return z
```

```
def g(x):
    z=exp(x*x*x/3.0)
    return z

def euler(f1,x0,y0,xn,n):
    x =[[0 for i in range(n)] for j in range(2)]
    x[0][0]=x0
    x[1][0]=y0
    dx=(xn-x0)/n
    for i in range(n-1):
        x[1][i+1]=x[1][i]+f1(x[0][i],x[1][i])*dx
        x[0][i+1]=x0+dx*(i+1.0)
    return x

x0=0
x1=2
y0=1
n=100
x=euler(f,x0,y0,x1,n)
print("x=",x);
xx=[0 for i in range(n)]
yy=[0 for i in range(n)]
dx=(x1-x0)/n
for i in range(n-1):
    xx[i]=x0+dx*i
    yy[i]=g(xx[i])

plt.title('euler differential equation solution')
plt.xlabel('x ')
plt.ylabel('y ');
plt.plot(x[0],x[1],'-',xx,yy,'+');
```

```
runfile('E:/okul/SCO1/euler.py', wdir='E:/okul/SCO1')
x= [[0, 0.02, 0.04, 0.06, 0.08, 0.1, 0.12, 0.14, 0.16, 0.18, 0.2, 0.22, 0.24, 0.26, 0.28, 0.3, 0.32, 0.34, 0.36, 0.38, 0.4, 0.42, 0.44, 0.46, 0.48,
0.5, 0.52, 0.54, 0.56, 0.58, 0.6, 0.62, 0.64, 0.66, 0.68, 0.7000000000000001, 0.72, 0.74, 0.76, 0.78, 0.8, 0.8200000000000001, 0.84, 0.86,
0.88, 0.9, 0.92, 0.9400000000000001, 0.96, 0.98, 1.0, 1.02, 1.04, 1.06, 1.08, 1.1, 1.12, 1.1400000000000001, 1.16, 1.18, 1.2, 1.22, 1.24,
1.26, 1.28, 1.3, 1.32, 1.34, 1.36, 1.3800000000000001, 1.4000000000000001, 1.42, 1.44, 1.46, 1.48, 1.5, 1.52, 1.54, 1.56, 1.58, 1.6, 1.62,
1.6400000000000001, 1.6600000000000001, 1.68, 1.7, 1.72, 1.74, 1.76, 1.78, 1.8, 1.82, 1.84, 1.86, 1.8800000000000001,
1.9000000000000001, 1.92, 1.94, 1.96, 1.98], [1, 1.0, 1.000008, 1.000040000256, 1.0001120031360184, 1.0002400174724198,
1.0004400654759142, 1.0007281922147713, 1.0011204776661196, 1.0016330513506846, 1.00228210956796, 1.0030839352556142,
1.0040549205049416, 1.0052115917733633, 1.0065706378454409, 1.0081489406055826, 1.0099636086986725, 1.0120320141692873,
1.0143718321860467, 1.017001083975073, 1.019938183105593, 1.0232019852915308, 1.0268118418956393, 1.0307876573474593,
1.0351499507133537, 1.039919921686241, 1.0451195212946722, 1.0507715276658338, 1.056899627215181, 1.0635285016770746,
1.070683921436358, 1.0783928456706997, 1.086683529868216, 1.0955856413448966, 1.1051303834522932, 1.11535062923846,
1.126281065404997, 1.1379583474911161, 1.150421267312839, 1.1637109337928369, 1.177870968435228, 1.1929477168311988,
1.2089904777271447, 1.2260517513488303, 1.2441875088547822, 1.2634574849919251, 1.2839254962487943, 1.3056597870492939,
1.328733406806029, 1.3532246209602778, 1.3792173594796828, 1.4068017066692764, 1.4360744365816507, 1.467139598793785,
1.5001091598578788, 1.5351037063390434, 1.5722532160324483, 1.6116979047162703, 1.6535891566556555, 1.6980905480395725,
1.7453789736213785, 1.7956458880616741, 1.849098674857494, 1.9059621573067116, 1.9664802677255142, 2.030917893138344,
2.09956291792642, 2.17272848649032, 2.25075551189716, 2.33401545979326, 2.4229134406258654, 2.5178916474983994,
2.619433181858715, 2.72806631477676, 2.8443692379083227, 2.9689753654826108, 3.1025792569293285, 3.245943239233519,
3.399904818956843, 3.5653849863051104, 3.743397527901352, 3.935059481329901, 4.141602883385945, 4.364387985689041,
4.604918136356336, 4.864856555317378, 5.146045264214723, 5.450526470407779, 5.780566749243912, 6.13868442049307,
6.527680574850875, 6.9506742761012115, 7.411142545544364, 7.9129658295882646, 8.460479761269136, 9.05853415463373,
9.712560320598286, 10.428647967915357, 11.21363315775628, 12.075199020533011]]
```



11-2 RUNGE KUTTA METHODS

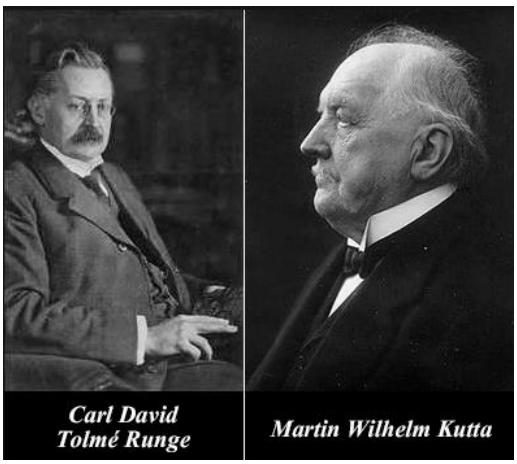


Figure 8.4-1 Carl Davis Tolme Runge and Martin Wilhelm Kutta

Runge-Kutta method as in most general form can be given as:

For differential form $\frac{dy}{dx} = f(x, y)$

$y_{n+1} = y_n + h*\phi(x_i,y_i,h)$

$$\phi(x_1, y_1, h) = \sum_{i=1}^{n} b_i k_i$$

In this equation a is constant values depends on polynamial degree of Runge-Kutta equation and coefficients k, can be defined as:

$k_1=f(x_i,y_i)$

$k_2=f(x_i+c_2h,y_i+a_{21}k_1h)$

$k_3=f(x_i+c_3h,y_i+a_{31}k_1h+ q_{32}k_2h)$

….

$k_n=f(x_i+c_nh,y_i+a_{n,1}k_1h+ \ldots+ a_{n,n-1}k_{n-1}h)$

In more generalized form, these equations can be written as:

$x_i = x_n + c_i h$ i=1…s

$$y_i = y_n + h\sum_{j=1}^{i-1} a_{ij}k_j$$

$$k_i = f(x_i, y_i)$$

$$y_{n+1} = y_n + h\sum_{j=1}^{s} b_i k_i$$ where s is the degree of coefficients of equations. In numerical analysis books, coefficients are usually shown as Buthcher tableu which puts the coefficients of the method in a table as follows:

$$\begin{array}{c|ccccc} c_1 & a_{11} & a_{12} & a_{13} & \dots & a_{1s} \\ c_2 & a_{21} & a_{22} & a_{23} & \dots & a_{2s} \\ c_3 & a_{31} & a_{32} & a_{33} & \dots & a_{3s} \\ \vdots & \vdots & \vdots & \vdots & \dots & \vdots \\ c_s & a_{s1} & a_{s2} & a_{s3} & \dots & a_{s4} \\ \hline & b_1 & b_2 & b_3 & \dots & b_s \end{array}$$

As it is seen from the equation the definition is very general. Depends on polynamial degree of Runge-Kutta equation accuracy will improve. For example Runge-Kutta equation derived by using **a second degree polynomial (RK2)** has the following form

$y_{i+1} = y_i + (0.5k_1 + 0.5k_2)h$

$k_1=f(x_i,y_i)$

$k_2=f(x_i+h,y_i+k_1h)$

This equation can be given as Buthcher tableu as:

$$\begin{array}{c|cc} 0 & 0 & 0 \\ 1 & 1 & 0 \\ \hline & 0.5 & 0.5 \end{array}$$

Second degree Ralston method:

$y_{i+1} = y_i + (1.0/3.0k_1 + 2.0/3.0k_2)h$

$k_1=f(x_i,y_i)$

$k_2=f(x_i+(3.0/4.0)h,y_i+(3.0/4.0)k_1h)$

This equation can be given as Buthcher tableu as:

$$\begin{array}{c|cc} 0 & 0 & 0 \\ 3/4 & 3/4 & 0 \\ \hline & 1/3 & 2/3 \end{array}$$

Runge-Kutta with third degree polynomial solution RK3:

$y_{i+1} = y_i + ((1/6) k_1 + (4/6)k_2 +(1/6)k_3)h$

$k_1=f(x_i,y_i)$

$k_2=f(x_i+0.5h,y_i+0.5k_1h)$

$k_3=f(x_i+h,y_i - k_1h +2k_2h)$

This equation can be given as Buthcher tableu as:

$$\begin{array}{c|ccc} 0 & 0 & 0 & 0 \\ 1/2 & 1/2 & 0 & 0 \\ 1 & -1 & 2 & 0 \\ \hline & 1/6 & 4/6 & 1/6 \end{array}$$

Runge-Kutta with fourth degree polynomial solution RK4:

$y_{i+1} = y_i + (1/6)*(k_1 + 2k_2 +2k_3+k_4)h$

$k_1=f(x_i,y_i)$

$k_2=f(x_i+0.5h,y_i+0.5k_1h)$
$k_3=f(x_i+0.5h,y_i+0.5k_2h)$
$k_4=f(x_i+h,y_i+k_3h)$
This equation can be given as Buthcher tableu as:

$$\begin{array}{c|cccc} 0 & 0 & 0 & 0 & 0 \\ 1/2 & 1/2 & 0 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ \hline & 1/6 & 2/6 & 2/6 & 1/6 \end{array}$$

Runge-Kutta with sixth degree polynomial solution RK6:
$y_{i+1} = y_i + (1/90)*(7k_1 + 32k_3 +12k_4+32k_5+7k_6)h$
$k_1=f(x_i,y_i)$
$k_2=f(x_i+0.25h , y_i+0.25k_1h)$
$k_3=f(x_i+0.25h , y_i+0.125k_1h+0.125k_2h)$
$k_4=f(x_i+0.5h , y_i – 0.5k_2h+k_3h)$
$k_5=f(x_i+0.75h , y_i + (3/16)k_1h+(9/16)k_4h)$
$k_6=f(x_i+h , y_i - (3/7)k_1h+(2/7)k_2h+(12/7)k_3h - (12/7)k_4h+(8/7)k_5h)$

This equation can be given as Buthcher tableu as:

$$\begin{array}{c|ccccc} 0 & 0 & 0 & 0 & 0 & 0 \\ 1/4 & 1/4 & 0 & 0 & 0 & 0 \\ 1/4 & 1/8 & 1/8 & 0 & 0 & 0 \\ 2/4 & 1/4 & -1/4 & 1 & 0 & 0 \\ 3/4 & 3/16 & 0 & 0 & 9/16 & 0 \\ 1 & -3/7 & 2/7 & 12/7 & -12/7 & 8/7 \\ \hline & 1/90 & 7/90 & 32/90 & 12/90 & 7/90 \end{array}$$

Runge-Kutta-Verner methods with different degree of accuricies
For the degrees a general formula of Runge_Kutta equations can be written as:

$$x_i = x_n + c_i h \quad \text{i=1…s}$$

$$y_i = y_n + h\sum_{j=1}^{i-1} a_{ij}k_j$$

$$k_i = f(x_i, y_i)$$

$$y_{n+1} = y_n + h\sum_{j=1}^{s} b_i k_j \text{ or}$$

$$y_{n+1} = y_n + h\sum_{j=1}^{s} bh_i k_j$$

Verner[42-46] given coefficient of a pair of solutions (b and bh). For the constant step size any of these equations can be used, in the coming sections utilisation of these type of equations as a base of variable step size calculations will also be introduced. Due to growing sizes of equations Buthcher tableus will be given directly as table values for the remaining equations.

Table 8.4-1 9th degree Runge-Kutta-Verner coefficients

c[1] = 0

c[2] = 3/50
c[3] = 1439/15000
c[4] = 1439/10000
c[5] = 4973/10000
c[6] = 389/400
c[7] = 1999/2000
c[8] = 1
c[9] = 1

a[2,1] = 3/50

a[3,1] = 519479/27000000
a[3,2] = 2070721/27000000

a[4,1] = 1439/40000
a[4,2] = 0
a[4,3] = 4317/40000

a[5,1] = 109225017611/82828840000
a[5,2] = 0
a[5,3] = -417627820623/82828840000
a[5,4] = 43699198143/10353605000

a[6,1] = -80368152926439073494525521723 69/
191934985946683241245914401600
a[6,2] = 0
a[6,3] = 246134619571490020064824665/154381649665540511760236 8
a[6,4] = -138804959568856862340740672 79/113663489566254201783474344
a[6,5] = 755005057777788994734129/136485922925633667082436

a[7,1] = -1663299841566102097180506666498880934230261/305584245061561703070209577913113842320 00
a[7,2] = 0
a[7,3] = 1308381241952854917990436288110930 33/631862949514135618861563657970240
a[7,4] = -3287100453856023634160618787153901962873/
207243149153767556291357110268514092 00
a[7,5] = 277182679014033214086524252036924 1/396438716042723436917079980147600
a[7,6] = -1799166916139193/96743806114007800

a[8,1] = -832144750039369683895428386437986853923637763/152229745500696007487636518446676199452048 87
a[8,2] = 0
a[8,3] = 818622075710363565982285196611368750/393657623790372815185607239534 3129
a[8,4] = -98189851654916584648411945813854634347937418 75/616425979626589940698693709231964635818660 11
a[8,5] = 317966921418485587204257110425481347693 75/453025403350004597555785801600630862809 2
a[8,6] = -14064542118843830075/76692874826430685364 4
a[8,7] = -142467030483628812 5/278283910476476808821 7

a[9,1] = 382735282417/11129397249634
a[9,2] = 0
a[9,3] = 0
a[9,4] = 5535620703125000/21434089949505429
a[9,5] = 13867056347656250/32943296570459319
a[9,6] = 626271188750/142160006043
a[9,7] = -51160788125000/289890548217
a[9,8] = 163193540017/946795234

b[1] = 382735282417/11129397249634
b[2] = 0
b[3] = 0
b[4] = 5535620703125000/21434089949505429
b[5] = 13867056347656250/32943296570459319
b[6] = 626271188750/142160006043
b[7] = -51160788125000/289890548217
b[8] = 163193540017/946795234
b[9] = 0

bh[1] = 1243106378698856756467986 13/289007246878946642659682767 0
bh[2] = 0
bh[3] = 0
bh[4] = 2658631517371649903613309218 75/111319727146337230394031936 9579
bh[5] = 3075493557174030806536302953125/684374992204232387654694969 9876
bh[6] = 677980000087338798132630 55/295327921476667375500363 72
bh[7] = -1099436585155390846238326375/150557064964464088591961 67
bh[8] = 2617125265308637318157180 2/368794478890732346033505
bh[9] = 1/30

Table 8.4-2 10th degree Runge-Kutta-Verner coefficients

c[1] = 0
c[2] = 1/200
c[3] = 49/450
c[4] = 49/300
c[5] = 91/200
c[6] = 34704460/57271701
c[7] = 167/200
c[8] = 183/200
c[9] = 1
c[10] = 1

a[2,1] = 1/200

a[3,1] = -4361/4050
a[3,2] = 2401/2025

a[4,1] = 49/1200
a[4,2] = 0
a[4,3] = 49/400

a[5,1] = 1781/2800
a[5,2] = 0
a[5,3] = -13689/5600
a[5,4] = 507/224

a[6,1] = -2757769235683215548894853133264 60/10878254403907579741558876498 2099
a[6,2] = 0
a[6,3] = 95760015121587056480974387300 00/92976533366731450782554499984 7
a[6,4] = -221785384656429549022904586896 00/27892960010019435234766349995 41
a[6,5] = 12323686750109283453854913779200/1554036343415368534508410928 3157

a[7,1] = 822150801447127646499 3/820610858494509040000 0
a[7,2] = 0
a[7,3] = -9852144759099/2364568872400
a[7,4] = 15322550932778398907299/3996134347464283007200
a[7,5] = -18338463121898520004/36506562121215938675
a[7,6] = 2334047554460212511930751137351938349 9/3495732942589377959866017554367280 0000

a[8,1] = 810886430227405450337307801 69/297518211023193714015280 0000
a[8,2] = 0
a[8,3] = -2837794586103/67559110640
a[8,4] = -1416757560688131609503834114 1/1344719188593468553337836000
a[8,5] = 2395552232834276839307772/297600628643880682688 75
a[8,6] = -4076715891031001341580357765362043260356514682697/605358015235585136339810926359877215071864 00000
a[8,7] = 36551527355459957808/2801171464968864455

a[9,1] = -3347747115771808477876752833599/1101327591307901549464211073280
a[9,2] = 0
a[9,3] = 1214704878477625125/119815105452943264
a[9,4] = -6558111861786403812445645241 5/1020034229734207253942870967 2
a[9,5] = -1333730829115754792732984060 95/8407082691618982195537383070 4
a[9,6] = 6225156836540393863837014637589522674477368410502819501376 93/
3289942188601405845401861424555681176690770946533324320857 60
a[9,7] = 4616918867155144139901376312 5/2343692704156841275628930358208
a[9,8] = 1888086786587759749309187 5/3469664148196911643475533504

a[10,1] = -7430981552872219612752603 7/5142719003775206577733464 0
a[10,2] = 0
a[10,3] = 913722369253875/113761793498234
a[10,4] = -4406582271592926020603968 90/5810999688168409354523876 7
a[10,5] = 3729088829393576334181438 0/1041174669636029596191423 9
a[10,6] = -6451528881135810657803603921051793104524965326847/2647354251218048148981310421313673204514879109 60
a[10,7] = 4737574861900868386818925 00/5563211008029426393604598 87
a[10,8] = 0
a[10,9] = 0

b[1] = 9420080774669597/198627609019792680
b[2] = 0
b[3] = 0
b[4] = 18658605936510000/72821569629535727
b[5] = 29695087517503 0000/1101802245630054969
b[6] =
188752769802742126868248355661431511898505538963300 09/1487809471396097061043941575966483579945755770362244 40
b[7] = 18663850606812500/74993696164706319

```
b[8] = 179884749312500/58508928482581269
b[9] = 349315176247648/7273791403140339
b[10] = 0

bh[1] = 7362904929137/155056681514280
bh[2] = 0
bh[3] = 0
bh[4] = 7505178129270000/29317774785916981
bh[5] = 1851744839320000/6843492208882329
bh[6] = 750882778189818437512810407839645051128089/600436329571553627073578999231917632220924 0
bh[7] = 27902602073000000/110704980052661709
bh[8] = 0
bh[9] = 0
bh[10] = 331667036438/6791588611709
```

Table 8.4-3 13th degree Runge-Kutta-Verner coefficients

```
c[1] = 0
c[2] = 1/20
c[3] = 341/3200
c[4] = 1023/6400
c[5] = 39/100
c[6] = 93/200
c[7] = 31/200
c[8] = 943/1000
c[9] = 7067558016280/7837150160667
c[10] = 909/1000
c[11] = 47/50
c[12] = 1
c[13] = 1

a[2,1] =  1/20

a[3,1] = -7161/1024000
a[3,2] =  116281/1024000

a[4,1] =  1023/25600
a[4,2] =  0
a[4,3] =  3069/25600

a[5,1] =  4202367/11628100
a[5,2] =  0
a[5,3] = -3899844/2907025
a[5,4] =  3982992/2907025

a[6,1] =  5611/114400
a[6,2] =  0
a[6,3] =  0
a[6,4] =  31744/135025
a[6,5] =  923521/5106400

a[7,1] =  21173/343200
a[7,2] =  0
a[7,3] =  0
a[7,4] =  8602624/76559175
a[7,5] = -26782109/689364000
a[7,6] =  5611/283500

a[8,1] = -1221101821869329/690812928000000
a[8,2] =  0
a[8,3] =  0
a[8,4] = -125/2
a[8,5] = -1024030607959889/168929280000000
a[8,6] =  1501408353528689/265697280000000
a[8,7] =  6070139212132283/92502016000000

a[9,1] = -147251426448621580388138470887726424634604443330709 4\
2078290519780445318011330571 55/1246894801620032001157059621643986024803301558 39\
3487900440453636168046069686436608
a[9,2] =  0
a[9,3] =  0
a[9,4] = -5172294311085668458375175655246981230039025336933699\
1141383152707723193724692800 00/1246193810048091458972786305712152983652570794 10\
2362529218509367490764871329951 91
a[9,5] = -1207067925846925480797893644173318794948457151612046\
```

```
996653451429640689165261497037 5/272203115476165722171047818453110069949728408 50\
483890150850769616734461403986280 96
a[9,6] =  780125155843893641323090552530431036567795592568497 18\
270146067480312677011148162 5/183110425412731972197889874507158786859226102980 86\
185950524144307362914310080537 6
a[9,7] =  664113122959911642134782135839106469928140328160577 03\
535715534039295000949251187 5/151784655985862481363330231072953491752797651500 89\
078301139943253016877823170816
a[9,8] =
103328481844520156040568367672866568591240077969706680464460157750000 00/
131270355003603364807383424874072791453797202863895016524958273367939 3783

a[10,1] = -290555733603374150885386184422310364413140605 11/
226747598910895776913279626023705976320000000 00
a[10,2] =  0
a[10,3] =  0
a[10,4] = -204627495245910491054033652390 69/
454251913499893469596231268750
a[10,5] = -180269259803172281163724663224981 097/
381009225582568710865798328320000 00
a[10,6] =  211276702141728028701282869920039408106552214 89/
467947387799789290614582269797670863367372800 0
a[10,7] =  318607235173649312405151265849660869927653414425 413/
671471671555896530313293807293546542391091200 0000
a[10,8] =  212083202434519082281842245535894/
200224260447756725638228653711738 79
a[10,9] = -\
269840492940084251872116648508712979856226984822951779370341395122671458 3/
469545674913934315077000442080871141884676035902717550325616728175875000000

a[11,1] = -\
234265984581408683695120714006560917907383847624294391 7/
135848096135105677702223140013915876085753216279552000 0
a[11,2] =  0
a[11,3] =  0
a[11,4] = -996286030132538159613930889652/
163530688859961649054643256 75
a[11,5] = -260530859592565341525880893638 41/
437755280456568306101129994240 0
a[11,6] =
209808223450967602922240867949781053126445339256349335 39/
377588999200755080387872783911549464197221296217415680 0
a[11,7] =
890722993756379186418929622095833835264322635782294899/
139212420013951126575019419555940138228301198037647 36
a[11,8] =  161021426143124178389075121929246710833125/
109972077221310346506670413643464228943714 43
a[11,9] =  300760669768102517834232497565452434946672266195876 4\
963718742623926848522439253598648849625 13/465544333750134645558506533660450560 3\
760824779615521285751892810315680492364106674524398280000
a[11,10] = -311552374371117306659232068 75/
392862141594230515010338956291

a[12,1] = -\
286655699182566397177829532910103388753491278772403436 3/
868226711619262703011213925016143612030669233795338240
a[12,2] =  0
a[12,3] =  0
a[12,4] = -169570887141714686763870543589547540 00/
143690415119654683326368228101570221
a[12,5] = -458349397448457291294931467335603354057 5/
451957703655250747157313034270335135744
a[12,6] =
234630538855340425865625847344618441915474017251994957 5/
256726716407895402892744978301151486254183185289662464
a[12,7] =
165712155931984680217128369091361069858625657348480866 2625/
134314804112551464772591551049560935053616444320881090 56
a[12,8] =  345685379554677052215495825476969226377187500/
747711674369300772216672031795513475463620 89
a[12,9] = -320589096271707254279143431215272753400810277402321\
024057136157075724905616701523016035208704867454219601 1/947569549683965814783 01\
512445127360498465774712725761537244920597319265730601723910349107473832403325 9\
120
a[12,10] =  402795458327062334331004385884589332109375 00/
889646084279948284691697212637733894721510 1
```

```
a[12,11] = -612293360107076959161309399399335887725 0/
1050517001510235513198246721302027675953

a[13,1] = -6186759055354825006728008593445384103586601538996 37
/2035442821182140471001194753406676848742921023897 60
a[13,2] =  0
a[13,3] =  0
a[13,4] = -4411194916804718600478400319122931000/
4037305390246996745076149126963301 9
a[13,5] = -1673471140944929253453942253172852022 5/
1801243715290088669307203927210237952
a[13,6] =
1351375197570546790980421841527496777612547518656305 25/
1602958779448628959777132636191189511270371659398348 8
a[13,7] =
3893756836740987601254855190349219613792971043158487 5/
3409564540901916060995487980014693069747584431472 64
a[13,8] = -6748865855011993037732355335815350667265625/
7002880395717424621213565406715087764770357
a[13,9] = -1756005520307450928195422767042525091954178296002 78\
8308926563193523662404739779789732685671/3487678145784699836056880980461864809 0\
4607278021030540735333862087061574934154942830062320
a[13,10] =  5338102458923561108401389767418162929687 5/
8959357584795694524874969598508592944141
a[13,11] =  0
a[13,12] =  0

b[1] =  4490186773775461685197 3/1014046409980231013380680
b[2] =  0
b[3] =  0
b[4] =  0
b[5] =  0
b[6] =  7916386751916152796481000 00/2235604725089973126411512319
b[7] =  3847749490868980348119500000/155170450621382716181412375 17
b[8] = -1373451243239774147656250000 0/875132892924995907746928783
b[9] =  1227476547031319687842881203774063505031923427600698639829444\
3554969616342274215316330684448207141/4893451474937155176503858341435109348888 2\
9280686609654482896526796523353052166757299452852166040
b[10] = -9798363684577739445312500000/3087229863414560318226306 99
b[11] =  2820355431831908400687 50/1229540762987304042599 1
b[12] = -3068142729369769367 53/1299331183183744997286
b[13] =  0

bh[1] =  1083540173940701940657 7/2445218293569351379783 20
bh[2] =  0
bh[3] =  0
bh[4] =  0
bh[5] =  0
bh[6] =  1390818977832189549137 5000/3922113552789426537564 0567
bh[7] =  7348794752702724348762 5000/2965040457733427697739 9443
bh[8] =  6829314064125764960937 5000/1535320864780694574994 6119
bh[9] =  2206064794899667861101771137997457886052201820894972155944 85\
6020333843762602214277638 1/1111542009262325874512959185795727215759010577565736\
0796413766213815772366809295586 40
bh[10] = -5479712294956424582031 25000/2323721402570099164256360 1
bh[11] =  0
bh[12] =  0
bh[13] = -2873545687097896418 9/7978349370426504369 3
```

Table 16th degree Runge-Kutta-Verner coefficients

```
c[1] = 0
c[2] = 1731/50000
c[3] = 7630049/53810000-983539/53810000*6^(1/2)
c[4] = 22890147/107620000-2950617/107620000*6^(1/2)
c[5] = 561/1000
c[6] = 387/1000-129/2000*6^(1/2)
c[7] = 387/1000+129/2000*6^(1/2)
c[8] = 129/200
c[9] = 387/800
c[10] = 6757/100000
c[11] = 1/4
c[12] = 1427971650951258372/2166662646162554701
c[13] = 4103/5000
c[14] = 2253/2500
```

```
c[15] = 1
c[16] = 1

a[2,1] =  1731/50000

a[3,1] =  -177968356965557/1002427673820000+14180534491313/
250606918455000*6^(1/2)
a[3,2] =  64021741529527/200485534764000-7504450763411/
100242767382000*6^(1/2)

a[4,1] =  22890147/430480000-2950617/430480000*6^(1/2)
a[4,2] =  0
a[4,3] =  68670441/430480000-8851851/430480000*6^(1/2)

a[5,1] =  592203994261020339/513126355505556250+
730386990293623641/2052505422022225000*6^(1/2)
a[5,2] =  0
a[5,3] =  -871215388418279490 3/2052505422022225000-\
2843421359195851533/2052505422022225000*6^(1/2)
a[5,4] =  1873698362223295443/513126355505556250+
528258592225556973/513126355505556250*6^(1/2)

a[6,1] =  11380823631/157617812000-339148869/39404453000*6^(1/2
)
a[6,2] =  0
a[6,3] =  0
a[6,4] =  -2355345717024309/58864341808507450*6^(1/2)+
161932328870918 31/58864341808507450
a[6,5] =  165912282616977/4179075230308000-33181894472511/
2089537615154000*6^(1/2)

a[7,1] =  26523528363/231790900000+863255358/123138915625*6^(1/
2)
a[7,2] =  0
a[7,3] =  0
a[7,4] =  -3820874817801648481778 7/842517966262441068418750-\
8611878855628236982280 7/842517966262441068418750*6^(1/2)
a[7,5] =  92362336407446913/290322814529044000-\
232039320950012997/2467743923496874000*6^(1/2)
a[7,6] =  -362925891/1690350537500+857800423623/3380701075000*6
^(1/2)

a[8,1] =  43/600
a[8,2] =  0
a[8,3] =  0
a[8,4] =  0
a[8,5] =  0
a[8,6] =  43/150+43/2400*6^(1/2)
a[8,7] =  -43/2400*6^(1/2)+43/150

a[9,1] =  7353/102400
a[9,2] =  0
a[9,3] =  0
a[9,4] =  0
a[9,5] =  0
a[9,6] =  22833/102400+8901/204800*6^(1/2)
a[9,7] =  -8901/204800*6^(1/2)+22833/102400
a[9,8] =  -3483/102400

a[10,1] =  3767087424722149887008 53/778845602812500000000000,
`:`
a[10,2] =  0
a[10,3] =  0
a[10,4] =  0
a[10,5] =  0
a[10,6] =  1879146667539568401952 79/2596152009375000000000000-\
2104408465562906932689 11/1557691205625000000000000*6^(1/2)
a[10,7] =  2104408465562906932689 11/1557691205625000000000000*
6^(1/2)+1879146667539568401952 79/2596152009375000000000000
a[10,8] =  -1855266722189674422664 7/8653840031250000000000000,
`:`
a[10,9] =  -3167799860072183913409/3042365635986328125 0000

a[11,1] =  -426968570497/54394415898750-92754382349/
12087647977500*6^(1/2)
a[11,2] =  0
```

```
a[11,3] =  0
a[11,4] =  0
a[11,5] =  0
a[11,6] =  1/30
a[11,7] =  -2865012129681958/114898584332330625-\
12962517687655099/229797168664661250*6^(1/2)
a[11,8] =  4389715333607/309890657317500+92754382349/
11477431752500*6^(1/2)
a[11,9] =  371017529396/9306344041875*6^(1/2)+4990058173976/
83757096376875
a[11,10] =  109952352459599312500 0/625766790986975601889 1+
100957348037989687500/625766790986975601889 1*6^(1/2)

a[12,1] =  1838203110479840386993853900915465658752149857359559\
5063164077882800315372787284683238439478955141517997198007108623761931447163756
/1397425694449972434491896099389093361416102532297045004793268899809500852862 08\
2123960473460811129176944470618749780786917955084132937 5+4078857781851586092107\
9389251758259530589647075646761263679625961149140826089641344688345089135162291\
4818800693274034252252905536/2808492638860122607362409616917500295697019157645 5\
1106332267651411613722940986932751171812393853121981375088465359331278371679268\
75*6^(1/2)
a[12,2] =  0
a[12,3] =  0
a[12,4] =  0
a[12,5] =  0
a[12,6] =  -3338813117898494119715734728681282814382022107217 23\
1232517421453677345828875773955477782287601740687580861343899520155634039 04/227\
0872004608103037127689848604039623086639035441372934050180593816493796129405349\
9141489814607142022329887277387784945577276 35+4819272892477768171373308666720 68\
9121421091953625792970278044071549950640195056472955523769829034800621890424847\
009130000000/2316289444700265097870243645576120415548371816150200392731184205 69\
2823672051993456912431961089928486277648502293554064448882187 7*6^(1/2)
a[12,7] =  1698450855653613368055560092963943745276369523793889\
6102606662872515552183276208687563236699647756792865753591219139615556676545782\
6139904/1593979913718173260623679425966077497797978183146596135797795272107637 6\
2009861408743487373293393731509448918731447404529311920099960958687 5*6^(1/2)-13\
6666607496463622270135608863772076443625468798139480390426740993024803946981763\
2093483647161087213128226198457261516936675984376996944 16/3719286465342404274 78\
8585327254180828195282427342057650194855634917821113563432870681372043512520401\
887141437067106105683944802332422369375
a[12,8] =  5610987899273278525411960528081442902198567594809764\
3797561956736732657005510768128839255833702537657025532355947644271736376737662\
08/9288159819814403301827880474062633413542335679163959810935886777036160923284\
6012626732332450844264293840456574956036349633197336361875-5587476413495323413 8\
4649167832304925076570507885572072105200355632180011316296456776552672453906332\
7600257543743479921263738432/3653030893622016645164135969252861614944735753371 1\
5296250511752859728108868696929614024803255122785403232359817965288739565550625
*6^(1/2)
a[12,9] =  5459853981808361523356614860220324489669695891073433\
9754065270985433507945162707737759469214674480807272210648148477499238783276259\
328/3012479190922988526348868751299593107946629320141844998271450758516372986 98\
3120740305674792395020116934474230264160407944799340240581 25-6526172450962537 74\
7372702280281321524894343532103481802188740153783862532174342615150135214261625\
9666371008110923845480360464885 76/8649093284303728183602838792132050268657965 31\
7662489228456648746817034128576286937426571324705771222895418404433420637223081\
6544375*6^(1/2)
a[12,10] =  -8108825145085088104344721048166325225173729495689 3\
6469642672016111201241422775232896972065898731565417987376035772523573400039944\
0000000/2655582966610640210612741027567977548105720815295466925220401896406182 0\
6693081506133206500662983001047298787619827411375675716583283801757*6^(1/2)+939\
1667348404584010955422210328707125006120661611061908889750805619418785820948002\
4558903609392219121905247310870706451074869134577600000000/5815726696877302061 24\
1902850373870830351528585497072566232680153129538726578484984317222364519327722\
9358434488742203091272981931739152584783
a[12,11] =  1234617126598879151772713393966068608104790287778 69\
3480148704506062609140195602856612882124981284004760156959603419 52/281629106670\
3206747542452093588407037042351473078388967410755112208260568290472056143249782\
532261762750789227161324 61

a[13,1] =  -\
5604277267532204213922762997804258633063362270605336394676614441693363 1/
5880854077232319052559012261322343050735211853455734266601562500000000+
2814045797346992321414555246044877241590249725 27/
1478009944832743180452316204077188415527343750000*6^(1/2)
a[13,2] =  0
a[13,3] =  0
```

```
a[13,4] =  0
a[13,5] =  0
a[13,6] =  -10271639002297503565612382379472253326756215 17/
17926189443113266407874769829286743164062 5000-\
27452923916412025253731039793368135133723 21/
11702216468464340311060649744558385937500000*6^(1/2)
a[13,7] =  -\
15722999985374822730516577336442692528237807223833293 0121/
36699907367985458573273204094330716033963413238525390625+
57576064428027950953189860673178379041842786506645902 52101/
35231911073266040230342275930557487392604876708984375 00000*6^(1/2)
a[13,8] =  -\
93114481685939341460159650199040136021338029433258183466227812 85907057/
42559708490101242171931354496687399854013133630055761593627929 68750000-\
84421373920409769642436657381346317247707491758 1/
42101883599465783369768681649661630249023437500 00*6^(1/2)
a[13,9] =
88577423385667259022295186769532781645734013039163915307052133548 5617578/
30109854138029501101546924846546529011250565614375779993463516235 3515625-\
28140457973469923214145552460448772415902497252 7/
28448191636473798322140232250483030319213867187 5*6^(1/2)
a[13,10] =
31547911672978015395641212405219968509774423938663902378735910795 9254802182/
13448185050650584801258784221551557438021254320089493232912847115 4748828125-\
29403964536478722766460687765922922297376519379346 23/
73454650587819837107958374295307847772452865207031 25*6^(1/2)
a[13,11] =
22509961634065453786165320390188465862176315994538225 41/
38249130379709599356330414820427563643350402832031 2500
a[13,12] =  26893409573076918532949023883344540039593781469575 2\
98662335292519863593923360441517089497209588097479705143662934584242721740244 93
/95951638601957880850056911478087170846689475228048283510540802781519489531905 5\
44384278222710212049396080564957556179687500000000

a[14,1] =  47342003848024391498707976847688893013083074441159 77\
94657198636250516689398877026303 19/44802546873926050730401222636656855760802419\
99385206026461532080148539245605468750 0000-\
86636953098707799112556240282909218710049320960 1/
33255223758736721560177114591736739349365234375 00*6^(1/2)
a[14,2] =  0
a[14,3] =  0
a[14,4] =  0
a[14,5] =  0
a[14,6] =  87177932180780244746331003531823876287852715 7/
13444642082334949805906077371965057373046875 0+
10764126848099939608184897527184985799481 8/
10970827939185319041619359135523486816406 25*6^(1/2)
a[14,7] =
49610378635186229280003480511419070548480074351335411 7014/
11009972210395637571981961228299214810189023971557617 1875-\
13299384126061974857693125993903073511915408915993748 31099/
66059833262373825431891767369752888611341438293457031 250*6^(1/2)
a[14,8] =  40774077277747636354598451708891165494123131383777 23\
52295386119893921751932859942664 71/152642905462481621010589859415880795182567 41\
25537703173635794612571352470397949218750 0+
12376707585529685587508034326129888387149902994 3/
45109160999427625039037873196066032409667968750 0*6^(1/2)
a[14,9] =  -10522038608500556459828649038302068473735749030796 3\
72764961618751973793724796364606986664/38994174250054222540345740003973828622 35\
89282965337583519734091827155605550765991210937 5+
34654781239483119645022496113163687484019728384 04/
25603372472826418489926209025434727287292480468 75*6^(1/2)
a[14,10] =  -27843764471262693189365201135620670490328475323282\
82021947485162169389576952709433468710898 4/122570410662851642220025943006055939\
29434139193022166317802121412999357024704596261133984375+
57477430027199859868387311410547201669924149505529 2/
10493521512545691015422624899329692538921837886718 75*6^(1/2)
a[14,11] =  -\
34241134351848245624232809437676889009431930503529853032576417589898516/
56133478243586519811009850090242810076032300624399426827131652832031 25
a[14,12] =  -34320443758939323781023685680522865010338509105169\
99202088532705211633432793920547702809961532438008401883737341854688972639605 33\
46001639386102688557057427640726 09/114317410634168226097164769041056729214392 61\
98650927778920823267461111371275907599801714870165813394147519068210931766844 49\
49946165802584355181814345751953125 00000
a[14,13] =  47469308760239193350794516127267176492182641999 84/
```

```
185920655384070497552001443881340893464327555948 77

a[15,1] =  -251883292492588254437485270381424098799230121337389\
853132654309322802508557086 01/113706413255744693120569618740772985508276423087 7\
464731699571703634755806428625 0+
123427305898186017017959259853550863134308253554988 1956/
210563377146962874451839064296855214406989884589580 8125*6^(1/2)
a[15,2] =  0
a[15,3] =  0
a[15,4] =  0
a[15,5] =  0
a[15,6] =  -548211421196850555624772162054286139499054303960 88/
395943983700946128908558774674809794739310127809 5-\
151127675382598285607289146950447125666497592500 0/
403862863374965051486729950168305990634096336565 69*6^(1/2)
a[15,7] =  -\
609224242740615999186035240493906573054312626351975404056979 52/
648486174748903216977458462475995314853156403241746190951687 5+
845585757516359787331099618939842387869295504626153756993416 16/
194545852424670965093237538742798594455946920972523857285506 25*6^(1/2)
a[15,8] =  -116118147575045169733222875835719955334334798191459\
879782123534889390467935109772/881062690195483524567227513129587089250371395751\
217068145330081498841764249 3125-\
176324722711694310025656085505072661620440362221411708/
285619406719829107485771207042040133465420149964555625*6^(1/2)
a[15,9] =  177694487225138983422768374906650972869276072470733 3\
561856698714346729490018303321 6/255121700813788961505634214608456186712248516 35\
966192837199577424187510295063568 75-\
197483689437097627228734815765681381014893205687981112 96/
648455426232225907128654593599712913511181368717565062 5*6^(1/2)
a[15,10] =  976592661391240748181932648019295477816599265437863\
815101909541842185707462150338239935300000 00/185600766544697062059634829087870 5\
685081230820560312732685536096172760824279655110118208003359 9-\
852970846117821224749111313630789000588880252246079137450000 00/
692106594502013938431667467229540363263383556499153838517339 11*6^(1/2)
a[15,11] =
473389749049752963256114649231353822492912259509649519870869750525/
354124408823603417997988424283654229412165081213226224792608462 91
a[15,12] =  333514392451584382480734940567841440978729127734159\
045364007283876903345639683941147024141088075051581063851164687328534582028999 6\
674848871853154570655914289590314484876463 7/231661102532728742771480201132225 28\
860907939049899006215923656276490975781021635721905022324254906067733123106655 9\
342498274574429937128559858829860608854337674205464481896 6
a[15,13] =  -\
387149926569584133897432527260168975992839116829452556366435546875000 00/
485404949269715874992945893825722120361691354298779017023475213004217 67
a[15,14] =
148002502009403237171246161756412612351192957957688147178039550781 25/
335655771251418777602873805886324212234331940781569482984884711604 89

a[16,1] =
230578569608639756108085818693989717364564133108504131394438984998658410128 7/
617508244345282265819087370078275122671246164669900462139876057008239440000-\
854046233055897126321659052339741831376078991407 19/
124822287169084833758410283469525117460541643292500*6^(1/2)
a[16,2] =  0
a[16,3] =  0
a[16,4] =  0
a[16,5] =  0
a[16,6] =  102903996961580448264190625267026062654799259083/
504639808489000485748162999967332043881948473 0+
413209254873042193133002720521283745670811281 25/
514732604658780495463126259966678684759587442 46*6^(1/2)
a[16,7] =
627984433498764575067189208439756613999495645980184881444 66/
413255349878257332405826358255371522077705135978014138062 5-\
723088070819329615544257110897167710135714199506573007291 03/
123976604963477199721747907476611456623311540793404241418 75*6^(1/2)
a[16,8] =
179490914212648256439084852292422555322146901975147054495929761465466129337 7/
525964811939942644356016261097529886746796916442754567166339757859786725 00+
122006604722271018045951293191391690196582713058 17/
169315614565599591152077093440565782633977606025 00*6^(1/2)
a[16,9] =  -277524473278010966734284561239473931911566263637147\
730045574702242327047590725 6/
228417153675584029725018045422706955827996328208181619436454383447149337555625+
```

```
3416184932223588505286636209358967325043159656287 6/
96101338378773357469245211954911505447551097205625*6^(1/2)
a[16,10] =  -276805546597690166235309791767274482512922443107 69\
99601534219081906897055608306312500 0/329955777742964896057656138225660684467725\
843879707295534158135405137503652223147143 7+
442655212757989537347967035610017975994476655814173031250 0/
307711373866732070774887719980463674649497700065896798767 7*6^(1/2)
a[16,11] =  -\
292603171929706291053929402159930330736639136252680853622275/
154736228262791611502270768872902624435105509642758581439 64
a[16,12] =  -981571712956910698856930219322099934382493208458 20\
936475960869317546660986625941530952589885163051657947397448735398290696172035 2\
350913668221693302043 1/286476991170934153076146641094402171801937250068596542 93\
102867866950176225328769329439768932779738811385458811343006393940507197909254 7\
998950955940992
a[16,13] =
272949114470983790579914876665078253290605029897140651852416992187 5/
215811588862213947314277581210944780292065614924312730925368695146 9
a[16,14] =  0
a[16,15] =  0

b[1] =  819816036620317341111994371150033 1/
561057579384085860167277847128765528
b[2] =  0
b[3] =  0
b[4] =  0
b[5] =  0
b[6] =  0
b[7] =  0
b[8] =  -455655493073428838813281446213740000000/
116380801115091056124046422583731249786 9
b[9] =  199651636487060080811350757469156147200000000/
863944041905370868683946862057824325165445 99
b[10] =  892311079199814187055669708043437500000000000000000000000/
699979870988335674445594679856445060562597693583175985391
b[11] =  471042739549459067131849138711434 92/
209684639122339601934631113492763467
b[12] =  208450044214045004640105847407967506508321767983703830842263\
512947307311966736473110623309727407347372795031193876271463816786771561360425 2\
413931190748280284408 3/366708498911363730202382253282651002506051447185019263 05\
140966586758054847604681466336103169284755987753542321202462371554120593858149 7\
555398785619767865923896 08
b[13] =  605303728214230650979591128690917968750000000 0/
103899257350518063455290077573775162739725126989
b[14] =  917401104920993498360358406096725463867187500/
672424981591134665331579073745360738298955146 3
b[15] =  258544955766526895137169959649395 7/
845743451607641401632086060484275 31
b[16] =  0

bh[1] =  552562031208180939317806684253/
276696542577346678585233440414 64
bh[2] =  0
bh[3] =  0
bh[4] =  0
bh[5] =  0
bh[6] =  0
bh[7] =  0
bh[8] =  221223388631423597589898601690000000/
100946136798587090054685074667127461
bh[9] =  101835408791305297984657812561920000000/
114976383320074375997650665024131210013 9
bh[10] =  131372030907763001445323984375000000000000000000/
115182019232155109891264665311074370373957191171 33
bh[11] =  483361123270144050450808615172 8/
190813212414541452301966615245 03
bh[12] =  -212966237458232464810691979570337364535311827306674223072 4\
172731025813964712473647144010599206669825382719359113196238857709025512340589 9\
57/103554373927236708088519054620109721889126872811820733259259598755485188297 2\
292670881794178380097716583123063485287435793657425889233080568
bh[13] =  108476159175364085584435806396484375000000 0/
318289548603124907193854969132050248873342 3
bh[14] =  0
bh[15] =  0
bh[16] =  183919007106064988712789510078 4/
380451395235106343514208754153 97
```

Runge-Kutta-Fehlberg methods

Similar to Verner methods, Fehlberg method is also given a pair of solutions. Later, it will be used in variable step calculations, but it can also utilised as single step solution of a good accuracy. Runge-Kutta-Fehlberg Buthcher tableu is given as:

0	0	0	0	0	0	0
$1/4$	$1/4$	0	0	0	0	0
$3/8$	$3/32$	$9/32$	0	0	0	0
$12/13$	$1932/2197$	$-7200/2197$	$7296/2197$	0	0	0
1	$439/2197$	-8	$3680/513$	$-845/4104$	0	0
$1/2$	$-8/27$	2	$-3544/2565$	$1859/4104$	$-11/40$	0
	$25/216$	0	$1408/2565$	$2197/4104$	$-1/5$	0
	$16/135$	0	$6656/12825$	$28561/56430$	$-9/50$	$2/55$

Table 8.4-5 The same coefficients in table format :

```
c[1] =  0
c[2] =  1/4
c[3] =  3/8
c[4] =  12/13
c[5] =  1
c[6] = 1/2


a[2,1] =  1/4

a[3,1] =  3/32
a[3,2] =  9/32

a[4,1] =  1932/2197
a[4,2] =  -7200/2197
a[4,3] =  7296/2197

a[5,1] =  439/216
a[5,2] =  -8
a[5,3] =  3680/513
a[5,4] =  -845/4104

a[6,1] =  -8/27
a[6,2] =  2
a[6,3] =  -3544/2565
a[6,4] =  1859/4104
a[6,5] =  -11/40

b[1] =  25/216
b[2] =  0
b[3] =  1408/2565
b[4] =  2197/4104
b[5] =  -1/5
b[5] =  0

bh[1] =  16/135
bh[2] =  0
bh[3] =  6656/12825
bh[4] =  28561/56430
bh[5] =  -9/50
bh[5] =  2/55
```

6th degree Runge-Kutta- Cash-Karp coefficients are as below:

Runge-Kutta-Cash-Karp method(RKCK)

Runge-Kutta-Cash-Karp is a modified form of the Runge-Kutta_fehlberg method (different coefficients). Similar to RKF, this method also have a pair of solutions

0	0	0	0	0	0	0
$1/5$	$1/5$	0	0	0	0	0
$3/10$	$3/40$	$9/40$	0	0	0	0
$3/5$	$3/10$	$-9/10$	$6/5$	0	0	0
1	$-11/54$	$5/2$	$-70/27$	$35/27$	0	0
$7/8$	$1631/55296$	$175/512$	$575/13824$	$44275/110592$	$253/4096$	0
	$37/378$	0	$250/621$	$125/594$	0	$512/1771$
	$2825/27648$	0	$18575/48384$	$13525/55296$	$277/14336$	$1/4$

The same coefficients is also given in table form below

Table 6th degree Runge-Cash-Karp coefficients

```
c[1] =  0
c[2] =  0.2
c[3] =  0.3
c[4] =  0.6
c[5] =  1
c[6] = 7/8


a[2,1] =  0.2

a[3,1] =  3/40
a[3,2] =  9/40

a[4,1] =  0.3
a[4,2] =  -9/10
a[4,3] =  6/5

a[5,1] =  -11/54
a[5,2] =  5/2
a[5,3] =  -70/27
a[5,4] =  35/27

a[6,1] =  1631/55296
a[6,2] =  175/512
a[6,3] =  575/13824
a[6,4] =  44275/110592
a[6,5] =  253/4096


b[1] =  37/378
b[2] =  250/621
b[3] =  125/591
b[4] =  512/1771
b[5] =  0

bh[1] =  2825/27648
bh[2] =  18575/48384
bh[3] =  13525/55296
bh[4] =  277/14336
bh[5] =  1/4
```

Table -7 7th degree Runge-Kutta-Dormant-Prince coefficients

```
c[1]=0.0e0;
c[2]=0.2e0;
c[3]=0.3e0;
c[4]=0.8e0;
c[5]=8.0e0/9.0e0;
c[6]=1;
c[7]=1;
a[2][1]=0.2e0;
a[3][1]=3.0e0/40.0e0;
a[3][2]=9.0e0/40.0e0;
a[4][1]=44.0e0/45.0e0;
a[4][2]=-56.0e0/15.0e0;
a[4][3]=32.0e0/9.0e0;
a[5][1]=19372.0e0/6561.0e0;
a[5][2]=-25360.0e0/2187.0e0;
a[5][3]=64448.0e0/6561.0e0;
```

```
    a[5][4]=-212.0e0/729.0e0;
    a[6][1]=9017.0e0/3168.0e0;
    a[6][2]=-355.0e0/33.0e0;
    a[6][3]=46732.0e0/5247.0e0;
    a[6][4]=49.0e0/176.0e0;
    a[6][5]=-5103.0e0/18656.0e0;
    a[7][1]=35.0e0/384.0e0;
    a[7][3]=500.0e0/1113.0e0;
    a[7][4]=125.0e0/192.0e0;
    a[7][5]=-2187.0e0/6784.0e0;
    a[7][6]=11.0e0/84.0e0;

    b[1]=71.0e0/57600.0e0;
    b[3]=-71.0e0/16695.0e0;
    b[4]=71.0e0/1920.0e0;
    b[5]=-17253.0e0/339200.0e0;
    b[6]=22.0e0/525.0e0;
    b[7]=0.0e0;
    bh[1]=-12715105075.0e0/11282082432.0e0;
    bh[3]=87487479700.0e0/32700410799.0e0;
    bh[4]=-10690763975.0e0/1880347072.0e0;
    bh[5]=701980252875.0e0/199316789632.0e0;
    bh[6]=-1453857185.0e0/822651844.0e0;
    bh[7]=69997945.0e0/29380423.0e0;
```

Program of Runge-Kutta is given in general form. For coefficients of different formulations a Butcher class is created and then a general Runge_Kutta method reads coefficients from this class.
Butcher table and general Runge_Kutta method based on Butcher table and example case is is given below

Program Butcher table class (Python code)

```
# -*- coding: utf-8 -*-
"""
Created on Fri Aug 31 10:22:54 2018

@author: Mustafa Turhan Çoban
"""
from math import *

class  Buthcher():
  def set(self,ai,bi,ci):
      self.a=ai;
      self.b=bi;
      self.c=ci;
  def __init__(self,namei):
    self.name = namei;
    if str(self.name) ==("RK2"):
        a=[[0,0],[1,0]];
        b=[[0.5,0.5]];
        c=[0,1];
        self.set(a,b,c);
    elif str(self.name) == ("Raltston"):
        a=[[0,0],[3.0/4.0,0]];
        b=[[1.0/3.0,2.0/3.0]];
        c=[1.0/6.0,4.0/6.0,1.0/6.0];
        self.set(a,b,c);
    elif str(self.name) == ("RK3"):
        a=[[0,0,0],[0.5,0,0],[-1,2,0]];
        b=[[0,0.5,1]];
        c=[1.0/6.0,4.0/6.0,1.0/6.0];
        self.set(a,b,c);
    elif str(self.name) == ("RK4"):
        a=[[0,0,0,0],[0.5,0,0,0],[0,0.5,0,0],[0,0,1,0]];
        b=[[1.0/6.0,2.0/6.0,2.0/6.0,1.0/6.0]];
        c=[0,0.5,0.5,1];
        self.set(a,b,c);
    elif str(self.name) == ("RK6"):
        a=[[0,0,0,0,0,0],[0.25,0,0,0,0,0],[0.125,0.125,0,0,0,0],[0,-0.5,1,0,0,0],[3.0/16.0,0,0,9.0/16.0,0,0],[-3.0/7.0,2.0/7.0,12.0/7.0,-
12.0/7.0,8.0/7.0,0]];
        b=[[7.0/90.0,0,32.0/90.0,12.0/90.0,32.0/90.0,7.0/90.0]];
        c=[0,0.25,0.25,0.5,0.75,1.0];
        self.set(a,b,c);
```

```
elif str(self.name) == ("RKV9"):
    #Runge-Kutta-Verner 9th degree formula
    n=10;
    c=[0.0 for i in range(n)];
    a = [[0 for x in range(n)] for y in range(n)];
    bh=[0.0 for i in range(n)];
    bh1=[0.0 for i in range(n)];
    c[0]=0;
    c[1] =  0.;
    c[2] =  .6e-1;
    c[3] =  .95933333333333333333333333333333333333333e-1;
    c[4] =  .1439;
    c[5] =  .4973;
    c[6] =  .9725;
    c[7] =  .9995;
    c[8] =  1.;
    c[9] =  1.;
    a[2][1] =  .6e-1;
    a[3][1] =  .192399629629629629629629629629629629629e-1;
    a[3][2] =  .766933703703703703703703703703703703703e-1;
    a[4][1] =  .35975e-1;
    a[4][2] =  0.;
    a[4][3] =  .107925;
    a[5][1] =  1.31868341523314826091974727643173561286 1;
    a[5][2] =  0.;
    a[5][3] = -5.04205806362856222542776163471563769334 4;
    a[5][4] =  4.22067464839541396450801435828390208048 3;
    a[6][1] = -41.87259166432751461803757780644346812905;
    a[6][2] =  0.;
    a[6][3] =  159.43256216313749177003656690703468304 53;
    a[6][4] = -122.11921356501003092025162033892421406 63;
    a[6][5] =  5.53174306620005376825263123833299915007 6;
    a[7][1] = -54.43015693531650433250642051294142461271;
    a[7][2] =  0.;
    a[7][3] =  207.06725136501846442736571738665098359 87;
    a[7][4] = -158.61081378458999918287424243650585994 69;
    a[7][5] =  6.99181658595024232199259728079179390709 6;
    a[7][6] = -.18597231062203233977651717995492946236 92e-1;
    a[8][1] = -54.66374178728197680241215648050386959351;
    a[8][2] =  0.;
    a[8][3] =  207.95280625538937345158248166998342442 38;
    a[8][4] = -159.28895747449950715089598058714266542 16;
    a[8][5] =  7.01874374079694443469817076096425249081 7;
    a[8][6] = -.18338785905045723064727820051417382683 61e-1;
    a[8][7] = -.51194849978820990778754324972451683958 40e-3;
    a[9][1] =  .34389578683570360092788201247283223865 20e-1;
    a[9][2] =  0.;
    a[9][3] =  0.;
    a[9][4] =  .25826245556335034046595580985861208587 67;
    a[9][5] =  .42093711896735371506425515140698019670 32;
    a[9][6] =  4.40539646966931017014883681619709566489 1;
    a[9][7] = -176.48311902429865761517409424990021250 29;
    a[9][8] =  172.36413340141507302940225827119024133 15;
    bh1[1] =  .34389578683570360092788201247283223865 20e-1;
    bh1[2] =  0.;
    bh1[3] =  0.;
    bh1[4] =  .25826245556335034046595580985861208587 67;
    bh1[5] =  .42093711896735371506425515140698019670 32;
    bh1[6] =  4.40539646966931017014883681619709566489 1;
    bh1[7] = -176.48311902429865761517409424990021250 29;
    bh1[8] =  172.36413340141507302940225827119024133 15;
    bh1[9] =  0.;
    bh[ 1] =  .49099676483824897309068549279712258364 79e-1;
    bh[ 2] =  0.;
    bh[ 3] =  0.;
    bh[ 4] =  .22511122295165241534013953205398753294 85;
    bh[ 5] =  .46946822530295620394319485250473874125 53;
    bh[ 6] =  .80657922499888677076341618089952179814 43;
    bh[ 7] =  0.;
    bh[ 8] = -.60711948917779597976729514652562171224 88;
    bh[ 9] =  .56861139440475692411476031787661381535 94e-1;
    b=[bh1,bh];
    self.set(a,b,c);
elif str(self.name) == ("RKCK45"):
    #Runge-Kutta-Fehlberg, Cash-Karp, Dormant-Price, Verner
    n=7;
```

```
        dydx=1.0;
        c=[0.0 for i in range(n)];
        a = [[0 for x in range(n)] for y in range(n)];
        bh1=[0.0 for i in range(n)];
        bh2=[0.0 for i in range(n)];
        c[2] =  0.2;
        c[3] =  0.3;
        c[4] =  0.6;
        c[5] =  1;
        c[6] = 7.0/8.0;
        a[1][1] = 0;
        a[2][1] =  0.2;
        a[3][1] =  3.0/40.0;
        a[3][2] =  9.0/40.0;
        a[4][1] =  0.3;
        a[4][2] =  -9.0/10.0;
        a[4][3] =  6.0/5.0;
        a[5][1] =  -11.0/54.0;
        a[5][2] =  0.25;
        a[5][3] =  -70.0/27.0;
        a[5][4] =  35.0/27.0;
        bh1[1] =  37.0/378.0;
        bh1[2] =  250.0/621.0;
        bh1[3] =  125.0/591.0;
        bh1[4] =  512.0/1771.0;
        bh1[5] =  0.0;
        bh2[1] =  2825.0/27648.0;
        bh2[2] =  18575.0/48384.0;
        bh2[3] =  13525.0/55296.0;
        bh2[4] =  277.0/14336.0;
        bh2[5] =  1.0/4.0;
        b=[bh1,bh2];
        self.set(a,b,c);
    elif str(self.name) == ("RKDP45"):
        #Runge-Kutta Dormant-Prince order 4-5
        n=8;
        dydx=1.0;
        c=[0.0 for i in range(n)];
        a = [[0 for x in range(n)] for y in range(n)];
        bh1=[0.0 for i in range(n)];
        bh2=[0.0 for i in range(n)];
        c[1]=0.0e0;
        c[2]=0.2;
        c[3]=0.3;
        c[4]=0.8;
        c[5]=8.0/9.0;
        c[6]=1;
        c[7]=1;
        a[2][1]=0.2;
        a[3][1]=3.0/40.0;
        a[3][2]=9.0/40.0;
        a[4][1]=44.0/45.0;
        a[4][2]=-56.0/15.0;
        a[4][3]=32.0/9.0;
        a[5][1]=19372.0/6561.0;
        a[5][2]=-25360.0/2187.0;
        a[5][3]=64448.0/6561.0;
        a[5][4]=-212.0/729.0;
        a[6][1]=9017.0/3168.0;
        a[6][2]=-355.0/33.0;
        a[6][3]=46732.0/5247.0;
        a[6][4]=49.0/176.0;
        a[6][5]=-5103.0/18656.0;
        a[7][1]=35.0/384.0;
        a[7][3]=500.0/1113.0;
        a[7][4]=125.0/192.0;
        a[7][5]=-2187.0/6784.0;
        a[7][6]=11.0/84.0;
        bh1[1]=5179.0/57600.0;
        bh1[2]=0;
        bh1[3]=7571.0/16695.0;
        bh1[4]=393.0/640.0;
        bh1[5]=-92097.0/339200.0;
        bh1[6]=187.0/2100.0;
        bh1[7]=1.0/40.0;
        bh2[1]=35.0/384.0;
```

```
        bh2[2]=0;
        bh2[3]=500.0/1113.0;
        bh2[4]=125.0/192.0;
        bh2[5]=-2187.0/6784.0;
        bh2[6]=11.0/84.0;
        bh2[7]=0.0;
        b=[bh1,bh2];
        self.set(a,b,c);
    elif str(self.name) == ("RKV13"):
        #Runge-Kutta-Verner 13th degree
        n=14;
        c=[0.0 for i in range(n)];
        a = [[0 for x in range(n)] for y in range(n)];
        bh=[0.0 for i in range(n)];
        c[0]=0;
        c[1] =  0.;
        c[2] =  .5e-1;
        a[2][1] =  .5e-1;
        c[3] =  .1065625;
        a[3][1] = -.69931640625e-2;
        a[3][2] =  .1135556640625;
        c[4] =  .15984375;
        a[4][1] =  .399609375e-1;
        a[4][2] =  0.;
        a[4][3] =  .1198828125;
        c[5] =  .39;
        a[5][1] =  .36139756280045751240529407211840283451 29;
        a[5][2] =  0.;
        a[5][3] = -1.34152406670049277181998778820271583491 7;
        a[5][4] =  1.37012650390003525941469371608431300040 4;
        c[6] =  .465;
        a[6][1] =  .490472027972027972027972027972027972028 0e-1;
        a[6][2] =  0.;
        a[6][3] =  0.;
        a[6][4] =  .235097204221440473986298833549342714312 2;
        a[6][5] =  .180855592981356728810903963653454488485 0;
        c[7] =  .155;
        a[7][1] =  .616928904428904428904428904428904428904 4e-1;
        a[7][2] =  0.;
        a[7][3] =  0.;
        a[7][4] =  .112365683146402766226255703513001544230 3;
        a[7][5] = -.388504607145136676704904810811124456745 6e-1;
        a[7][6] =  .197918871252204585537918871252204585537 9e-1;
        c[8] =  .943;
        a[8][1] = -1.76763024022232687573559711957214558671 4;
        a[8][2] =  0.;
        a[8][3] =  0.;
        a[8][4] = -62.5;
        a[8][5] = -6.06188937737666910082136145965933199975 8;
        a[8][6] =  5.65082319822276313856129803060084017420 1;
        a[8][7] =  65.62169641937623283799566054863063741227;
        c[9] =  .901802041735856958259707940678372149956 0;
        a[9][1] = -1.18094506655497079982511628262829795788 2;
        a[9][2] =  0.;
        a[9][3] =  0.;
        a[9][4] = -41.50473441114320841606641502701994225874;
        a[9][5] = -4.43443831910372501122516922984610021177 6;
        a[9][6] =  4.26040818858613302481219371074469324076 1;
        a[9][7] =  43.75364022446171584987676829438379303004;
        a[9][8] =  .787142548991231068744647504422630755086 0e-2;
        c[10] =  .909;
        a[10][1] = -1.28140599944148840545951029118205424626 6;
        a[10][2] =  0.;
        a[10][3] =  0.;
        a[10][4] = -45.04713996013986630220754257136007322267;
        a[10][5] = -4.73136206944957647731146426549128281094 3;
        a[10][6] =  4.51496701659380784118585158459724099621 4;
        a[10][7] =  47.44909557172985134869022392235929015114;
        a[10][8] =  .105922829711166113568739395551654287522 8e-1;
        a[10][9] = -.57468422638446162544323184782862962320 21e-2;
        c[11] =  .94;
        a[11][1] = -1.72447013426248519175670981748448186173 1;
        a[11][2] =  0.;
        a[11][3] =  0.;
        a[11][4] = -60.92349008483054016518434619253765246063;
        a[11][5] = -5.95151837622239245520283276706185486829 0;
```

```
        a[11][6] =  5.5565237306984562359797916508435924968 39;
        a[11][7] =  63.9830119803330533683753637863599593928 1;
        a[11][8] =  .1464202825041496159275921391759452676003e-1;
        a[11][9] =  .6460408772358203603621865144977650714892e-1;
        a[11][10] = -.7930323169008878984024452548693373291447e-1;
        c[12] =  1.;
        a[12][1] = -3.3016226677470790163539947897909836255 69;
        a[12][2] =  0.;
        a[12][3] =  0.;
        a[12][4] = -118.01127235975250856669233039578988685 10;
        a[12][5] = -10.14142238845611248642783916034510897595;
        a[12][6] =  9.1393113322320579235440122735568270006 19;
        a[12][7] =  123.37594282840426836848471809865018943 64;
        a[12][8] =  4.6232443788745804748398076250676309247 92;
        a[12][9] = -3.3832777380682019236525550971536811240814;
        a[12][10] =  4.5275921003246181894512653393511290353 25;
        a[12][11] = -5.8284954858116229631930880191629857037 55;
        c[13] =  1.;
        a[13][1] = -3.0395150337663090300401028518212002510 56;
        a[13][2] =  0.;
        a[13][3] =  0.;
        a[13][4] = -109.26086808941762546864441923221646233 52;
        a[13][5] = -9.2906424974002934497176655426568975491 58;
        a[13][6] =  8.4305049817649111421342992538361678034 54;
        a[13][7] =  114.20100103783313135574240410955234274 76;
        a[13][8] = -.9637271342145479358162375658987901652762;
        a[13][9] = -5.0348840888021897911986803361833323231 18;
        a[13][10] =  5.9581308240029231775404021653881720727 94;
        a[13][11] =  0.;
        a[13][12] =  0.;
        bh[1] =  .4431261522908979212486436510209029764893e-1;
        bh[2] =  0.;
        bh[3] =  0.;
        bh[4] =  0.;
        bh[5] =  0.;
        bh[6] =  .3546095642343226447863179350895055038855;
        bh[7] =  .2478480431366653069619986721504458660016;
        bh[8] =  4.4481347324757844927251283171596488713 12;
        bh[9] =  19.8468863661187336993093239929768793529 1;
        bh[10] = -23.5816233774656184196951796087039496508 5;
        bh[11] =  0.;
        bh[12] =  0.;
        bh[13] = -.3601679437289775162124536737746202409110;
        b=[bh];
        self.set(a,b,c);
    elif str(self.name) == ("RKF6"):
        #Runge-Kutta-Fehlberg 6th degree
        n=7;
        c=[0.0 for i in range(n)];
        a = [[0 for x in range(n)] for y in range(n)];
        bh=[0.0 for i in range(n)];
        c[1] =  1;
        c[2] =  0.2;
        c[3] =  0.3;
        c[4] =  0.6;
        c[5] =  1;
        c[5] = 7.0/8.0;
        a[1][1] =  1;
        a[2][1] =  0.2;
        a[3][1] =  3.0/40.0;
        a[3][2] =  9.0/40.0;
        a[4][1] =  0.3;
        a[4][2] =  -9.0/10.0;
        a[4][3] =  6.0/5.0;
        a[5][1] =  -11.0/54.0;
        a[5][2] =  0.25;
        a[5][3] =  -70.0/27.0;
        a[5][4] =  35.0/27.0;
        bh[1] =  2825.0/27648.0;
        bh[2] =  18575.0/48384.0;
        bh[3] =  13525.0/55296.0;
        bh[4] =  277.0/14336.0;
        bh[5] =  1.0/4.0;
        b=[bh];
        self.set(a,b,c);
```

Program 8.4-8 Runge_Kutta class in python to calculate single or set differential equations

```
# -*- coding: utf-8 -*-
"""
Created on Fri Aug 31 13:10:28 2018

@author: Mustafa Turhan Çoban
"""
from  Butcher import *;
from f_xi import *;

class  Runge_Kutta():
  def __init__(self,namei):
      self.name=namei;
      self.bu=Buthcher(namei);
      self.a=self.bu.a;
      self.b=self.bu.b;
      self.c=self.bu.c;
  def calc_1(self,f,x0,xn,y0,hi,bk):
      ff=float((xn-x0)/hi);
      n=int(ff);
      self.h=hi;
      xx=[0 for i in range(2)];
      x=[0 for i in range(n+1)];
      y=[0 for i in range(n+1)];
      s=len(self.c);
      ki=[0 for i in range(n+1)];
      x[0]=x0;y[0]=y0;
      for ni in range(n):
          for i in range(s):
              xi=x[ni]+self.c[i]*self.h;
              yi=y[ni];
              for j in range(i):
                  yi+=self.h*self.a[i][j]*ki[j];
              xx[0]=xi;
              xx[1]=yi;
              ki[i]=f.func(xx);
          y[ni+1]=y[ni];
          for i in range(s):
              y[ni+1]+=self.h*self.b[bk][i]*ki[i];
          x[ni+1]=x[ni]+self.h;
      XY=[[0 for i2 in range(n+1)] for j2 in range(2)];
      XY[0]=x;
      XY[1]=y;
      return XY;

  def  tasi(self,x1,y1):
      n1=len(y1);
      z=[0 for i in range(n1+1)];
      z[0]=x1;
      for i in range(1,n1+1):
          z[i]=y1[i-1];
      print("z=",z);
      return z;

  def calc_n(self,f,x0,xn,y0,hi,bk):
      #bk index of k if more than one equation is existed
      self.h=hi;
      ff=float((xn-x0)/hi+1e-30);
      n=int(ff);
      m=len(y0);
      x=[0 for i in range(n+1)];
      y=[[0 for i2 in range(m)] for j2 in range(n+1)];
      print(y[0]);
      s=len(self.c);
      z=[0 for i in range(m+1)];
      ki=[0 for i in range(s)];
      x[0]=x0;
      for ni in range(n+1):
          x[ni]=x0+ni*self.h;
      y[0]=y0;
      for ni in range(n):
          for k in range(m):
              for i in range(s):
                  xi=x[ni]+self.c[i]*self.h;
```

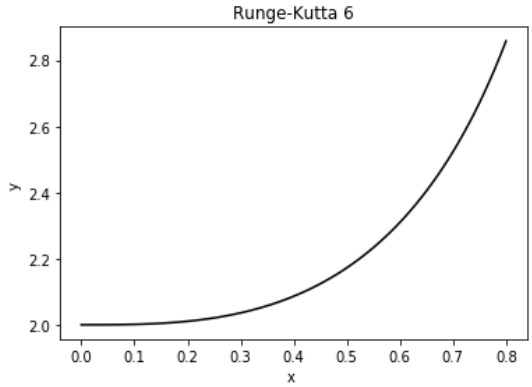
```
                z[0]=xi;
                for ix in range(m):
                    z[ix+1]=y[ni][ix];
                for j in range(i):
                    z[k+1]+=self.h*self.a[i][j]*ki[j];
                ki[i]=f.func(z,k);
            y[ni+1][k]=y[ni][k];
            for i in range(s):
                y[ni+1][k]+=self.h*self.b[bk][i]*ki[i];
    XY=[[0 for i2 in range(n+1)] for j2 in range(m+1)];
    XY[0]=x;
    for k in range(m):
        for ni in range(n+1):
            XY[k+1][ni]=y[ni][k];
    return XY;
```

Program 8.4-9 Sixth order Runge-Kutta (RK6) equation using Buthher class (Python code)

```
# -*- coding: utf-8 -*-
"""
Created on Fri Aug 31 13:10:28 2018

@author: Mustafa Turhan Çoban
"""
from Runge_Kutta import *;
from f_x import *;
import numpy as np;
import matplotlib.pyplot as plt;

class f1(f_x):
    def func(self,X):
        x=X[0];
        y=X[1];
        F=y*y*sin(x)*sin(x);
        return F;


rk=Runge_Kutta("RK6");
f=f1();
XY=rk.calc_1(f,0.0,0.8,2.0,0.01,0);
print(XY);
x=XY[0];
y=XY[1];
plt.title('Runge-Kutta 6')
plt.xlabel('x ')
plt.ylabel('y ');
plt.plot(x,y,'k');
```

```
 runfile('D:/okul/SCO1/Runge_Kutta_test_RK6_1.py', wdir='D:/okul/SCO1')
Reloaded modules: Runge_Kutta, Butcher, f_xi, f_x
[[0.0, 0.01, 0.02, 0.03, 0.04, 0.05, 0.06000000000000005, 0.07, 0.08, 0.09, 0.09999999999999999, 0.10999999999999999,
0.11999999999999998, 0.12999999999999998, 0.13999999999999999, 0.15, 0.16, 0.17, 0.18000000000000002, 0.19000000000000003,
0.20000000000000004, 0.21000000000000005, 0.22000000000000006, 0.23000000000000007, 0.24000000000000007,
0.25000000000000006, 0.26000000000000006, 0.2700000000000001, 0.2800000000000001, 0.2900000000000001,
0.3000000000000001, 0.3100000000000001, 0.3200000000000001, 0.3300000000000001, 0.3400000000000014,
0.35000000000000014, 0.36000000000000015, 0.37000000000000016, 0.38000000000000017, 0.3900000000000002,
0.4000000000000002, 0.4100000000000002, 0.4200000000000002, 0.4300000000000002, 0.4400000000000002,
0.45000000000000023, 0.46000000000000024, 0.47000000000000025, 0.48000000000000026, 0.49000000000000027,
0.5000000000000002, 0.5100000000000002, 0.5200000000000002, 0.5300000000000002, 0.5400000000000003, 0.5500000000000003,
0.5600000000000003, 0.5700000000000003, 0.5800000000000003, 0.5900000000000003, 0.6000000000000003, 0.6100000000000003,
0.6200000000000003, 0.6300000000000003, 0.6400000000000003, 0.6500000000000004, 0.6600000000000004, 0.6700000000000004,
0.6800000000000004, 0.6900000000000004, 0.7000000000000004, 0.7100000000000004, 0.7200000000000004, 0.7300000000000004,
0.7400000000000004, 0.7500000000000004, 0.7600000000000005, 0.7700000000000005, 0.7800000000000005, 0.7900000000000005,
0.8000000000000005], [2.0, 2.0000013333075586, 2.0000106658702514, 2.0000359941683423, 2.0000853096695534,
2.0001665972293488, 2.00028783412938, 2.0004569897517466, 2.0006820258861464, 2.00097089766665, 2.001331555134674,
2.001771945424804, 2.0023000155703943, 2.0029237159263467, 2.0036510042071387, 2.004489850139076, 2.005448240726798,
2.0065341861353527, 2.007755726190629, 2.009120937502593, 2.0106379412176634, 2.0123149114086223, 2.0141600841127394,
2.0161817670313047, 2.0183883499064637, 2.020788315594225, 2.0233902518557123, 2.0262028638922014, 2.029234987653242,
2.032495603951213, 2.035993853420053, 2.0397390523606527, 2.043740709520522, 2.048008543860918, 2.0525525033706464,
2.057382784992311, 2.062509855733902, 2.0679444750463984, 2.07369771855652, 2.0797810032530317, 2.0862061142351456,
2.092985233142692, 2.1001309683999394, 2.1076563874184013, 2.1155750509187783, 2.1239010495485644, 2.1326490429899403,
2.1418343017726333, 2.151472752028696, 2.161581023450873, 2.172176500743812, 2.183277378888052, 2.194902722571101,
2.2070725301782668, 2.2198078027789916, 2.2331306185927375, 2.2470642134728376, 2.2616330680079257, 2.2768630019096614,
2.292781276433536, 2.3094167056680117, 2.3267997776275444, 2.34496278619912, 2.3639399751217804, 2.383767695326836,
```

2.4044845771358694, 2.4261317190077967, 2.4487528947490995, 2.4723947813578184, 2.4971072099676364, 2.5229434427002473, 2.549960478630286, 2.5782193925271737, 2.60778571057397, 2.638729827888825, 2.6711274734069064, 2.7050602285404257, 2.7406161070467014, 2.77789020472977, 2.8169854290175724, 2.858013320140403]]



EXAMPLE :

Solve differential equation

$\frac{dy}{dx} = yx^2 - y$ at the range x=0 ve x=2 , with initial condition y(0)=1.

a) solve analytically from x=0 to x=2.
b) solve numerically by hand with h=0.5 step size with 4th order Runge_Kutte and calculate y(2)
c) solve numerically by computer with h=0.1 step size with 4th order Runge-Kutta
d) Plot to results and compare with analytical solution

Analytical solution :

$$\frac{dy}{dx} = yx^2 - y = y(x^2 - 1)$$

$$\frac{dy}{y} = (x^2 - 1)dx$$

$$\int \frac{dy}{y} = \int (x^2 - 1)dx$$

$$\ln(y) = (\frac{x^3}{3} - x) + C$$

Boundary condition x=0 y=1➔ C=0

$$y = \exp(\frac{x^3}{3} - x)$$

Solution with h=0.5 and h=0.1 by using spreadsheet:

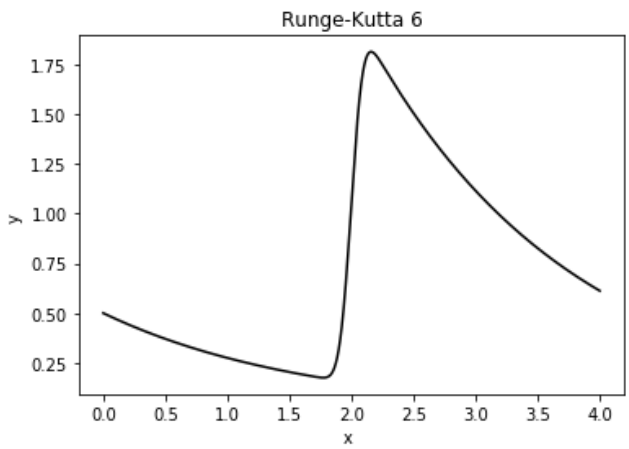
h	0.5														
	x1	y1	k1	x2	y2	k2	x3	y3	k3	x4	y4	k4	y	y exact	error
	0	1	-1	0.25	0.75	-0.703	0.25	0.8242	-0.773	0.5	0.6136	-0.46	1	1	0
	0.5	0.6323	-0.474	0.75	0.5138	-0.225	0.75	0.5761	-0.252	1	0.5063	0	0.6323	0.6323	-5.4E-06
	1	0.5133	0	1.25	0.5133	0.2888	1.25	0.5855	0.3294	1.5	0.678	0.8475	0.5133	0.5134	7.02E-05
	1.5	0.687	0.8587	1.75	0.9017	1.8597	1.75	1.1519	2.3758	2	1.8749	5.6248	0.687	0.6873	0.000294
	2	1.9332	5.7996	2.25	3.3831	13.744	2.25	5.3692	21.812	2.5	12.839	67.407	1.9332	1.9477	0.01452

h	0.1														
	x1	y1	k1	x2	y2	k2	x3	y3	k3	x4	y4	k4	y	y exact	error
	0	1	-1	0.05	0.95	-0.948	0.05	0.9526	-0.95	0.1	0.905	-0.896	1	1	0

	0.1	0.9051	-0.896	0.15	0.8603	-0.841	0.15	0.8631	-0.844	0.2	0.8208	-0.788	0.9051	0.9051	-7E-08
	0.2	0.8209	-0.788	0.25	0.7815	-0.733	0.25	0.7843	-0.735	0.3	0.7474	-0.68	0.8209	0.8209	-1.2E-07
	0.3	0.7475	-0.68	0.35	0.7135	-0.626	0.35	0.7162	-0.628	0.4	0.6847	-0.575	0.7475	0.7475	-1.5E-07
	0.4	0.6848	-0.575	0.45	0.656	-0.523	0.45	0.6586	-0.525	0.5	0.6322	-0.474	0.6848	0.6848	-1.6E-07
	0.5	0.6323	-0.474	0.55	0.6086	-0.425	0.55	0.6111	-0.426	0.6	0.5897	-0.377	0.6323	0.6323	-1.6E-07
	0.6	0.5898	-0.377	0.65	0.5709	-0.33	0.65	0.5733	-0.331	0.7	0.5567	-0.284	0.5898	0.5898	-1.5E-07
	0.7	0.5567	-0.284	0.75	0.5425	-0.237	0.75	0.5449	-0.238	0.8	0.5329	-0.192	0.5567	0.5567	-1.4E-07
	0.8	0.5329	-0.192	0.85	0.5234	-0.145	0.85	0.5257	-0.146	0.9	0.5184	-0.098	0.5329	0.5329	-1.3E-07
	0.9	0.5184	-0.098	0.95	0.5135	-0.05	0.95	0.5159	-0.05	1	0.5134	0	0.5184	0.5184	-1.2E-07
	1	0.5134	0	1.05	0.5134	0.0526	1.05	0.516	0.0529	1.1	0.5187	0.1089	0.5134	0.5134	-1.2E-07
	1.1	0.5188	0.1089	1.15	0.5242	0.1691	1.15	0.5272	0.17	1.2	0.5358	0.2357	0.5188	0.5187	-1.2E-07
	1.2	0.5358	0.2358	1.25	0.5476	0.308	1.25	0.5512	0.31	1.3	0.5668	0.3911	0.5358	0.5358	-1.2E-07
	1.3	0.5668	0.3911	1.35	0.5864	0.4823	1.35	0.591	0.4861	1.4	0.6155	0.5908	0.5668	0.5668	-1.1E-07
	1.4	0.6155	0.5909	1.45	0.645	0.7112	1.45	0.651	0.7178	1.5	0.6873	0.8591	0.6155	0.6155	-6.9E-08
	1.5	0.6873	0.8591	1.55	0.7302	1.0242	1.55	0.7385	1.0357	1.6	0.7909	1.2337	0.6873	0.6873	7.08E-08
	1.6	0.7908	1.2337	1.65	0.8525	1.4685	1.65	0.8643	1.4887	1.7	0.9397	1.776	0.7908	0.7908	5.25E-07
	1.7	0.9396	1.7758	1.75	1.0284	2.121	1.75	1.0456	2.1566	1.8	1.1552	2.5877	0.9396	0.9396	1.9E-06
	1.8	1.1549	2.5869	1.85	1.2842	3.111	1.85	1.3104	3.1745	1.9	1.4723	3.8428	1.1549	1.1549	5.82E-06
	1.9	1.4716	3.8408	1.95	1.6636	4.6622	1.95	1.7047	4.7773	2	1.9493	5.8479	1.4716	1.4716	1.67E-05
	2	1.9477	5.8431	2.05	2.2398	7.1731	2.05	2.3063	7.3861	2.1	2.6863	9.1603	1.9477	1.9477	4.59E-05

Program 8.4-7 Runge-Kutta Fehlberg constant step size Python codes

```
class f1(f_xj):
    def func(self,X):
        x=X[0];
        y=X[1];
        F=-0.6*y+10.0*exp(-(x-2)*(x-2)/(2*0.075*0.075));
        return F;
rk=Runge_Kutta("RKF6");
f=f1();
XY=rk.calc_1(f,0.0,4.0,0.5,0.001,0);
print(XY);
x=XY[0];
y=XY[1];
plt.title('Runge-Kutta 6')
plt.xlabel('x ')
plt.ylabel('y ');
plt.plot(x,y,'k');
```



11.3 MULTISTEP (OPEN) METHODS (PREDICTOR-CORRECTOR METHODS)

If more than one step of equations are used for calculation of one step solution, it is called multistep method. Some people claims that this type of equations gives more stable solutions. **Milne** Formula is the first multistep method to be investigated.

Milne 1. Step formula (first prediction)

$$y_{i+1}^{p} = y_{i-3} + \frac{4h}{3}\left[2f(x_i, y_i) - f(x_{i-1}, y_{i-1}) + 2f(x_{i-2}, y_{i-2})\right]$$

Milne 2. Step Formula (first correction)

$$y_{i+1}^{(0)} = y_{j+1}^{p} + \frac{28}{29}\left[y_i - y_i^{p}\right]$$

Milne 3. Step formula (second correction)

$$y_{i+1}^{(k+1)} = y_{j-1} + \frac{h}{3}\left[f(x_{i+1}, y_{i+1}^{(k)}) + 4f(x_i, y_i) + f(x_{i-1}, y_{i-1})\right] \quad \text{k=0,1,....}$$

Due to using a first step to predict a guess and then corrected it with one or multistep correctors, Multistep methods are also called **Predictor-Corrector** methods. In order to apply this method, previously calculated points are needed. Single step methods like Runge-Kutta methods can serve us for calculating this first points. As an example problem

$y' = \frac{1}{1+x^2} - 2y^2$ diferential equation with the initial conditions y(0)=0 will be solved. Exact analytical solution of this diferential equation

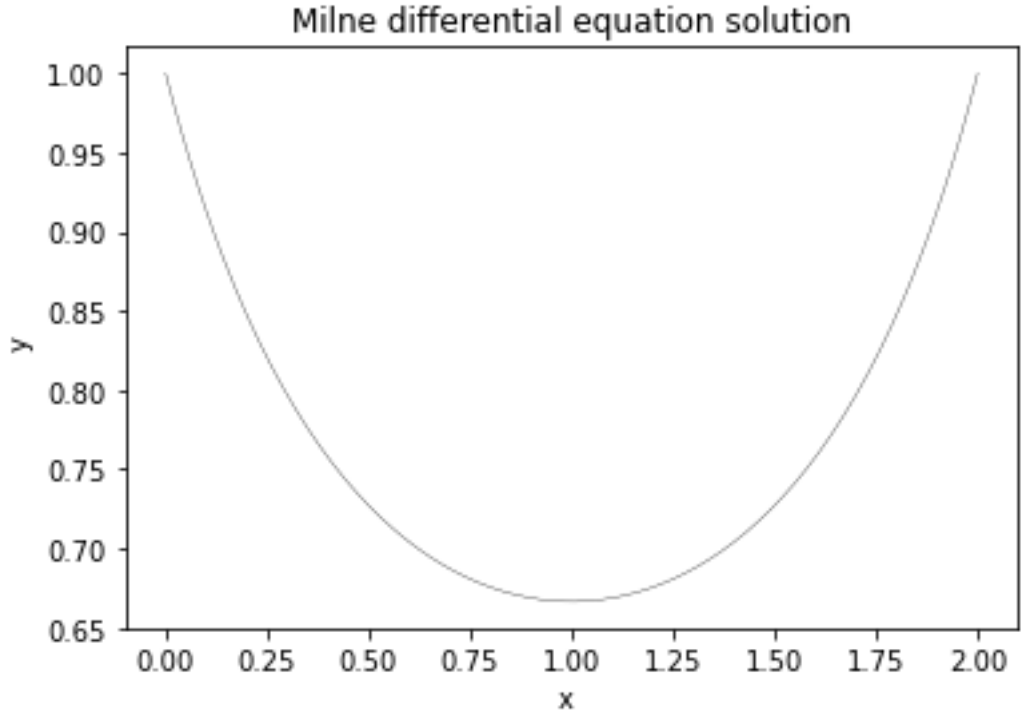
$$y = \frac{x}{1+x^2}$$

```
#=====================================================
# Numerical Analysis package in python
# example to show differential equation solution
# Milne Method
# Dr. Turhan Coban
# ====================================================
# Basic differential equation  dy/dx=f(x,y)
from math import *
from f_x import *
from f_xy import *
import gauss
import numpy as np;
import matplotlib.pyplot as plt;
import os;
def RK6M(f,x0,y0,xn,h):
    #Milne method
    n=int((xn-x0)/h)
    np1=n+1
    a=[[0 for x in range(np1)] for y in range(2)]
    yp=[0 for x in range(np1)]
    y=[[0 for x in range(np1)] for y in range(6)]
    # Runge-Kutta 6
    i=0;
    a[0][0]=x0
    a[1][0]=y0
    x2=x0+3.0*h
    # float k0,k1,k2,k3,k4,k5,k6
    for i in range(3):
        xx=x0+i*h
        a[0][i+1]=xx+h
        k1=f.func(a[0][i],a[1][i])
        k2=f.func(a[0][i]+0.25*h,a[1][i]+0.25*k1*h)
        k3=f.func(a[0][i]+0.25*h,a[1][i]+1.0/8.0*k1*h+1.0/8.0*k2*h)
        k4=f.func(a[0][i]+0.5*h,a[1][i]-0.5*k2*h+k3*h)
        k5=f.func(a[0][i]+3.0/4.0*h,a[1][i]+3.0/16.0*k1*h+9.0/16.0*k4*h)
        k6=f.func(a[0][i]+h,a[1][i]-3.0/7.0*k1*h+2.0/7.0*k2*h+12.0/7.0*k3*h-12.0/7.0*k4*h+8.0/7.0*k5*h)
        a[1][i+1]=a[1][i]+(7.0*k1+32.0*k3+12.0*k4+32.0*k5+7*k6)/90.0*h
    # Milne section
    for i in range(3,n):
        xx=x0+i*h
        a[0][i+1]=xx+h
```

```
        k1=f.func(a[0][i],a[1][i])
        k2=f.func(a[0][i-1],a[1][i-1])
        k3=f.func(a[0][i-2],a[1][i-2])
        yp[i+1]=a[1][i-3]+4.0*h/3.0*(2.0*k1-k2+2.0*k3)
        y[0][i+1]=yp[i+1]+28.0/29.0*(a[1][i]- yp[i])
        for k in range(5):
            k0=f.func(a[0][i+1],y[k][i+1])
            y[k+1][i+1]=a[1][i-1]+h/3.0*(k0+4.0*k1+k2)
        a[1][i+1]=y[k][i+1];
        i=i+1
    return a

x0=0;
y0=1.0;
xn=2.0;
h=0.01;
class fxy(f_xy):func=lambda self,x,y: y*y*(x-1.0)
f1 = fxy()
a=RK6M(f1,x0,y0,xn,h)
x1=a[0]
y1=a[1]
print("x=",a[0])
print("y=",a[1])
plt.title('Milne differential equation solution')
plt.xlabel('x ')
plt.ylabel('y ');
plt.plot(x1,y1,'k',linewidth=0.3)
```

```
runfile('E:/okul/SCO1/Milne_DifEqn.py', wdir='E:/okul/SCO1')
Reloaded modules: f_x, f_xy, gauss
x= [0, 0.01, 0.02, 0.03, 0.04, 0.05, 0.060000000000000005, 0.06999999999999999, 0.08, 0.09, 0.09999999999999999, 0.11, 0.12, 0.13,
0.14, 0.15000000000000002, 0.16, 0.17, 0.18000000000000002, 0.19, 0.2, 0.21000000000000002, 0.22, 0.23, 0.24000000000000002,
0.25, 0.26, 0.27, 0.28, 0.29000000000000004, 0.3, 0.31, 0.32, 0.33, 0.34, 0.35000000000000003, 0.36000000000000004, 0.37, 0.38,
0.39, 0.4, 0.41000000000000003, 0.42000000000000004, 0.43, 0.44, 0.45, 0.46, 0.47000000000000003, 0.48000000000000004, 0.49,
0.5, 0.51, 0.52, 0.53, 0.54, 0.55, 0.56, 0.5700000000000001, 0.5800000000000001, 0.59, 0.6, 0.61, 0.62, 0.63, 0.64, 0.65, 0.66, 0.67,
0.68, 0.6900000000000001, 0.7000000000000001, 0.7100000000000001, 0.72, 0.73, 0.74, 0.75, 0.76, 0.77, 0.78, 0.79, 0.8, 0.81,
0.8200000000000001, 0.8300000000000001, 0.8400000000000001, 0.85, 0.86, 0.87, 0.88, 0.89, 0.9, 0.91, 0.92, 0.93,
0.9400000000000001, 0.9500000000000001, 0.9600000000000001, 0.97, 0.98, 0.99, 1.0, 1.01, 1.02, 1.03, 1.04, 1.05, 1.06, 1.07, 1.08,
1.09, 1.1, 1.11, 1.12, 1.1300000000000001, 1.1400000000000001, 1.1500000000000001, 1.1600000000000001, 1.17, 1.18, 1.19, 1.2,
1.21, 1.22, 1.23, 1.24, 1.25, 1.26, 1.27, 1.28, 1.29, 1.3, 1.31, 1.32, 1.33, 1.34, 1.35, 1.36, 1.37, 1.3800000000000001,
1.3900000000000001, 1.4000000000000001, 1.4100000000000001, 1.42, 1.43, 1.44, 1.45, 1.46, 1.47, 1.48, 1.49, 1.5, 1.51, 1.52, 1.53,
1.54, 1.55, 1.56, 1.57, 1.58, 1.59, 1.6, 1.61, 1.62, 1.6300000000000001, 1.6400000000000001, 1.6500000000000001,
1.6600000000000001, 1.6700000000000002, 1.68, 1.69, 1.7, 1.71, 1.72, 1.73, 1.74, 1.75, 1.76, 1.77, 1.78, 1.79, 1.8, 1.81, 1.82, 1.83,
1.84, 1.85, 1.86, 1.87, 1.8800000000000001, 1.8900000000000001, 1.9000000000000001, 1.9100000000000001, 1.9200000000000002,
1.93, 1.94, 1.95, 1.96, 1.97, 1.98, 1.99, 2.0]
y= [1.0, 0.9901480271300579, 0.9805844283192825, 0.9712981399640686, 0.962278677511495, 0.9535160901834496,
0.9450009462638088, 0.9367242744741126, 0.9286775641279228, 0.920852708592968, 0.9132420098740426, 0.905838125474943,
0.8986340767499914, 0.8916231985855776, 0.8847991509764296, 0.8781558710502452, 0.8716875874181546, 0.8653887741494761,
0.8592541675212643, 0.8532787216178996, 0.8474576271659143, 0.8417862684882069, 0.8362602441603778, 0.8308753250943928,
0.8256274767936999, 0.8205128183781433, 0.815527646248934, 0.8106683938971107, 0.8059316568173912, 0.8013141529677225,
0.796812748794258, 0.7924244202305175, 0.7881462797150891, 0.7839755376364208, 0.7799095302410401, 0.7759456814446476,
0.7720815315422468, 0.768314699318512, 0.7646429114826828, 0.7610639650215444, 0.757575757291629, 0.7541762485669112,
0.7508634927290786, 0.7476355999694708, 0.7444907680251764, 0.7414272450010623, 0.7384433611054221, 0.735537491563329,
0.7327080888106714, 0.7299536454746669, 0.727272726989508, 0.724663934623825, 0.7221259384849403, 0.719657440576462,
0.7172572081614071, 0.7149240368360308, 0.7126567842249044, 0.7104543330596109, 0.7083156251813132, 0.7062396246151369,
0.7042253518578959, 0.7022718469408848, 0.7003782039822898, 0.6985435342327665, 0.6967670008736941, 0.6950477820413584,
0.6933851058536346, 0.6917782134098586, 0.6902263940297824, 0.6887289482273258, 0.6872852231467547, 0.6858945755094822,
0.6845564072334018, 0.6832701283527686, 0.6820351928074885, 0.6808510613368728, 0.6797172374264621, 0.6786332301771244,
0.6775985903976831, 0.676612873451437, 0.67567567548348, 0.6747865962472651, 0.6739452754560643, 0.6731513555926841,
0.6724045183750198, 0.6717044475524763, 0.6710508654760268, 0.6704434958852982, 0.669882100573963, 0.6693664421718193,
0.6688963208967305, 0.668471537336373, 0.668091929278295, 0.6677573344963259, 0.6674676276506561, 0.6672226830842057,
0.6670224117849394, 0.6668667241977787, 0.6667555672416628, 0.6666888871428671, 0.666666666499601, 0.6666888871428671,
0.6667555672416628, 0.6668667241977787, 0.6670224117849394, 0.6672226830842057, 0.6674676276506561, 0.6677573344963259,
0.668091929278295, 0.668471537336373, 0.6688963208967305, 0.6693664421718193, 0.669882100573963, 0.6704434958852982,
0.6710508654760268, 0.6717044475524763, 0.6724045183750198, 0.6731513555926841, 0.6739452754560643, 0.6747865962472651,
0.67567567548348, 0.676612873451437, 0.6775985903976831, 0.6786332301771244, 0.6797172374264621, 0.6808510613368728,
0.6820351928074885, 0.6832701283527686, 0.6845564072334018, 0.6858945755094822, 0.6872852231467547, 0.6887289482273258,
0.6902263940297824, 0.6917782134098586, 0.6933851058536346, 0.6950477820413584, 0.6967670008736941, 0.6985435342327665,
0.7003782039822898, 0.7022718469408848, 0.7042253518578959, 0.7062396246151369, 0.7083156251813132, 0.7104543330596109,
0.7126567842249044, 0.7149240368360308, 0.7172572081614071, 0.719657440576462, 0.7221259384849403, 0.724663934623825,
0.727272726989508, 0.7299536454746669, 0.7327080888106714, 0.735537491563329, 0.7384433611054221, 0.7414272450010623,
0.7444907680251764, 0.7476355999694708, 0.7508634927290786, 0.7541762485669112, 0.757575757291629, 0.7610639650215444,
0.7646429114826828, 0.768314699318512, 0.7720815315422468, 0.7759456814446476, 0.7799095302410401, 0.7839755376364208,
0.7881462797150891, 0.7924244202305175, 0.796812748794258, 0.8013141529677225, 0.8059316568173912, 0.8106683938971107,
0.815527646248934, 0.8205128183781433, 0.8256274767936999, 0.8308753250943928, 0.8362602441603778, 0.8417862684882069,
0.8474576271659143, 0.8532787216178996, 0.8592541675212643, 0.8653887741494761, 0.8716875874181546, 0.8781558710502452,
```

```
0.8847991509764296, 0.8916231985855776, 0.8986340767499914, 0.905838125474943, 0.9132420098740426, 0.920852708592968,
0.9286775641279228, 0.9367242744741126, 0.9450009462638088, 0.9535160901834496, 0.962278677511495, 0.9712981399640686,
0.980584430342099, 0.9901480276104915, 1.0000000025262383]
```



As a second method, Hamming Formula will be investigated. This method is very close to Milne method. As amound of error it is a little worst than Milne method, but its stability is much better.

Hamming 1. step formula (first prediction)

$$y_{i+1}^{p} = y_{i-3} + \frac{4h}{3}\left[2f(x_i, y_i) - f(x_{i-1}, y_{i-1}) + 2f(x_{i-2}, y_{i-2})\right]$$

Hamming 2. Step formula (first correction)

$$y_{i+1}^{(0)} = y_{j+1}^{p} + \frac{112}{121}\left[y_i - y_i^{p}\right]$$

Hamming 3. Step formula (second correction)

$$y_{i+1}^{(k+1)} = \frac{1}{8}\left(9y_i - y_{i-2}\right) + \frac{3h}{8}\left[f(x_{i+1}, y_{i+1}^{(k)}) + 2f(x_i, y_i) - f(x_{i-1}, y_{i-1})\right] \quad \text{k=0,1,….}$$

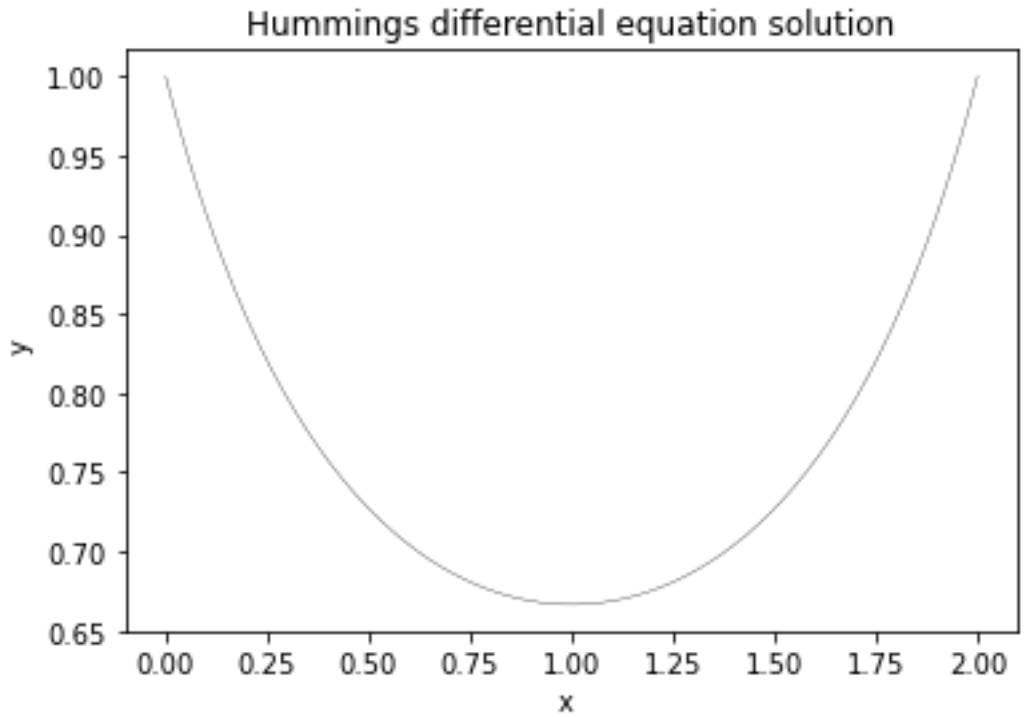
```
#=====================================================
# Numerical Analysis package in python
# example to show differential equation solution
# Hummings Method
# Dr. Turhan Coban
# ====================================================
# Basic differential equation  dy/dx=f(x,y)
from math import *
from f_x import *
from f_xy import *
import gauss
import numpy as np;
import matplotlib.pyplot as plt;
import os;
def RK6M(f,x0,y0,xn,h):
    #Milne method
    n=int((xn-x0)/h)
    np1=n+1
    a=[[0 for x in range(np1)] for y in range(2)]
    yp=[0 for x in range(np1)]
    y=[[0 for x in range(np1)] for y in range(6)]
    # Runge-Kutta 6
```

```
    i=0;
    a[0][0]=x0
    a[1][0]=y0
    x2=x0+3.0*h
    # float k0,k1,k2,k3,k4,k5,k6
    for i in range(3):
        xx=x0+i*h
        a[0][i+1]=xx+h
        k1=f.func(a[0][i],a[1][i])
        k2=f.func(a[0][i]+0.25*h,a[1][i]+0.25*k1*h)
        k3=f.func(a[0][i]+0.25*h,a[1][i]+1.0/8.0*k1*h+1.0/8.0*k2*h)
        k4=f.func(a[0][i]+0.5*h,a[1][i]-0.5*k2*h+k3*h)
        k5=f.func(a[0][i]+3.0/4.0*h,a[1][i]+3.0/16.0*k1*h+9.0/16.0*k4*h)
        k6=f.func(a[0][i]+h,a[1][i]-3.0/7.0*k1*h+2.0/7.0*k2*h+12.0/7.0*k3*h-12.0/7.0*k4*h+8.0/7.0*k5*h)
        a[1][i+1]=a[1][i]+(7.0*k1+32.0*k3+12.0*k4+32.0*k5+7*k6)/90.0*h
    # Hummings  section
    for i in range(3,n):
        xx=x0+i*h
        a[0][i+1]=xx+h
        k1=f.func(a[0][i],a[1][i])
        k2=f.func(a[0][i-1],a[1][i-1])
        k3=f.func(a[0][i-2],a[1][i-2])
        yp[i+1]=a[1][i-3]+4.0*h/3.0*(2.0*k1-k2+2.0*k3)
        y[0][i+1]=yp[i+1]+112.0/121.0*(a[1][i]- yp[i])
        for k in range(5):
            k0=f.func(a[0][i+1],y[k][i+1]);
            y[k+1][i+1]=1.0/8.0*(9.0*a[1][i]-a[1][i-2])+3.0*h/8.0*(k0+2.0*k1-k2);
        a[1][i+1]=y[k][i+1];
        i=i+1
    return a

x0=0;
y0=1.0;
xn=2.0;
h=0.01;
class fxy(f_xy):func=lambda self,x,y: y*y*(x-1.0)
f1 = fxy()
a=RK6M(f1,x0,y0,xn,h)
x1=a[0]
y1=a[1]
print("x=",a[0])
print("y=",a[1])
plt.title('Hummings differential equation solution')
plt.xlabel('x ')
plt.ylabel('y ');
plt.plot(x1,y1,'k',linewidth=0.3)
```

```
runfile('E:/okul/SCO1/Hummings_DifEqn.py', wdir='E:/okul/SCO1')
Reloaded modules: f_x, f_xy, gauss
x= [0, 0.01, 0.02, 0.03, 0.04, 0.05, 0.060000000000000005, 0.06999999999999999, 0.08, 0.09, 0.09999999999999999, 0.11, 0.12, 0.13,
0.14, 0.15000000000000002, 0.16, 0.17, 0.18000000000000002, 0.19, 0.2, 0.21000000000000002, 0.22, 0.23, 0.24000000000000002,
0.25, 0.26, 0.27, 0.28, 0.29000000000000004, 0.3, 0.31, 0.32, 0.33, 0.34, 0.35000000000000003, 0.36000000000000004, 0.37, 0.38,
0.39, 0.4, 0.41000000000000003, 0.42000000000000004, 0.43, 0.44, 0.45, 0.46, 0.47000000000000003, 0.48000000000000004, 0.49,
0.5, 0.51, 0.52, 0.53, 0.54, 0.55, 0.56, 0.5700000000000001, 0.5800000000000001, 0.59, 0.6, 0.61, 0.62, 0.63, 0.64, 0.65, 0.66, 0.67,
0.68, 0.6900000000000001, 0.7000000000000001, 0.7100000000000001, 0.72, 0.73, 0.74, 0.75, 0.76, 0.77, 0.78, 0.79, 0.8, 0.81,
0.8200000000000001, 0.8300000000000001, 0.8400000000000001, 0.85, 0.86, 0.87, 0.88, 0.89, 0.9, 0.91, 0.92, 0.93,
0.9400000000000001, 0.9500000000000001, 0.9600000000000001, 0.97, 0.98, 0.99, 1.0, 1.01, 1.02, 1.03, 1.04, 1.05, 1.06, 1.07, 1.08,
1.09, 1.1, 1.11, 1.12, 1.1300000000000001, 1.1400000000000001, 1.1500000000000001, 1.1600000000000001, 1.17, 1.18, 1.19, 1.2,
1.21, 1.22, 1.23, 1.24, 1.25, 1.26, 1.27, 1.28, 1.29, 1.3, 1.31, 1.32, 1.33, 1.34, 1.35, 1.36, 1.37, 1.3800000000000001,
1.3900000000000001, 1.4000000000000001, 1.4100000000000001, 1.42, 1.43, 1.44, 1.45, 1.46, 1.47, 1.48, 1.49, 1.5, 1.51, 1.52, 1.53,
1.54, 1.55, 1.56, 1.57, 1.58, 1.59, 1.6, 1.61, 1.62, 1.6300000000000001, 1.6400000000000001, 1.6500000000000001,
1.6600000000000001, 1.6700000000000002, 1.68, 1.69, 1.7, 1.71, 1.72, 1.73, 1.74, 1.75, 1.76, 1.77, 1.78, 1.79, 1.8, 1.81, 1.82, 1.83,
1.84, 1.85, 1.86, 1.87, 1.8800000000000001, 1.8900000000000001, 1.9000000000000001, 1.9100000000000001, 1.9200000000000002,
1.93, 1.94, 1.95, 1.96, 1.97, 1.98, 1.99, 2.0]
y= [1.0, 0.9901480271300579, 0.9805844283192825, 0.9712981399640686, 0.9622786780336916, 0.9535160921054695,
0.9450009459286753, 0.9367242752698179, 0.9286775623751528, 0.9208527080264512, 0.9132420068022803, 0.905838123707979,
0.8986340725553045, 0.8916231958111659, 0.8847991458430924, 0.8781558674386735, 0.8716875815033508, 0.8653887698441847,
0.8592541609567252, 0.8532787167388171, 0.8474576200618223, 0.841786263135608, 0.8362602366089968, 0.8308753193522336,
0.8256274688724126, 0.8205128123168153, 0.8155276380227376, 0.8106683875757004, 0.8059316483409443, 0.8013141464358676,
0.7968127401135662, 0.7924244135299369, 0.7881462708688951, 0.7839755308021794, 0.7799095212619762, 0.7759456745062057,
0.7720815224577917, 0.768314692300598, 0.7646429023159589, 0.7610639579448831, 0.7575757480620683, 0.7541762414488336,
0.7508634834529815, 0.7476355928244246, 0.7444907587161802, 0.7414272378410425, 0.7384433517748961, 0.7355374843982398,
0.7327080794680497, 0.7299536383126284, 0.7272727176425723, 0.724663927471432, 0.7221259291400576, 0.719657433439004,
0.7172571988237332, 0.7149240297176792, 0.7126567748985565, 0.7104543259635776, 0.7083156158695171, 0.7062396175438105,
0.7042253425631113, 0.7022718398959463, 0.7003781947063177, 0.6985435272152883, 0.6967669916177689, 0.6950477750518899,
0.6933850966185028, 0.6917782064484984, 0.6902263848157708, 0.688728941293785, 0.687285213953828, 0.6858945686031377,
```

0.6845563980612128, 0.6832701214727094, 0.6820351836554213, 0.6808510544819405, 0.6797172282936697, 0.6786332233459474, 0.6775985812831193, 0.6766128666424606, 0.6756756663859275, 0.6747865894587772, 0.6739452663741587, 0.6731513488228374, 0.6724045093072708, 0.671704440799308, 0.6710508564208366, 0.6704434891467477, 0.6698820915296425, 0.6693664354457425, 0.6688963118615134, 0.6684715306205558, 0.6680919202503506, 0.667757327788496, 0.6674676186281002, 0.6672226763820431, 0.6670224027658448, 0.6668667174989256, 0.6667555582240686, 0.6666888804449362, 0.6666666574815211, 0.6666888804434506, 0.6667555582210943, 0.6668667174944563, 0.667022402759871, 0.667222676374552, 0.6674676186190757, 0.6677573277779184, 0.6680919202381975, 0.6684715306068008, 0.6688963118461267, 0.6693664354286907, 0.6698820915108884, 0.67044348912625, 0.6710508563985502, 0.6717044407751838, 0.6724045092812547, 0.6731513487948715, 0.6739452663441802, 0.6747865894267182, 0.6756756663517152, 0.6766128666060165, 0.6775985812443596, 0.6786332233047818, 0.6797172282500018, 0.6808510544356674, 0.6820351836064328, 0.6832701214208879, 0.6845563980064332, 0.685894568545266, 0.6872852138927219, 0.6887289412292925, 0.69022638474773, 0.6917782063767367, 0.6933850965428363, 0.6950477749721228, 0.6967669915336924, 0.6985435271266801, 0.7003781946129406, 0.7022718397975478, 0.7042253424594221, 0.7062396174345436, 0.7083156157543661, 0.7104543258422159, 0.7126567747706354, 0.7149240295828263, 0.7172571986815512, 0.7196574332890685, 0.7221259289819147, 0.7246639273045968, 0.7272727174665261, 0.7299536381268164, 0.7327080792718776, 0.7355374841910715, 0.7384433515560496, 0.7414272376097873, 0.7444907584717328, 0.7476355925659439, 0.7508634831795644, 0.7541762411595103, 0.7575757477557963, 0.761063957620541, 0.7646429019723401, 0.7683146919364032, 0.7720815220716211, 0.77594567409655, 0.7799095208272072, 0.7839755303405392, 0.7881462703784848, 0.7924244130087039, 0.7968127395592897, 0.8013141458461436, 0.8059316477131683, 0.8106683869070485, 0.8155276373101459, 0.8205128115569564, 0.825627468061671, 0.8308753184866764, 0.8362602356843429, 0.8417862621471932, 0.8474576190045611, 0.8532787156071587, 0.8592541597446083, 0.8653887685449847, 0.8716875801098263, 0.8781558659429141, 0.884799144236472, 0.8916231940843519, 0.8986340706982981, 0.9058381217106182, 0.9132420046545999, 0.9208527057229083, 0.9286775599140656, 0.9367242726881113, 0.9450009432647626, 0.9535160897110406, 0.9622786759805589, 0.9712981410836944, 0.9805844305869331, 0.9901480306610596, 1.0000000049218651]

In [144]:



First points calculated by Runge Kutta -6 method is given with bold font. As the third method **Adams-Bashford-Moulton formulas.** This formulas like Runge-Kutta sets can be given in different orders. They are given as two sets. The first set (Adams-Bashford) is used as a predictor, and the second set is used as a corrector step.

Predictor Formula :

Adams-Bashford 1 step formula

$$y_{i+1} = y_i + hf(x_i, y_i)$$

Adams-Bashford 2 step formula

$$y_{i+1} = y_i + \frac{h}{2}\left(3f(x_i, y_i) - f(x_{i-1}, y_{i-1})\right)$$

Adams-Bashford 3 step formula

$$y_{i+1} = y_i + \frac{h}{12}\left(23f(x_i, y_i) - 16f(x_{i-1}, y_{i-1}) + 5f(x_{i-2}, y_{i-2})\right)$$

Adams-Bashford 4 step formula

$$y_{i+1}=y_i+\frac{h}{24}\left(55f(x_i,y_i)-59f(x_{i-1},y_{i-1})+37f(x_{i-2},y_{i-2})-9f(x_{i-3},y_{i-3})\right)$$

Adams-Bashford 5 step formula

$$y_{i+1}=y_i+\frac{h}{720}\left(1901f(x_i,y_i)-2774f(x_{i-1},y_{i-1})+2616f(x_{i-2},y_{i-2})-1274f(x_{i-3},y_{i-3})+251f(x_{i-4},y_{i-4})\right)$$

Corrector Formula :

Adams-Moulton 0 step formula

$$y_{i+1}=y_i+hf(x_{i+1},y_{i+1})$$

Adams-Moulton 1 step formula

$$y_{i+1}=y_i+\frac{h}{2}\left(f(x_{i+1},y_{i+1})+f(x_i,y_i)\right)$$

Adams-Moulton 2 step formula

$$y_{i+1}=y_i+\frac{h}{12}\left(5f(x_{i+1},y_{i+!})+8f(x_i,y_i)-f(x_{i-1},y_{i-!})\right)$$

Adams-Moulton 3 step formula

$$y_{i+1}=y_i+\frac{h}{24}\left(9f(x_{i+1},y_{i+1})+19f(x_i,y_i)-5f(x_{i-1},y_{i-1})+f(x_{i-2},y_{i-2})\right)$$

Adams-Moulton 4 step formula

$$y_{i+1}=y_i+\frac{h}{720}\left(251f(x_{i+1},y_{i+1})+646f(x_i,y_i)-264f(x_{i-1},y_{i-1})+106f(x_{i-2},y_{i-2})-19f(x_{i-3},y_{i-3})\right)$$

Program list is given as the following program. In this program two method is defined. In the first method Adams-Bashford 4 step Formula is used together with Adams-Milton 3 step Formula. In the second method Adams-Bashford 5 step Formula is used together with Adams-Milton 4 step Formula.

```
#=====================================================
# Numerical Analysis package in java
# example to show differential equation solution
# Adams-BAshford-Milton Predictor Corrector formula
# with Adams-Bashford 4 step predictor and Adams-Milford 3 step corrector
# also with Adams-Bashford 5 step predictor and Adams-Milford 4 step corrector
# Dr. Turhan Coban
# ====================================================
# dy/dx=f(x,y)
from math import *
from f_x import *
from f_xy import *
import gauss
import numpy as np;
import matplotlib.pyplot as plt;
import os;

def RK6AB4AM3(f,x0,y0,xn,h):
    # Adams-Bashford (4 adım formülü)  Adams-Moulton (3 adım formülü) predictor corrector method
    # with RK6 initial step
    n=int((xn-x0)/h)
    np1=n+1
    a=[[0 for x in range(np1)] for y in range(2)]
    for i in range(3):
        xx=x0+i*h
        a[0][i+1]=xx+h
        k1=f.func(a[0][i],a[1][i])
        k2=f.func(a[0][i]+0.25*h,a[1][i]+0.25*k1*h)
        k3=f.func(a[0][i]+0.25*h,a[1][i]+1.0/8.0*k1*h+1.0/8.0*k2*h)
        k4=f.func(a[0][i]+0.5*h,a[1][i]-0.5*k2*h+k3*h)
        k5=f.func(a[0][i]+3.0/4.0*h,a[1][i]+3.0/16.0*k1*h+9.0/16.0*k4*h)
        k6=f.func(a[0][i]+h,a[1][i]-3.0/7.0*k1*h+2.0/7.0*k2*h+12.0/7.0*k3*h-12.0/7.0*k4*h+8.0/7.0*k5*h)
        a[1][i+1]=a[1][i]+(7.0*k1+32.0*k3+12.0*k4+32.0*k5+7*k6)/90.0*h
    # Adams-Bashford section
    for i in range(3,n):
        xx=x0+i*h
        a[0][i+1]=xx+h;
        #Adams-Bashford
```

```
        k1=f.func(a[0][i],a[1][i]);
        k2=f.func(a[0][i-1],a[1][i-1]);
        k3=f.func(a[0][i-2],a[1][i-2]);
        k4=f.func(a[0][i-3],a[1][i-3]);
        a[1][i+1]=a[1][i]+h/24.0*(55.0*k1-59.0*k2+37.0*k3-9.0*k4);
        #Adams-Moulton
        for j in range(5):
            k0=f.func(a[0][i+1],a[1][i+1])
            a[1][i+1]=a[1][i]+h/24.0*(9.0*k0+19.0*k1-5.0*k2+k3);
    return a

def RK6AB5AM4(f,x0,y0,xn,h):
    # Adams-Bashford (5 astep formula)  Adams-Moulton (4 step formula)
    # predictor corrector method
    # with RK6 initial step
    n=int((xn-x0)/h)
    np1=n+1
    a=[[0 for x in range(np1)] for y in range(2)]
    for i in range(3):
        xx=x0+i*h
        a[0][i+1]=xx+h
        k1=f.func(a[0][i],a[1][i])
        k2=f.func(a[0][i]+0.25*h,a[1][i]+0.25*k1*h)
        k3=f.func(a[0][i]+0.25*h,a[1][i]+1.0/8.0*k1*h+1.0/8.0*k2*h)
        k4=f.func(a[0][i]+0.5*h,a[1][i]-0.5*k2*h+k3*h)
        k5=f.func(a[0][i]+3.0/4.0*h,a[1][i]+3.0/16.0*k1*h+9.0/16.0*k4*h)
        k6=f.func(a[0][i]+h,a[1][i]-3.0/7.0*k1*h+2.0/7.0*k2*h+12.0/7.0*k3*h-12.0/7.0*k4*h+8.0/7.0*k5*h)
        a[1][i+1]=a[1][i]+(7.0*k1+32.0*k3+12.0*k4+32.0*k5+7*k6)/90.0*h
    for i in range(3,n):
       xx=x0+i*h
       a[0][i+1]=xx+h
       #Adams-Bashford
       k1=f.func(a[0][i],a[1][i]);
       k2=f.func(a[0][i-1],a[1][i-1]);
       k3=f.func(a[0][i-2],a[1][i-2]);
       k4=f.func(a[0][i-3],a[1][i-3]);
       k5=f.func(a[0][i-4],a[1][i-4]);
       a[1][i+1]=a[1][i]+h/720.0*(1901.0*k1-2774.0*k2+2616.0*k3-1274.0*k4+251*k5);
       #Adams-Moulton j>=1 1
       for j in range(5):
           k0=f.func(a[0][i+1],a[1][i+1])
           a[1][i+1]=a[1][i]+h/720.0*(251.0*k0+646.0*k1-264.0*k2+106.0*k3-19*k4)
    return a

x0=0;
y0=1.0;
xn=10.0;
h=0.01;
class fxy(f_xy):func=lambda self,x,y: 1.0/(1.0+x*x)-2.0*y*y
f1 = fxy()
a=RK6AB4AM3(f1,x0,y0,xn,h)
x1=a[0]
y1=a[1]
print("x=",a[0])
print("y=",a[1])
plt.title('Hummings differential equation solution')
plt.xlabel('x ')
plt.ylabel('y ');
plt.plot(x1,y1,'k',linewidth=0.3)
```

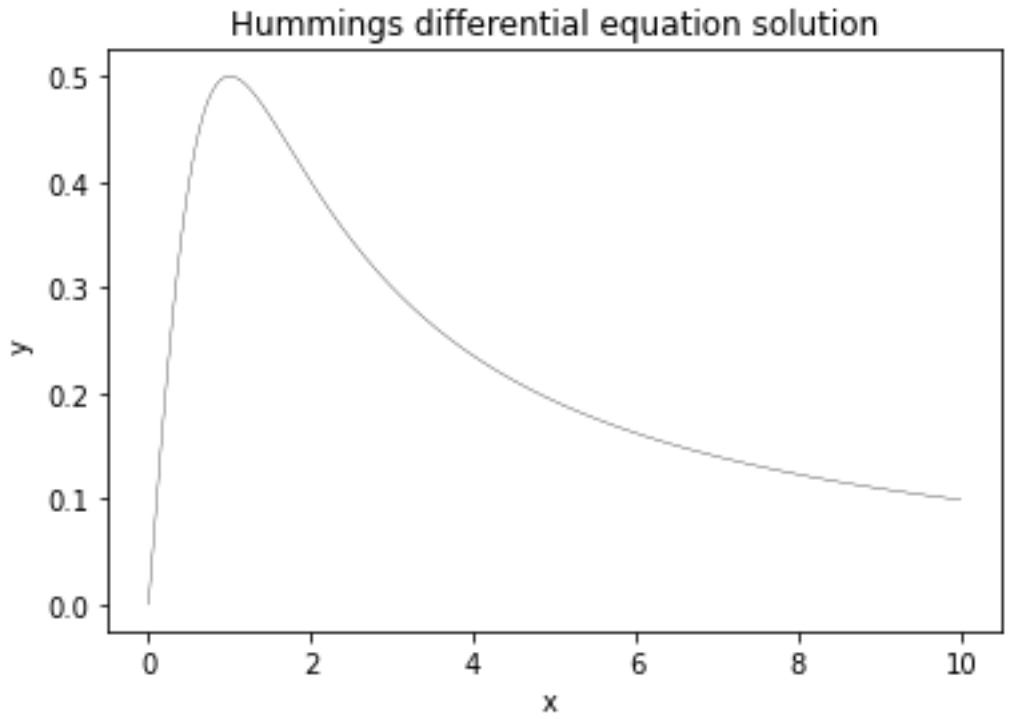
```
runfile('E:/okul/SCO1/Adams_Bashford_DifEqn.py', wdir='E:/okul/SCO1')
Reloaded modules: f_x, f_xy, gauss
x= [0, 0.01, 0.02, 0.03, 0.04, 0.05, 0.060000000000000005, 0.06999999999999999, 0.08, 0.09, 0.09999999999999999, 0.11, 0.12, 0.13,
0.14, 0.15000000000000002, 0.16, 0.17, 0.18000000000000002, 0.19, 0.2, 0.21000000000000002, 0.22, 0.23, 0.24000000000000002,
0.25, 0.26, 0.27, 0.28, 0.29000000000000004, 0.3, 0.31, 0.32, 0.33, 0.34, 0.35000000000000003, 0.36000000000000004, 0.37, 0.38,
0.39, 0.4, 0.41000000000000003, 0.42000000000000004, 0.43, 0.44, 0.45, 0.46, 0.47000000000000003, 0.48000000000000004, 0.49,
0.5, 0.51, 0.52, 0.53, 0.54, 0.55, 0.56, 0.5700000000000001, 0.5800000000000001, 0.59, 0.6, 0.61, 0.62, 0.63, 0.64, 0.65, 0.66, 0.67,
0.68, 0.6900000000000001, 0.7000000000000001, 0.7100000000000001, 0.72, 0.73, 0.74, 0.75, 0.76, 0.77, 0.78, 0.79, 0.8, 0.81,
0.8200000000000001, 0.8300000000000001, 0.8400000000000001, 0.85, 0.86, 0.87, 0.88, 0.89, 0.9, 0.91, 0.92, 0.93,
0.9400000000000001, 0.9500000000000001, 0.9600000000000001, 0.97, 0.98, 0.99, 1.0, 1.01, 1.02, 1.03, 1.04, 1.05, 1.06, 1.07, 1.08,
1.09, 1.1, 1.11, 1.12, 1.1300000000000001, 1.1400000000000001, 1.1500000000000001, 1.1600000000000001, 1.17, 1.18, 1.19, 1.2,
1.21, 1.22, 1.23, 1.24, 1.25, 1.26, 1.27, 1.28, 1.29, 1.3, 1.31, 1.32, 1.33, 1.34, 1.35, 1.36, 1.37, 1.3800000000000001,
1.3900000000000001, 1.4000000000000001, 1.4100000000000001, 1.42, 1.43, 1.44, 1.45, 1.46, 1.47, 1.48, 1.49, 1.5, 1.51, 1.52, 1.53,
1.54, 1.55, 1.56, 1.57, 1.58, 1.59, 1.6, 1.61, 1.62, 1.6300000000000001, 1.6400000000000001, 1.6500000000000001,
1.6600000000000001, 1.6700000000000002, 1.68, 1.69, 1.7, 1.71, 1.72, 1.73, 1.74, 1.75, 1.76, 1.77, 1.78, 1.79, 1.8, 1.81, 1.82, 1.83,
1.84, 1.85, 1.86, 1.87, 1.8800000000000001, 1.8900000000000001, 1.9000000000000001, 1.9100000000000001, 1.9200000000000002,
```

1.93, 1.94, 1.95, 1.96, 1.97, 1.98, 1.99, 2.0, 2.01, 2.02, 2.03, 2.04, 2.05, 2.0599999999999996, 2.07, 2.0799999999999996, 2.09, 2.0999999999999996, 2.11, 2.1199999999999997, 2.13, 2.1399999999999997, 2.15, 2.1599999999999997, 2.17, 2.1799999999999997, 2.19, 2.1999999999999997, 2.21, 2.2199999999999998, 2.23, 2.2399999999999998, 2.25, 2.26, 2.27, 2.28, 2.29, 2.3, 2.31, 2.32, 2.3299999999999996, 2.34, 2.3499999999999996, 2.36, 2.3699999999999997, 2.38, 2.3899999999999997, 2.4, 2.4099999999999997, 2.42, 2.4299999999999997, 2.44, 2.4499999999999997, 2.46, 2.4699999999999998, 2.48, 2.4899999999999998, 2.5, 2.51, 2.52, 2.53, 2.54, 2.55, 2.56, 2.57, 2.5799999999999996, 2.59, 2.5999999999999996, 2.61, 2.6199999999999997, 2.63, 2.6399999999999997, 2.65, 2.6599999999999997, 2.67, 2.6799999999999997, 2.69, 2.6999999999999997, 2.71, 2.7199999999999998, 2.73, 2.7399999999999998, 2.75, 2.76, 2.77, 2.78, 2.79, 2.8, 2.81, 2.82, 2.8299999999999996, 2.84, 2.8499999999999996, 2.86, 2.8699999999999997, 2.88, 2.8899999999999997, 2.9, 2.9099999999999997, 2.92, 2.9299999999999997, 2.94, 2.9499999999999997, 2.96, 2.9699999999999998, 2.98, 2.9899999999999998, 3.0, 3.01, 3.02, 3.03, 3.04, 3.05, 3.06, 3.07, 3.08, 3.09, 3.0999999999999996, 3.11, 3.1199999999999997, 3.13, 3.1399999999999997, 3.15, 3.1599999999999997, 3.17, 3.1799999999999997, 3.19, 3.1999999999999997, 3.21, 3.2199999999999998, 3.23, 3.2399999999999998, 3.25, 3.26, 3.27, 3.28, 3.29, 3.3, 3.31, 3.32, 3.33, 3.34, 3.3499999999999996, 3.36, 3.3699999999999997, 3.38, 3.3899999999999997, 3.4, 3.4099999999999997, 3.42, 3.4299999999999997, 3.44, 3.4499999999999997, 3.46, 3.4699999999999998, 3.48, 3.4899999999999998, 3.5, 3.51, 3.52, 3.53, 3.54, 3.55, 3.56, 3.57, 3.58, 3.59, 3.5999999999999996, 3.61, 3.6199999999999997, 3.63, 3.6399999999999997, 3.65, 3.6599999999999997, 3.67, 3.6799999999999997, 3.69, 3.6999999999999997, 3.71, 3.7199999999999998, 3.73, 3.7399999999999998, 3.75, 3.76, 3.77, 3.78, 3.79, 3.8, 3.81, 3.82, 3.83, 3.84, 3.8499999999999996, 3.86, 3.8699999999999997, 3.88, 3.8899999999999997, 3.9, 3.9099999999999997, 3.92, 3.9299999999999997, 3.94, 3.9499999999999997, 3.96, 3.9699999999999998, 3.98, 3.9899999999999998, 4.0, 4.01, 4.02, 4.03, 4.04, 4.05, 4.06, 4.07, 4.08, 4.09, 4.1, 4.109999999999999, 4.12, 4.13, 4.14, 4.1499999999999995, 4.16, 4.17, 4.18, 4.1899999999999995, 4.2, 4.21, 4.22, 4.2299999999999995, 4.24, 4.25, 4.26, 4.27, 4.28, 4.29, 4.3, 4.31, 4.32, 4.33, 4.34, 4.35, 4.36, 4.37, 4.38, 4.39, 4.3999999999999995, 4.41, 4.42, 4.43, 4.4399999999999995, 4.45, 4.46, 4.47, 4.4799999999999995, 4.49, 4.5, 4.51, 4.52, 4.53, 4.54, 4.55, 4.56, 4.57, 4.58, 4.59, 4.6, 4.61, 4.62, 4.63, 4.64, 4.6499999999999995, 4.66, 4.67, 4.68, 4.6899999999999995, 4.7, 4.71, 4.72, 4.7299999999999995, 4.74, 4.75, 4.76, 4.77, 4.78, 4.79, 4.8, 4.81, 4.82, 4.83, 4.84, 4.85, 4.86, 4.87, 4.88, 4.89, 4.8999999999999995, 4.91, 4.92, 4.93, 4.9399999999999995, 4.95, 4.96, 4.97, 4.9799999999999995, 4.99, 5.0, 5.01, 5.02, 5.03, 5.04, 5.05, 5.06, 5.07, 5.08, 5.09, 5.1, 5.11, 5.12, 5.13, 5.14, 5.1499999999999995, 5.16, 5.17, 5.18, 5.1899999999999995, 5.2, 5.21, 5.22, 5.2299999999999995, 5.24, 5.25, 5.26, 5.27, 5.28, 5.29, 5.3, 5.31, 5.32, 5.33, 5.34, 5.35, 5.36, 5.37, 5.38, 5.39, 5.3999999999999995, 5.41, 5.42, 5.43, 5.4399999999999995, 5.45, 5.46, 5.47, 5.4799999999999995, 5.49, 5.5, 5.51, 5.52, 5.53, 5.54, 5.55, 5.56, 5.57, 5.58, 5.59, 5.6, 5.61, 5.62, 5.63, 5.64, 5.6499999999999995, 5.66, 5.67, 5.68, 5.6899999999999995, 5.7, 5.71, 5.72, 5.7299999999999995, 5.74, 5.75, 5.76, 5.77, 5.78, 5.79, 5.8, 5.81, 5.82, 5.83, 5.84, 5.85, 5.86, 5.87, 5.88, 5.89, 5.8999999999999995, 5.91, 5.92, 5.93, 5.9399999999999995, 5.95, 5.96, 5.97, 5.9799999999999995, 5.99, 6.0, 6.01, 6.02, 6.03, 6.04, 6.05, 6.06, 6.07, 6.08, 6.09, 6.1, 6.11, 6.12, 6.13, 6.14, 6.15, 6.16, 6.17, 6.18, 6.1899999999999995, 6.2, 6.21, 6.22, 6.2299999999999995, 6.24, 6.25, 6.26, 6.27, 6.28, 6.29, 6.3, 6.31, 6.32, 6.33, 6.34, 6.35, 6.36, 6.37, 6.38, 6.39, 6.4, 6.41, 6.42, 6.43, 6.4399999999999995, 6.45, 6.46, 6.47, 6.4799999999999995, 6.49, 6.5, 6.51, 6.52, 6.53, 6.54, 6.55, 6.56, 6.57, 6.58, 6.59, 6.6, 6.61, 6.62, 6.63, 6.64, 6.65, 6.66, 6.67, 6.68, 6.6899999999999995, 6.7, 6.71, 6.72, 6.7299999999999995, 6.74, 6.75, 6.76, 6.77, 6.78, 6.79, 6.8, 6.81, 6.82, 6.83, 6.84, 6.85, 6.86, 6.87, 6.88, 6.89, 6.9, 6.91, 6.92, 6.93, 6.9399999999999995, 6.95, 6.96, 6.97, 6.9799999999999995, 6.99, 7.0, 7.01, 7.02, 7.03, 7.04, 7.05, 7.06, 7.07, 7.08, 7.09, 7.1, 7.11, 7.12, 7.13, 7.14, 7.15, 7.16, 7.17, 7.18, 7.1899999999999995, 7.2, 7.21, 7.22, 7.2299999999999995, 7.24, 7.25, 7.26, 7.27, 7.28, 7.29, 7.3, 7.31, 7.32, 7.33, 7.34, 7.35, 7.36, 7.37, 7.38, 7.39, 7.4, 7.41, 7.42, 7.43, 7.4399999999999995, 7.45, 7.46, 7.47, 7.4799999999999995, 7.49, 7.5, 7.51, 7.52, 7.53, 7.54, 7.55, 7.56, 7.57, 7.58, 7.59, 7.6, 7.61, 7.62, 7.63, 7.64, 7.65, 7.66, 7.67, 7.68, 7.6899999999999995, 7.7, 7.71, 7.72, 7.7299999999999995, 7.74, 7.75, 7.76, 7.77, 7.78, 7.79, 7.8, 7.81, 7.82, 7.83, 7.84, 7.85, 7.86, 7.87, 7.88, 7.89, 7.9, 7.91, 7.92, 7.93, 7.94, 7.95, 7.96, 7.97, 7.9799999999999995, 7.99, 8.0, 8.01, 8.02, 8.03, 8.04, 8.05, 8.06, 8.07, 8.08, 8.09, 8.1, 8.11, 8.12, 8.13, 8.14, 8.15, 8.16, 8.17, 8.18, 8.19, 8.2, 8.209999999999999, 8.22, 8.23, 8.24, 8.25, 8.26, 8.27, 8.28, 8.29, 8.3, 8.31, 8.32, 8.33, 8.34, 8.35, 8.36, 8.37, 8.38, 8.39, 8.4, 8.41, 8.42, 8.43, 8.44, 8.45, 8.459999999999999, 8.47, 8.48, 8.49, 8.5, 8.51, 8.52, 8.53, 8.54, 8.55, 8.56, 8.57, 8.58, 8.59, 8.6, 8.61, 8.62, 8.63, 8.64, 8.65, 8.66, 8.67, 8.68, 8.69, 8.7, 8.71, 8.72, 8.73, 8.74, 8.75, 8.76, 8.77, 8.78, 8.79, 8.8, 8.81, 8.82, 8.83, 8.84, 8.85, 8.86, 8.87, 8.88, 8.89, 8.9, 8.91, 8.92, 8.93, 8.94, 8.95, 8.96, 8.97, 8.98, 8.99, 9.0, 9.01, 9.02, 9.03, 9.04, 9.05, 9.06, 9.07, 9.08, 9.09, 9.1, 9.11, 9.12, 9.13, 9.14, 9.15, 9.16, 9.17, 9.18, 9.19, 9.2, 9.21, 9.22, 9.23, 9.24, 9.25, 9.26, 9.27, 9.28, 9.29, 9.3, 9.31, 9.32, 9.33, 9.34, 9.35, 9.36, 9.37, 9.38, 9.39, 9.4, 9.41, 9.42, 9.43, 9.44, 9.45, 9.46, 9.47, 9.48, 9.49, 9.5, 9.51, 9.52, 9.53, 9.54, 9.55, 9.56, 9.57, 9.58, 9.59, 9.6, 9.61, 9.62, 9.63, 9.64, 9.65, 9.66, 9.67, 9.68, 9.69, 9.7, 9.71, 9.72, 9.73, 9.74, 9.75, 9.76, 9.77, 9.78, 9.79, 9.8, 9.81, 9.82, 9.83, 9.84, 9.85, 9.86, 9.87, 9.88, 9.89, 9.9, 9.91, 9.92, 9.93, 9.94, 9.95, 9.96, 9.97, 9.98, 9.99, 10.0]
y= [0, 0.009999000099989653, 0.019992003198718838, 0.02997302427814569, 0.03993610254790441, 0.0498753123387846, 0.05978477572905524, 0.06965867371561538, 0.07949125745609083, 0.08927685922175287, 0.09900990301842155, 0.10868491483471343, 0.11829653247957332, 0.1278395149738311, 0.13730875146352683, 0.14669926962592233, 0.15600624354243123, 0.16522500101612245, 0.17435103031494942, 0.18337998632539762, 0.19230769610479503, 0.20113016382406085, 0.20984357509615048, 0.2184443006888586, 0.22692889962394217, 0.23529412166769842, 0.24353690922115487, 0.25165439862088274, 0.25964392086411364, 0.26750300177430814, 0.2752293616255837, 0.28282091424644834, 0.29027576562510066, 0.29759221204014236, 0.304768737741906, 0.31180401221072856, 0.3186968870194059, 0.32544639232774686, 0.3320517330376204, 0.3385122846371582, 0.3448275887628524, 0.35099734850818537, 0.35702142350715327, 0.3628998248206168, 0.3686327096528387, 0.3742203759248669, 0.37966325673060447, 0.3849619147004898 5, 0.39011703629670474, 0.3951294260627487, 0.4000000008490777, 0.40472978403531673, 0.4093198997683307, 0.4137715672341894, 0.4180860949807964, 0.4222648753066859, 0.4263093787302244, 0.4302211485522064, 0.43400179552359847, 0.4376529926289857, 0.4411764699950997, 0.4445740099326754, 0.4478474421187908, 0.4509986389257977, 0.4540295109019522, 0.4569420024079072, 0.4597380874123332, 0.4624197654490902, 0.46498905773758736, 0.46744800346723103, 0.4697986562461815, 0.472043080714012, 0.474183349317288, 0.476221539246559, 0.4781597295327827, 0.479999998300771, 0.48174442017686875, 0.4833950638477359, 0.48495398976681003, 0.48642324800476866, 0.4878048762400925, 0.48910089788564703, 0.49031332034704883, 0.4914441334084629, 0.4924953077413861, 0.49346879353190604, 0.4943665192218829, 0.49519039035948426, 0.4959422885545024, 0.49662407053390 4, 0.49723756729309754, 0.4977845833384544, 0.4982668960166829, 0.4986862549267309, 0.49904438140997615, 0.49934296811455864, 0.4995836786298117, 0.4997681471868535, 0.4998979784215171, 0.49997474719591195, 0.4999999984750304, 0.4999752472549381, 0.49990197853920776, 0.499781647360385, 0.4996156788433982, 0.4994054683079515, 0.4991523814070638, 0.49885775429904056, 0.4985228938502876, 0.4981490778664955, 0.4977375553498411, 0.49728954677996834, 0.49680624441662197, 0.4962888126219189, 0.495738388200347, 0.4951560807546851, 0.49454297305613776, 0.49390012142707546, 0.493228556134863, 0.49252928179535005, 0.49180327778468313, 0.4910514986581817, 0.4902748745751002, 0.48947431172817446, 0.4886506927769236, 0.4878048772837472, 0.486937702151926, 0.4860499820646944, 0.4851425099246165, 0.4842160572925521, 0.4832713748255549, 0.4823091927130948, 0.4813302211110482, 0.48033515057294157, 0.4793246524779823, 0.47829937945544776, 0.47725996580504515, 0.4762070279128896, 0.47514116466278344, 0.47406295784251135, 0.4729729725448967, 0.4718717575633928, 0.47075984578200974, 0.4696377545594022, 0.46850598610696675,

0.4673650278608198, 0.4662153528475469, 0.46505742004363326, 0.46389167472850307, 0.46271854883111124,
0.4615384612700471, 0.46035181828712224, 0.4591590137744291, 0.4579604295948679, 0.45675643589615095,
0.45554739141830314, 0.4543336437946872, 0.45311552984659004, 0.4518933758714149, 0.4506674979245309,
0.4494382020948375, 0.4482057847741075, 0.44697053292017797, 0.44573272431406147, 0.4444926278110561,
0.44325050358593493, 0.44200660337229963, 0.44076117069618537, 0.4395144411040065, 0.4382666423849353,
0.43701799478780656, 0.43576871123264393, 0.43451899751690287, 0.43326905251652836, 0.43201906838192466,
0.4307692307289347, 0.42951971882492845, 0.42827070577009774, 0.4270223586740562, 0.42577483882784234,
0.42452830187142304, 0.42328289795679463, 0.4220387719067777, 0.42079606336960096, 0.4195549069693694,
0.4183154324525096, 0.4170777648302845, 0.4158420245174709, 0.4146083274672869, 0.4133767853026612 5,
0.4121475054439303, 0.41092059123304947, 0.4096961420544045, 0.40847425345230565, 0.4072550172452468,
0.4060385216370111, 0.40482485132470064, 0.40361408760376977, 0.4024063084701368, 0.40120158871944955,
0.40000000004357783, 0.3988016111244044, 0.3976064877249848, 0.3964146927781451, 0.39522628647258384,
0.39404132633654515, 0.3928598673191266, 0.39168196186928483, 0.3905076600126007, 0.38933700942586336,
0.38817005550953276, 0.38700684145813685, 0.3858474083286601, 0.3846917951069777, 0.38354003877238835,
0.38239217436029793, 0.38124823502310473, 0.38010825208933496, 0.3789722551210777, 0.3778402719697652,
0.3767123288303453, 0.37558845029388915, 0.37446865939867946, 0.3733529776798198, 0.37224142521740744,
0.37113402068330925, 0.3700307813865797, 0.36893172331755936, 0.3678368611906907, 0.3667462084860872,
0.3656597774898911, 0.3645775793334535, 0.3634996240313708, 0.3624259205184087, 0.3613564766853463,
0.36029129941377036, 0.3592303946098495, 0.3581737672371175, 0.35712142134829356, 0.35607336011616697,
0.3550295858635735, 0.3539901000924882, 0.35295490351226055, 0.35192399606701663, 0.35089737696225143,
0.349875044690635, 0.34885699705705464, 0.347843231202915, 0.3468337436297175, 0.34582853022193966,
0.34482758626923443, 0.34383090648796866, 0.3428384850421204, 0.34185031556355266, 0.3408663911716824,
0.3398867044925609, 0.3389112476773842, 0.33794001242044774, 0.33697298997656366, 0.3360101711779542,
0.3350515464506372, 0.33409710583031876, 0.3331468389778057, 0.3322007351939538, 0.3312587834341634,
0.33032097232243607, 0.3293872901650057, 0.32845772496355463, 0.3275322644280289, 0.3266108959890622,
0.325693606810021, 0.3247803837986821, 0.3238712136185 5165, 0.3229660826998377, 0.3220649772508537,
0.32116788326448475, 0.3202747865358606, 0.3193856726643537, 0.31850052706680193, 0.3176193349858308,
0.3167420814986608, 0.3158687515256405, 0.3149993298385131, 0.31413380106842403, 0.3132721497136774,
0.3124143601472481, 0.3115604166240575, 0.3107103032880183, 0.30986400417885673, 0.3090215032387174,
0.3081827843185575, 0.3073478311843367, 0.30651662752300823, 0.3056891569483177, 0.30486540300641385,
0.3040453491812784, 0.3032289788999789, 0.3024162755377506, 0.30160722242291166, 0.3008018028416173,
0.3000000000424566, 0.29920179724089774, 0.29840717762358493, 0.29761612435249213, 0.29682862056893755,
0.2960446493974628, 0.2952641939495809, 0.29448723732739696, 0.29371376262710486, 0.2929437529423643,
0.29217719136756104, 0.29141406100095407, 0.2906543449477131, 0.2898980263228493, 0.2891450882540428,
0.28839551388436957, 0.28764928637493103, 0.28690638890738906, 0.28616680468640915, 0.28543051694201454,
0.2846975089318539, 0.28396776394338535, 0.28324126529597843, 0.282517996342938, 0.28179794047345064,
0.2810810811144573, 0.2803674017324534, 0.2796568858352192, 0.278949516973482, 0.2782452787425128, 0.27754415478365835,
0.2768461287858121, 0.27615118448682435, 0.27545930567485416, 0.27477047618966494, 0.27408467992386476,
0.27340190082409377, 0.2727221228921597, 0.2720453301861236, 0.2713715068213367, 0.2707006369714305,
0.27003270486926095, 0.26936769480780876, 0.2687055911410362, 0.268046378284703, 0.26739004071714145,
0.26673656297999265, 0.2660859296789048, 0.26543812548419493, 0.2647931351314748, 0.26415094342224266,
0.26351153522444143, 0.2628748954729846, 0.2622410091702506, 0.2616098613865473, 0.26098143726054646,
0.2603557219996904, 0.25973270088057054, 0.2591123592492795, 0.25849468252173735, 0.25787965618399283,
0.2572672657925003, 0.25665749697437334, 0.2560503354276156, 0.25544576692132975, 0.2548437772959052,
0.25424435246318516, 0.2536474784066141, 0.25305314118136585, 0.25246132691445294, 0.2518720218048183,
0.25128521212340965, 0.2507008842132365, 0.25011902448941187, 0.24953961943917755, 0.2489626556219147,
0.24838811966913962, 0.24781599828448564, 0.24724627824367107, 0.2466789463944543, 0.24611398965657616,
0.24555139502168988, 0.24499114955327944, 0.24443324038656655, 0.24387765472840645, 0.24332437985717342,
0.24277340312263596, 0.24222471194582207, 0.24167829381887537, 0.24113413630490185, 0.240592227037808,
0.24005255372213052, 0.23951510413285798, 0.23897986611152446, 0.23844682758461663, 0.23791597652617155,
0.23738730099477034, 0.23686078911472327, 0.23633642907956912, 0.23581420915184856, 0.23529411766287153,
0.23477614301247918, 0.23426027366880037, 0.23374649816800308, 0.23323480511404104, 0.23272518317839555,
0.23221762109981298, 0.23171210768403794, 0.2312086318035425, 0.23070718239725144, 0.23020774847026393,
0.2297103190935717, 0.22921488340377394, 0.22872143060278902, 0.22822994995756324, 0.22774043079977685,
0.22725286252554736, 0.22676723459513032, 0.22628353653261787, 0.22580175792563498, 0.22532188842503362,
0.224843917744585, 0.22436783566067015, 0.22389363201196846, 0.2234212966991451, 0.22295081968453662,
0.22248219099183547, 0.22201540070577316, 0.22155043897180238, 0.22108729599577798, 0.22062596204363724,
0.2201664274410792, 0.21970868257324322, 0.21925271788438702, 0.21879852387775641, 0.21834609111430073,
0.21789541021427256, 0.21744647185498092, 0.21699926677142894, 0.2165537857557974, 0.21611001965712068,
0.2156679593809625, 0.21522759588909193, 0.21478892019915938, 0.21435192338437273, 0.21391659657317386,
0.21348293094891543, 0.2130509177495379, 0.21262054826724713, 0.21219181384819233, 0.2117647058921445,
0.2113392158521756, 0.21091533523433814, 0.21049305559734555, 0.2100723685522531, 0.20965326576213977,
0.20923573894179068, 0.20881977985738048, 0.20840538032615752, 0.20799253221612898, 0.20758122744574684,
0.20717145798359476, 0.2067632158480762, 0.20635649310710322, 0.20595128187778655, 0.2055475743261267,
0.20514536266670605, 0.20474463916238222, 0.20434539612398253, 0.20394762590999954, 0.20355132092628797,
0.2031564736257626, 0.20276307650809772, 0.20237112211942748, 0.2019806030520478, 0.20159151194411934,
0.20120384147937206, 0.2008175843868108, 0.20043273344042237, 0.200049281458884, 0.19966722130527312,
0.19928654588677838, 0.19890724815441238, 0.1985293211027255, 0.19815275776952132, 0.1977775512355733,
0.1974036946243431, 0.19703118110170018, 0.19666000387564295, 0.19629015619602122, 0.19592163135426033,
0.1955544226830866, 0.19518852355625424, 0.1948239273882739, 0.19446062763414249, 0.19409861778907467,
0.19373789138823574, 0.19337844200647605, 0.1930202632580669, 0.19266334879643796, 0.19230769231391617,
0.19195328754146623, 0.19160012824843248, 0.19124820824228228, 0.1908975213683511, 0.19054806150958886,
0.19019982258630802, 0.18985279855593293, 0.18950698341275096, 0.18916237118766488, 0.18881895594794698,
0.1884767317969945, 0.18813569287408674, 0.18779583335414346, 0.18745714744748507, 0.18711962939959398,
0.18678327349087778, 0.18644807403643368, 0.18611402538581445, 0.18578112192279606, 0.18544935806514648,
0.18511872826439632, 0.18478922700561057, 0.1844608488071621, 0.18413358822050654, 0.1838074398299586,

0.18348239825246973, 0.18315845813740755, 0.18283561416633645, 0.18251386105279968, 0.18219319354210298,
0.18187360641109956, 0.18155509446797655, 0.18123765255204283, 0.18092127553351828, 0.18060595831332452,
0.18029169582287696, 0.17997848302387828, 0.17966631490811333, 0.17935518649724536, 0.1790450928426137,
0.17873602902503277, 0.1784279901545925, 0.17812097137045999, 0.17781496784068276, 0.17750997476199307,
0.17720598735961376, 0.17690300088706545, 0.17660101062597494, 0.17630001188588498, 0.1760000000040654,
0.17570097034532559, 0.17540291830182803, 0.1751058392929035, 0.17480972876486725, 0.1745145821908366,
0.17422039507054982, 0.17392716293018617, 0.1736348813221874, 0.17334354582508024, 0.17305315204330038,
0.17276369560701751, 0.17247517217196173, 0.17218757741925106, 0.1719009070552202, 0.17161515681125064,
0.17133032244360172, 0.17104639973324307, 0.17076338448568823, 0.1704812725308294, 0.17020005972277327,
0.16991974193967824, 0.16964031508359262, 0.16936177508029404, 0.16908411787913, 0.16880733945285958,
0.16853143579749627, 0.16825640293215186, 0.16798223689888153, 0.16770893376253004, 0.16743648961057894,
0.16716490055299496, 0.1668941627220795, 0.16662427227231905, 0.16635522538023695, 0.1660870182442459,
0.16581964708450178, 0.16555310814275837, 0.16528739768222322, 0.16502251198741452, 0.16475844736401887,
0.16449520013875035, 0.16423276665921036, 0.16397114329374854, 0.16371032643132477, 0.16345031248137207,
0.16319109787366054, 0.16293267905816225, 0.16267505250491715, 0.1624182147038999, 0.16216216216488763,
0.16190689141732886, 0.16165239901021305, 0.16139868151194137, 0.16114573551019817, 0.1608935576118237,
0.16064214444268732, 0.16039149264756203, 0.16014159888999968, 0.15989245985220712, 0.15964407223492327,
0.15939643275729712, 0.15914953815676652, 0.1589033851889379, 0.15865797062746687, 0.15841329126393963,
0.1581693439077554, 0.15792612538600942, 0.15768363254337703, 0.15744186224199855, 0.15720081136136488,
0.15696047679820405, 0.15672085546636852, 0.15648194429672338, 0.15624374023703524, 0.156006240251862,
0.15576944132244339, 0.15553334044659234, 0.1552979346385871, 0.15506322092906416, 0.15482919636491188,
0.154595858009165, 0.1543632029408998, 0.15413122825513018, 0.1538999310627042, 0.15366930849020172,
0.15343935767983252, 0.15321007578933526, 0.15298145999187718, 0.15275350747595443, 0.1525262154452933,
0.1522995811187519, 0.1520736017302228, 0.1518482745285363, 0.15162359677736423, 0.1513995657551247,
0.15117617875488737, 0.15095343308427941, 0.15073132606539227, 0.15050985503468894, 0.15028901734291197,
0.1500688103549922, 0.14984923144995801, 0.14963027802084536, 0.14941194747460843, 0.1491942372320308,
0.1489771447276374, 0.14876066740960714, 0.14854480273968595, 0.1483295481931006, 0.14811490125847313,
0.14790085943773584, 0.14768742024604697, 0.14747458121170684, 0.14726233987607476, 0.1470506937934864,
0.14683964053117188, 0.14662917766917427, 0.14641930280026882, 0.14621001352988278, 0.1460013074760156,
0.14579318226915997, 0.14558563555222312, 0.145378664980449, 0.14517226822134077, 0.14496644295458397,
0.14476118687197012, 0.14455649767732107, 0.1443523730864136, 0.14414881082690487, 0.1439458086382581,
0.14374336427166898, 0.14354147548999255, 0.1433401400676706, 0.14313935579065948, 0.14293912045635865,
0.1427394318735395, 0.1425402878622748, 0.14234168625386862, 0.14214362489078672, 0.14194610162658755,
0.14174911432585352, 0.14155266086412296, 0.14135673912782246, 0.14116134701419966, 0.14096648243125665,
0.14077214329768367, 0.14057832754279337, 0.14038503310645556, 0.14019225793903228, 0.1400000000013135,
0.13980825726445315, 0.13961702770990567, 0.13942630932936295, 0.13923610012469173, 0.13904639810787145,
0.1388572013009325, 0.13866850773589498, 0.13848031545470782, 0.1382926225091883, 0.1381054269609621,
0.1379187268814036, 0.13773252035157685, 0.13754680546217662, 0.13736158031347015, 0.13717684301523916,
0.13699259168672234, 0.13680882445655812, 0.13662553946272793, 0.1364427348524999, 0.13626040878237275,
0.13607855941802038, 0.13589718493423655, 0.13571628351488008, 0.13553585335282037, 0.1353558926498834,
0.13517639961679806, 0.13499737247314275, 0.13481880944729244, 0.1346407087763661, 0.1344630687061745,
0.13428588749116824, 0.13410916339438636, 0.13393289468740513, 0.1337570796502872, 0.13358171657153117,
0.13340680374802155, 0.13323233948497884, 0.13305832209591023, 0.13288474990256047, 0.13271162123486305,
0.13253893443089185, 0.13236668783681305, 0.1321948798068373, 0.13202350870317237, 0.13185257289597593,
0.13168207076330882, 0.13151200069108857, 0.13134236107304323, 0.1311731503106654 8, 0.13100436681316718,
0.13083600899743403, 0.13066807528798072, 0.1305005641169063, 0.13033347392384986, 0.13016680315594653,
0.13000055026778368, 0.1298347137213576, 0.1296692919860303, 0.12950428353848667, 0.12933968686269198,
0.12917550044984955, 0.1290117227983588, 0.12884835241377354, 0.12868538780876054, 0.12852282750305843,
0.12836067002343673, 0.1281989139036554, 0.12803755768442437, 0.12787659991336361, 0.12771603914496327,
0.12755587394054418, 0.1273961028682186, 0.12723672450285123, 0.12707773742602047, 0.12691914022597992,
0.1267609314976202, 0.12660310984243092, 0.12644567386846312, 0.12628862219029155, 0.1261319534289777,
0.1259756662120327, 0.12581975917338062, 0.12566423095332196, 0.1255090801984975, 0.12535430556185218,
0.12519990570259942, 0.1250458792861856, 0.1248922249842547, 0.12473894147461327, 0.1245860274411956,
0.12443348157402911, 0.12428130256920003, 0.12412948912881915, 0.12397803996098795, 0.12382695377976495,
0.12367622930513211, 0.12352586526296166, 0.12337586038498304, 0.12322621340875, 0.12307692307760808,
0.12292798814066216, 0.12277940735274427, 0.1226311794743816, 0.12248330327176472, 0.12233577751671608,
0.12218860098665854, 0.12204177246458428, 0.12189529073902385, 0.12174915460401534, 0.12160336285907392,
0.1214579143091614, 0.12131280776465614, 0.12116804204132299, 0.12102361596028358, 0.12087952834798672,
0.12073577803617902, 0.12059236386187566, 0.12044928466733139, 0.12030653930001167, 0.12016412661256409,
0.12002204546278986, 0.11988029471361551, 0.11973887323306488, 0.11959777989423111, 0.11945701357524893,
0.11931657315926715, 0.11917645753442119, 0.11903666559380593, 0.11889719623544866, 0.11875804836228218,
0.11861922088211817, 0.11848071270762059, 0.11834252275627938, 0.11820464995038427, 0.11806709321699871,
0.11792985148793407, 0.1177929236997239, 0.11765630879359842, 0.11752000571545917, 0.11738401341585376,
0.11724833084995083, 0.11711295697751517, 0.11697789076288298, 0.11684313117493729, 0.11670867718708353,
0.11657452777722527, 0.11644068192774007, 0.11630713862545555, 0.11617389686162556, 0.11604095563190649,
0.11590831393633377, 0.1157759707792985, 0.11564392516952422, 0.11551217612004379, 0.11538072264817652,
0.11524956377550533, 0.11511869852785409, 0.11498812593526514, 0.1148578450319769, 0.11472785485640162,
0.11459815445110333, 0.11446874286277582, 0.11433961914222088, 0.11421078234432656, 0.11408223152804568,
0.11395396575637433, 0.11382598409633064, 0.11369828561893361, 0.11357086939918208, 0.11344373451603385,
0.11331688005238487, 0.11319030509504865, 0.11306400873473574, 0.11293799006603329, 0.11281224818738488,
0.11268678220107028, 0.11256159121318554, 0.11243667433362302, 0.11231203067605162, 0.11218765935789721,
0.11206355950032298, 0.11193973022821017, 0.11181617067013866, 0.11169287995836785, 0.11156985722881761,
0.11144710162104932, 0.11132461227824703, 0.11120238834719876, 0.11108042897827793, 0.11095873332542483,
0.11083730054612828, 0.11071612980140735, 0.1105952202557932, 0.11047457107731104, 0.11035418143746221,
0.11023405051120637, 0.11011417747694371, 0.10999456151649743, 0.10987520181509618, 0.10975609756135664,
0.1096372479472663, 0.10951865216816618, 0.10940030942273381, 0.1092822189129662, 0.10916437984416298,

0.10904679142490958, 0.10892945286706061, 0.1088123633857232, 0.10869552219924057, 0.10857892852917563, 0.10846258160029465, 0.10834648064055113, 0.10823062488106969, 0.10811501355613, 0.10799964590315098, 0.1078845211626749, 0.10776963857835171, 0.10765499739692343, 0.10754059686820854, 0.10742643624508662, 0.10731251478348297, 0.10719883174235335, 0.10708538638366881, 0.10697217797240062, 0.10685920577650526, 0.10674646906690953, 0.10663396711749575, 0.106521699205087, 0.10640966460943252, 0.1062978626131931, 0.10618629250192665, 0.10607495356407382, 0.10596384509094363, 0.10585296637669936, 0.10574231671834435, 0.10563189541570792, 0.10552170177143146, 0.1054117350909545, 0.10530199468250093, 0.10519247985706526, 0.10508318992839895, 0.10497412421299687, 0.10486528203008381, 0.10475666270160104, 0.10464826555219299, 0.104540089909194, 0.10443213510261513, 0.10432440046513107, 0.10421688533206706, 0.10410958904138601, 0.10400251093367557, 0.10389565035213534, 0.10378900664256413, 0.10368257915334732, 0.10357636723544424, 0.1034703702423757, 0.10336458753021151, 0.10325901845755812, 0.10315366238554637, 0.10304851867781913, 0.1029435867005193, 0.10283886582227761, 0.10273435541420063, 0.10263005484985885, 0.10252596350527472, 0.10242208075891092, 0.10231840599165855, 0.10221493858682545, 0.10211167793012463, 0.10200862340966264, 0.10190577441592817, 0.10180313034178058, 0.10170069058243854, 0.10159845453546877, 0.1014964216007748, 0.10139459118058575, 0.10129296267944533, 0.10119153550420075, 0.1010903090639917, 0.10098928277023951, 0.10088845603663625, 0.10078782827913395, 0.10068739891593385, 0.10058716736747579, 0.10048713305642751, 0.10038729540767413, 0.10028765384830772, 0.10018820780761675, 0.10008895671707582, 0.09998990001033527, 0.09989103712321093, 0.09979236749367394, 0.0996938905618406, 0.09959560576996225, 0.09949751256241526, 0.09939961038569105, 0.09930189868838615, 0.09920437692119234, 0.09910704453688685, 0.09900990099032254]



11 .4 SOME DIFFERENTIAL EQUATION SOLUTIONS

Few more example will be given in this chapter to understand nature of the real differential equation solutions. Fluid mechanics is a topic that mechanical engineering, chemical engineering and meterology is closely interested. Turbulent flow is one of the important sub topic. In the classical approach to the turbulance random movements are assumed. When it is investigated further it is shown that some diferential equations can be given chaotic results. This created a new science of chaotic systems. Edward Lorents, an athmospheric scientist developed Lorents set of diferential equations to investigate atmospheric turbulance effects. The equation set:

$$\frac{dx_1}{dt} = \sigma(x_2 - x_2)$$

$$\frac{dx_2}{dt} = (1 + \lambda - x_3)x_1 - x_2$$

$$\frac{dx_3}{dt} = x_1 x_2 - \gamma x_2$$

In this equations σ, λ, and γ are pozitif constants. Equation behave chaotically when

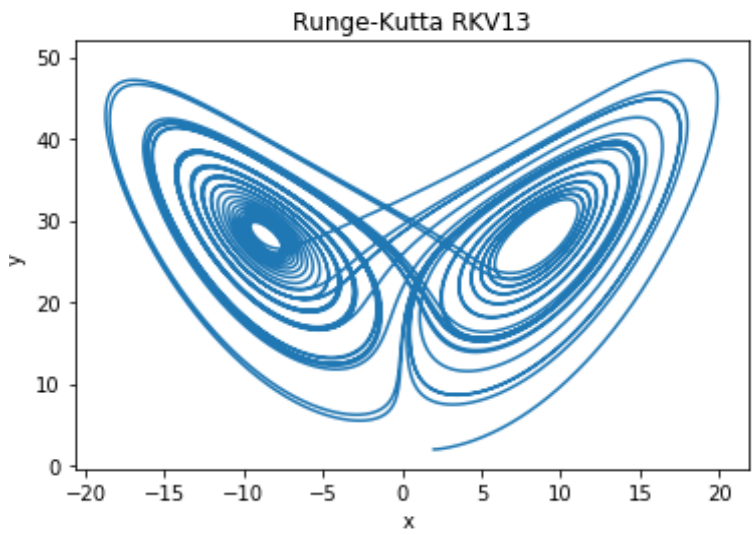
$\sigma > \gamma + 1$ and

$\lambda > \dfrac{(\sigma+1)(\sigma+\gamma+1)}{\sigma-\gamma-1}$. For example with $\sigma = 10$, $\lambda = 24$ ve $\gamma = 2$ system behaves chaotically. If we solve the Lorents equation set with $0\ s \le t \le 30\ s$ time range, $x_1(0) = 1.0$; $x_2(0) = 0.0$; $x_3(0) = 20.0$ conditions

Python equivalent of this solution:

Program 8.7-3A Solution of Lorenz equation with RKV13 python version

```
# -*- coding: utf-8 -*-
"""
Created on Fri Aug 31 13:10:28 2018

@author: Mustafa Turhan Çoban
"""
from Runge_Kutta import *;
from  Butcher import *;
from f_xi import *;
import numpy as np;
import matplotlib.pyplot as plt;

class f1(f_xi):
    def __init__(self):
      self.name="Lorentz equation";
      self.sigma=10.0;
      self.lambd=28.0;
      self.gamma=8.0/3.0;
    def func(self,x,x_ref):
        if x_ref==0: F=self.sigma*(x[2]-x[1]);
        elif x_ref==1: F=(1.0+self.lambd-x[3])*x[1]-x[2];
        elif x_ref==2: F=x[1]*x[2]-self.gamma*x[3];
        else: F=0.0;
        return F;

rk=Runge_Kutta("RKV13");
f=f1();
y=[0 for i in range(3)];
y[0]=2.0;
y[1]=2.0;
y[2]=2.0;
XY=rk.calc_n(f,0.0,40.0,y,0.001,0);
print(XY);
plt.title('Runge-Kutta RKV13')
plt.xlabel('x ')
plt.ylabel('y ');
plt.plot(XY[1],XY[3]);
```



If parameters is changed a completely different kind of behavior can be observed.

The next example is the harmonic motion with damping. The diferential equation is:

$$\frac{d^2y}{dt^2}+b\frac{dy}{dt}+\omega^2 y=0$$

The equation will be solved with initial conditions y(0)=1 $\frac{dy(0)}{dt}=1$ and ω =1 ve b=0.1. When b=0, it is called simple harmonic motion. a periodic motion that is neither driven nor damped. When b=0.1, it is a damped motion. Oscilations will get smaller and stop after a certain time. Converted form of the equation:

$$\frac{dy_1}{dt}=y_2$$

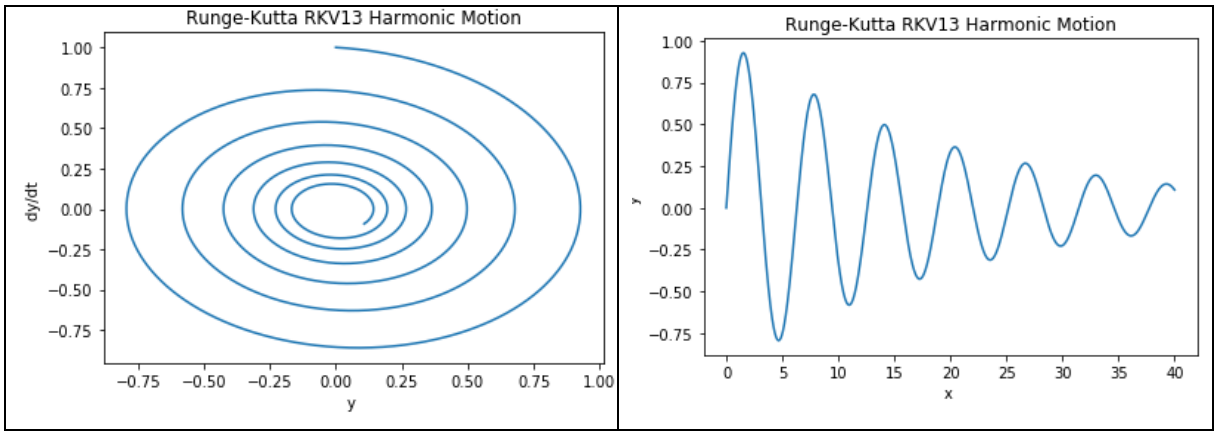
$$\frac{dy_2}{dt}=-by_2-\omega^2 y$$

$y_1(0)=0$

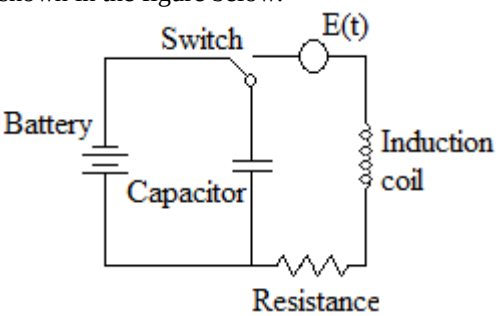
$y_2(0)=1$

Program 8.7-6A Harmonic motion with damping python code

```
# -*- coding: utf-8 -*-
"""
Created on Fri Aug 31 13:10:28 2018

@author: Mustafa Turhan Çoban
"""
from Runge_Kutta import *;
from  Butcher import *;
from f_xi import *;
import numpy as np;
import matplotlib.pyplot as plt;

class f1(f_xi):
   #Harmonic motion
   #d^2y/dt^2+c*dy/dt+w^2y=0
    def __init__(self):
      self.name="Harmonic motion";
      self.b=0.1;
      self.w=1.0;
    def func(self,x,x_ref):
        if x_ref==0: F=x[2];
        elif x_ref==1: F=-self.b*x[2]-self.w*self.w*x[1];
        else: F=0.0;
        return F;

rk=Runge_Kutta("RKV13");
f=f1();
y=[0 for i in range(2)];
y[0]=0.0;
y[1]=1.0;
XY=rk.calc_n(f,0.0,40.0,y,0.001,0);
print(XY);
plt.title('Runge-Kutta RKV13 Harmonic Motion')
plt.xlabel('x ')
plt.ylabel('y ');
plt.plot(XY[0],XY[1]);
```



The next example is the harmonic motion with damping in an electrical circuit. This kind of behavior can be seen frequently in circuits when current is not constant and changed as a function of time. Such a circuit is shown in the figure below.



The differential equation of the system:

$L\frac{d^2q}{dt^2}+R\frac{dq}{dt}+\frac{q}{C}-E(t)=0$ In this equation q is capacitor energy storage capacity (C, Coulomb). L enductance (H, Henry), R res,stance (Ω, Ohm), C capacitance (F), and $E(t)=E_0\sin(wt)$ is time dependent voltage, w frequency and t is time (s) . Current density is the derivative of capacitor energy storage capacity.

$i=\frac{dq}{dt}$ (Ampere). An analytical solution of this equation for R=0 ve $E(t)=E_0\sin(wt)$:

$q(t)=\frac{-E_0}{L(p^2-w^2)}\frac{w}{p}\sin(pt)+\frac{E_0}{L(p^2-w^2)}\sin(wt)$. And the current density

$i=\frac{dq(t)}{dt}=\frac{-E_0w}{L(p^2-w^2)}\sin(pt)+\frac{E_0w}{L(p^2-w^2)}\sin(wt)$. In this equation

$p=\frac{1}{\sqrt{LC}}$ dir.

Now this equation will be solved numerically for initial conditions q(0)=0 ve $i(0)=\frac{dq(0)}{dt}=0$ and physical conditions $w^2=3.5\ 1/s^2$ p=2, L=1 H, E_0=1 V, C=0.25 F. Since analytical solution is known, two solutions can be compared. It is a good exampel to observe the numerical solution error generation due to very complex behaviour of the function.

Converted equation :

$$\frac{dy_1}{dt} = y_2$$

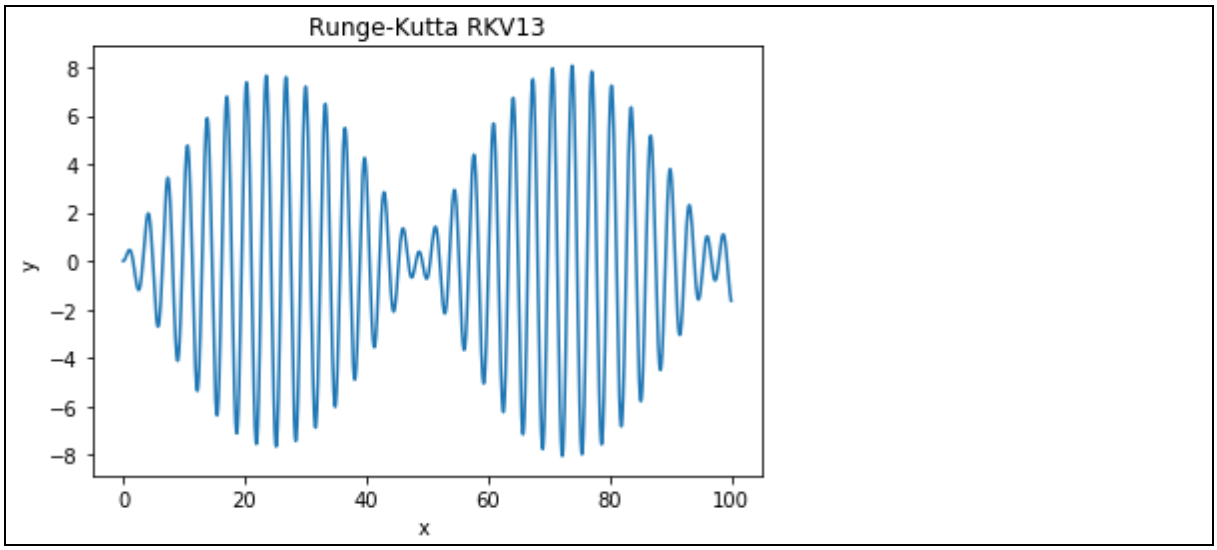
$$\frac{dy_2}{dt} = \frac{\left[-Ry_2 - \frac{y_1}{C} + E_0 \sin(wt) \right]}{L}$$

$y_1(0)=0$

$y_2(0)=0$

Program 8.7-7A Harmonic motion electrical circuit (Changing voltage problem) Python version

```
# -*- coding: utf-8 -*-
"""
Created on Fri Aug 31 13:10:28 2018

@author: Mustafa Turhan Çoban
"""
from Runge_Kutta import *;
from  Butcher import *;
from f_xi import *;
from math import *
import numpy as np;
import matplotlib.pyplot as plt;

class f1(f_xi):
    def __init__(self):
      self.name="Harmonic electronic circuit";
      self.E0=1.0;#Volt
      self.L=1.0;#Henry
      self.C=0.25;#Coulomb
      self.w2=3.5;#s^2
      self.R=0.0;#Ohm
      self.w=sqrt(self.w2);
      self.p=1.0/sqrt(self.L*self.C);
    def func(self,x,x_ref):
        if x_ref==0: F=x[2];
        elif x_ref==1: F=(-self.R*x[2]-x[1]/self.C+self.E0*sin(self.w*x[0]))/self.L;
        else: F=0.0;
        return F;

rk=Runge_Kutta("RKV13");
f=f1();
y=[0 for i in range(2)];
y[0]=0.0;
y[1]=0.0;
XY=rk.calc_n(f,0.0,100.0,y,0.001,0);
print(XY);
plt.title('Runge-Kutta RKV13')
plt.xlabel('x ')
plt.ylabel('y ');
plt.plot(XY[0],XY[2]);
```



Another example. Assume that a bullet is thrown from the gun into an atmosphere. Due to high velocity of the bullet, friction will effect the movement of the bullet. The diferential equation of the motion in cartesian coordinate system can be given as:

$$\frac{d^2x}{dt^2} = -\gamma(\sqrt{v_x^2 + v_y^2})v_x$$

$$\frac{d^2y}{dt^2} = -g - \gamma(\sqrt{v_x^2 + v_y^2})v_y$$

In this equation $\gamma = \frac{k}{m}$,where k friction coefficient between the bullet and air, v_x is x direction component of the velocity, v_y is y direction component of the velocity, m mass, g gravitational acceleration, and t is time.
With
t=x[0] , x=x[1], y=x[2], v_x=x[3], v_y=x[4]
variable convertion the diferential equation becomes

$$\frac{dx[1]}{dx[0]} = x[3]$$

$$\frac{dx[2]}{dx[0]} = x[4]$$

$$\frac{dx[3]}{dx[0]} = -\gamma(\sqrt{x^2[3] + x^2[4]})x[3]$$

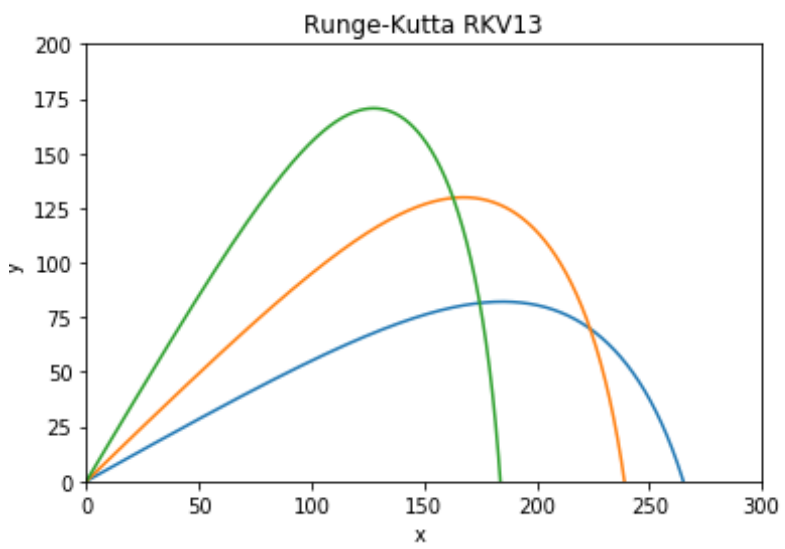
$$\frac{dx[4]}{dx[0]} = -g - \gamma(\sqrt{x^2[3] + x^2[4]})x[4]_z$$

Initial conditions x[0]=0 da x[1]=0 x[2]=0,
V_0=250 m/s x[3]= V_0 sin(θ) m/s, x[4]= V_0 cos(θ) m/s, g=9.806 m/s^2, γ=0.01 , θ =30,45,60 degree

Program 8.7-8A gun bullet trajectory with air friction , Python version

```
# -*- coding: utf-8 -*-
"""
Created on Fri Aug 31 13:10:28 2018

@author: Mustafa Turhan Çoban
"""
from Runge_Kutta import *;
from  Butcher import *;
from f_xi import *;
```

```
from math import *
import numpy as np;
import matplotlib.pyplot as plt;

class f1(f_xi):
    def __init__(self,gammai):
      self.name="Bullet trajectory";
      self.gamma=gammai;
      self.g=9.806;#m/s^2
    def func(self,x,x_ref):
        if x_ref==0: F=x[3];
        elif x_ref==1: F=x[4];
        elif x_ref==2: F=-self.gamma*sqrt(x[3]*x[3]+x[4]*x[4])*x[3];
        elif x_ref==3: F=-self.g - self.gamma*sqrt(x[3]*x[3]+x[4]*x[4])*x[4];
        else: F=0.0;
        return F;

rk=Runge_Kutta("RKV13");
f=f1(0.01);
y=[0 for i in range(4)];
V0=250.0;#m/s
alpha=pi/3.0;
y[0]=0.0;
y[1]=0.0;
y[2]=V0*sin(alpha);
y[3]=V0*cos(alpha);
a=rk.calc_n(f,0.0,20.0,y,0.001,0);
alpha=pi/4.0;
y[2]=V0*sin(alpha);
y[3]=V0*cos(alpha);
b=rk.calc_n(f,0.0,20.0,y,0.001,0);
alpha=pi/6.0;
y[2]=V0*sin(alpha);
y[3]=V0*cos(alpha);
c=rk.calc_n(f,0.0,20.0,y,0.001,0);
plt.title('Runge-Kutta RKV13')
plt.xlabel('x ')
plt.ylabel('y ');
plt.ylim(ymin=0);
plt.ylim(ymax=200);
plt.xlim(xmin=0);
plt.xlim(xmax=300);
plt.plot(a[1],a[2],b[1],b[2],c[1],c[2]);
```



1.0 PROBLEMS

PROBLEM 1

dy/dx = y*sin(x)* sin(x) diferansiyel equation is given. Solution is desired in the x=0 to x=1 range for h=dx=0.2 with the initial conditions of (x=0) y=2. Use fourth order Runge-Kutte method.

PROBLEM 2

$d^2y/dx^2+6\ dy/dx= \sin(\square x)$ diferansiyel equation at x=0 to x=1 range with the boundary conditions of x=0 y=2 and

x=1 de y=-1

Solve diferential equation for h=0.02 step size.

PROBLEM 3

$dy/dx= \sin(\square x)$ diferansiyel equation at x=0 to x=1 range with the boundary conditions of x=0 y=2 and

Solve diferential equation for h=0.02 step size.

PROBLEM 4

Falkner-Scan flow equation is given as

$f''' + f*f'' + \beta(1-f'^2) = 0$

Boundary conditions of the equation

f(0)=0

f'(0)=1

$f'(\infty)=1$

$\beta = 0.3$ ve $\beta = 1.0$

Instead of infinity(∞) , take value of 8.

PROBLEM 5

Solve diferential equation

$y'' = x^2 - y^2 - 3y'$

with initial conditions y(0)=0, y'(0)=0

PROBLEM 6

Solve diferential equation

$y''' = y'^2 - 4y$

with initial conditions y(0)=0, y'(0)=1, y''(0)=0 .

PROBLEM 7

Solve diferential equation

x' = 3x-y

y'=x+y

with initial conditions x(0)=0.2, ve y(0)=0.5 .

PROBLEM 8

Solve diferential equation

$2x''(t)-5x'(t)-3x(t)=45e^{2t}$ with initial conditions x(0)=0 ve x'(0)=1 .

PROBLEM 9

Solve diferential equation

x'(t)=x(t)-x(t)y(t)

y'(t)=-y(t)+x(t)y(t) with initial conditions x(0)=4 ve y(0)=1 at t=[0,8] range.

PROBLEM 10

Solve diferential equation

$x'(t) = \frac{2t}{1+t^2} x'(t) - \frac{2}{1+t^2} x(t) + 1$ with initial conditions x(0)=1.25 ve x(4)=-0.95 at range t=[0,4].

PROBLEM 11

Solve diferential equation

$y'' = -(y')^2 - y + \ln x$

with initial conditions: y(1)=0 y(2)=ln2 at the range 1<=x<=2

PROBLEM 12
For diferential equation

$y' - 2(1+y^2) = 0$ with , $y(0)=0$ starting value and, h=0,25 step size find $y(1)$ değerini with Heun method

PROBLEM 13
Solve diferential equation

$\frac{dy}{dx} = yx^2 - y$ at the range x=0 ve x=2 , with initial condition y(0)=1 .

c) solve analytically from x=0 to x=2.
d) solve numerically by hand with h=0.5 step size with 4th order Runge_Kutte and calculate y(2)
e) solve numerically by computer with h=0.1 step size with 4th order Runge-Kutta
f) Plot to results and compare with analytical solution

PROBLEM 14
Solve diferential equation

$\frac{dy}{dx} = -5y + e^{-2x}$ with initial conditions : x_0=0 y_0=1 . $y(1)$, and with step size h=0.2 by using Simpson 1/3 rule.

PROBLEM 15
Solve diferential equation

$\frac{dy}{dx} = \frac{1}{x+y}$ with initial conditions x_0=0 , y_0=2 , at the interval 0-0.6 with step size h=0.2,

a-) with 4th order Runge-Kutta method
b-) with Euler method

PROBLEM 16
Cooling rate of a substance are given with the following equation

$$\frac{dT}{dt} = -k(T - T_a)$$

In the equation T indicates the temperature of the substance in (°C), T_a is environmental temperature (°C) ,k is a constant (minute^{-1}) dir. Therefore cooling rate is function of temperature difference between body and environment. If a metal ball initially at 90 °C drop into a water bath at T_a=20 °C and if k=0.1 minute^{-1}, Find the time period required ball to reach 30 °C temperature
a) By analytical methods
b) By a numerical method

PROBLEM 17
Solution of diferential equation

$\frac{dy}{dx} = yx^2 - y$ are investigated with range x=0 ve x=2. Initial condition is given as y(0)=1

a) Solve it anaytically Plot the solution.
b) Solve it with Heun method with step size h=0.5 adim
c) Plot the numerical results also and compare it with analytical results

PROBLEM 18
In a chemical reaction a chemical A and a chemical B reacts to give chemical C. Concentration of chemical C as a function of time, y(t) is given with the following equation

$y' = k(a - y)(b - y)$ $\qquad$ $y(0)=0$

In this equation k is a positive constant, a and b is the initial concentration of chemicals A and B. Assume that

k=0.01 , a= 70 milimole/liter , b= 50 milimole/liter .

By using 4th order Runge-Kutta method with step size h=0.5 estimate concentration at t=1 second.

PROBLEM 19

Solve differential equation

$$\frac{dy}{dx} = yx^2 - y$$

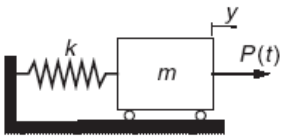
With range of x=0 ve x=2. Initial value is given as y(0)=1 .

a) Solve the problem analytically

b) Solve it with Heun method, step size h=0.5

c) Plot both results and compare relative error.

PROBLEM 20



In the spring-mass system shown in the figure force P(t) is equal to

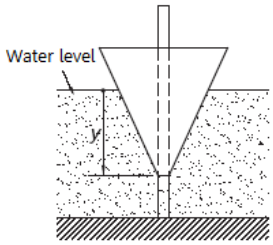
$$P(t) = \begin{cases} 10t \quad N & t < 2s \\ 20\,N & t \geq 2s \end{cases}$$

Diferential equation of the movement

$\frac{d^2y}{dt^2} = \frac{P(t)}{m} - \frac{k}{m}y$ with m=2.5 kg, k=75 N/m. Determine amplitude (maximum amount of movement) of the movement.

PROBLEM 21 A cone buoy can slide freely on a bar is floating on the surface. If cone is disturbed by any effect, it will oscilate. The diferential equation of the oscilation is given as:

$\frac{d^2y}{dt^2} = g(1 - ay^3)$ If a=16 m^{-3}, g=9.80665 m/s^2 and cone position initially moved upward to y=0.1 m and released, find the period(s) and amplitude(m) of the oscilation.



PROBLEM 23

Gun problem with air friction was given in the examples. Solve the same problem with 6th order Runge-Kutta method:

Nozle exit velocity 250 m/s, gravitational acceleration 9.806 m/s^2, friction coefficient $\gamma = (k / m) = 0.01$.

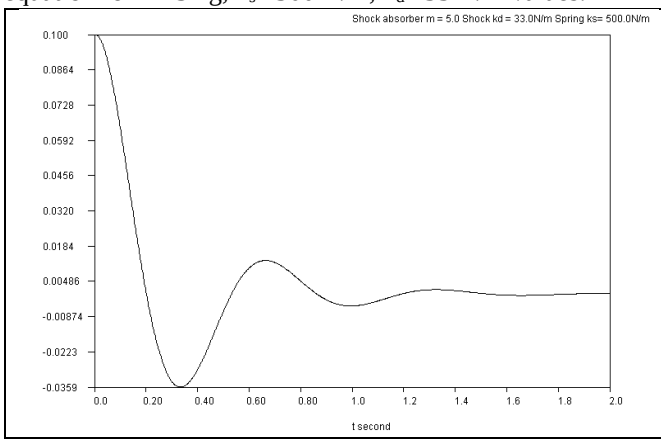
A target at 750 m distance will be hit. Determine the angle θ.

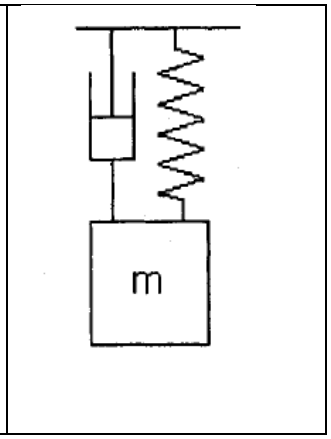
PROBLEM 24

In cars and other machines for vibration control mechanical system with a spring and a shock absorber is used. The differential equation of such a system is given as:

$$m\frac{d^2x}{dt^2}+k_d\frac{dx}{dt}+k_s x=0$$

In this equation m is mass, k_d shock absorber coefficient, k_s is spring coefficient. Solve the diferential equation for m=5 kg, k_s =500 N/m, k_d =33 N/m values.





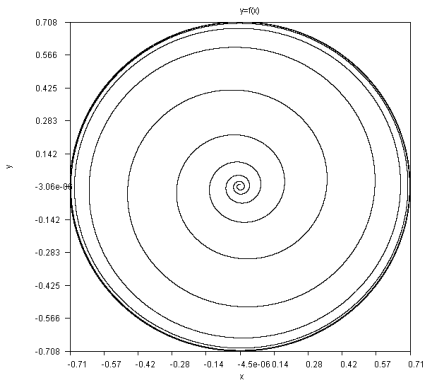
PROBLEM 25

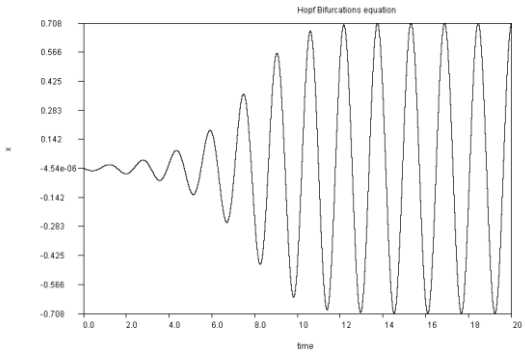
Hopf Bifurcations equation is given by

$$\frac{dx}{dt}=-(g+x^2+y^2)x-wy$$

$$\frac{dy}{dt}=-(g+x^2+y^2)y+wx$$

Solve the set of differential equations with g=-0.5,w=4, with the initial conditions x=0,y=0.01 at t=0. T_{final}=20 s





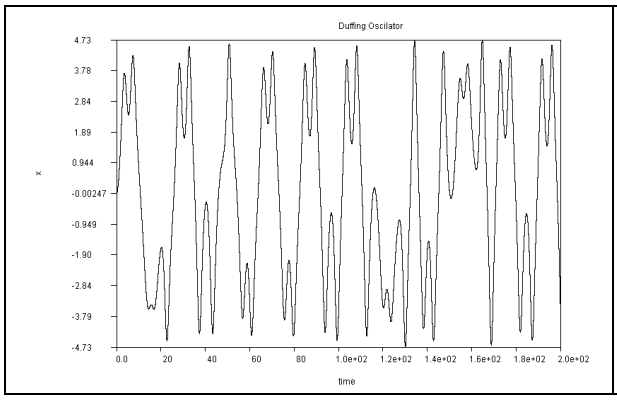
PROBLEM 26

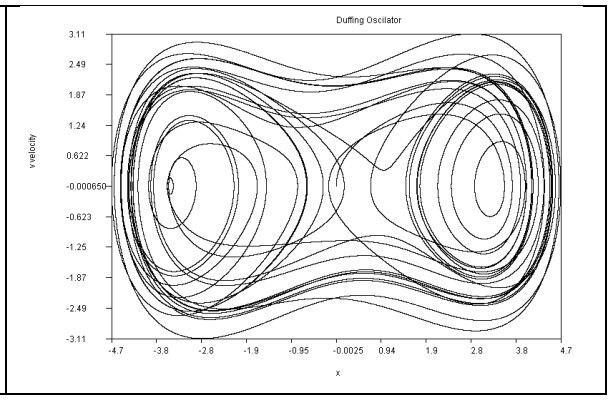
The Duffing Oscilator differential equations are given as

$$\frac{dx}{dt}=v$$

$$\frac{d^2x}{dt^2}=\frac{dv}{dx}=-0.2v+x-0.1x^3+\cos(t)$$

Solve the differential equation set with the initial conditions of x=v=0 at t=0 for 0 to t=200 s period.



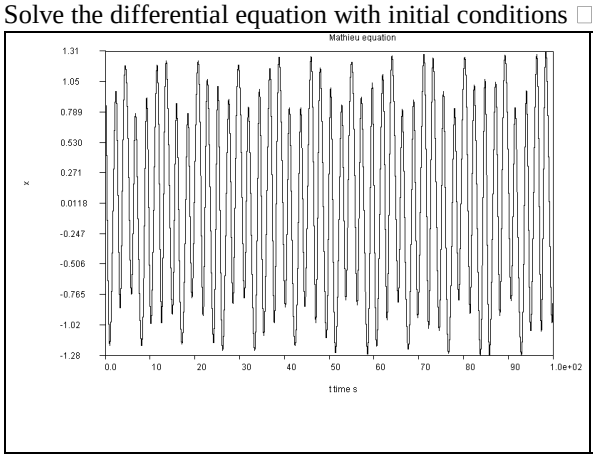


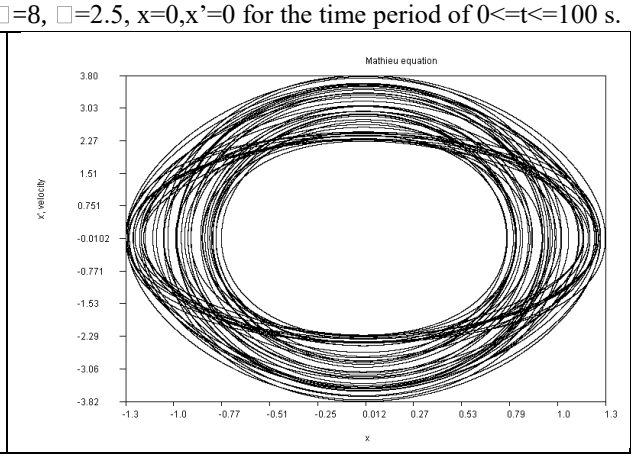
PROBLEM 27

The Mathieu equation is given as

$$\frac{d^2x}{dt^2} + (\delta + 2\varepsilon\cos(2t))x = 0$$

Solve the differential equation with initial conditions □=8, □=2.5, x=0,x'=0 for the time period of 0<=t<=100 s.





PROBLEM 28

H'enon-Heiles equation is given as

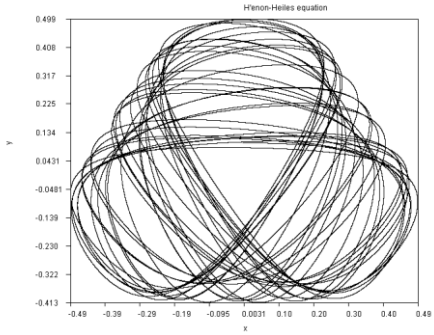
$$\frac{dx}{dt} = P_x$$

$$\frac{dy}{dt} = P_y$$

$$\frac{dP_x}{dt} = -x - 2xy$$

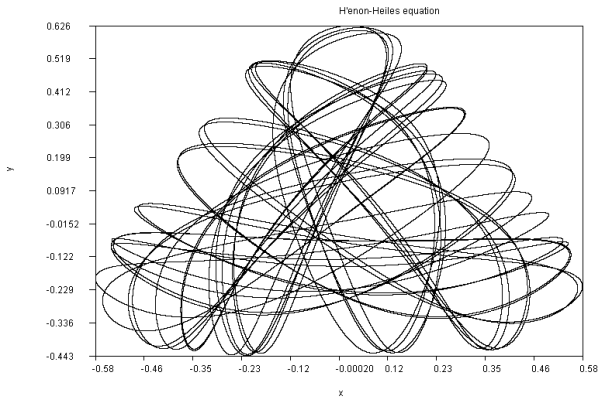
$$\frac{dP_y}{dt} = -y - x^2 + y^2$$

Solve the differential equation with initial conditions x=0,y=0.1 and P_x=0.49057, P_y=0 for the time period of 0<=t<=200 s.



PROBLEM 29

H'enon-Heiles equation is described in Problem 28. Solve the equation with the initial conditions of x=0,y=0 and P_x= 0.35355, P_y= 0.35355 for the time period of 0<=t<=200 s.



PROBLEM 30

Intermittency in Josephson Junctions: The study of the chaotic behavior of (quantum) Josephson junctions is of much fundamental and even practical interest. Written in dimensionless form, the differential equation for the quantum phase difference, φ, across the junction is given by

$$\frac{d^2\Phi}{dt^2} + \frac{1}{\sqrt{\beta_c}}\frac{d\Phi}{dt} + \sin\Phi = A\sin\Omega t$$

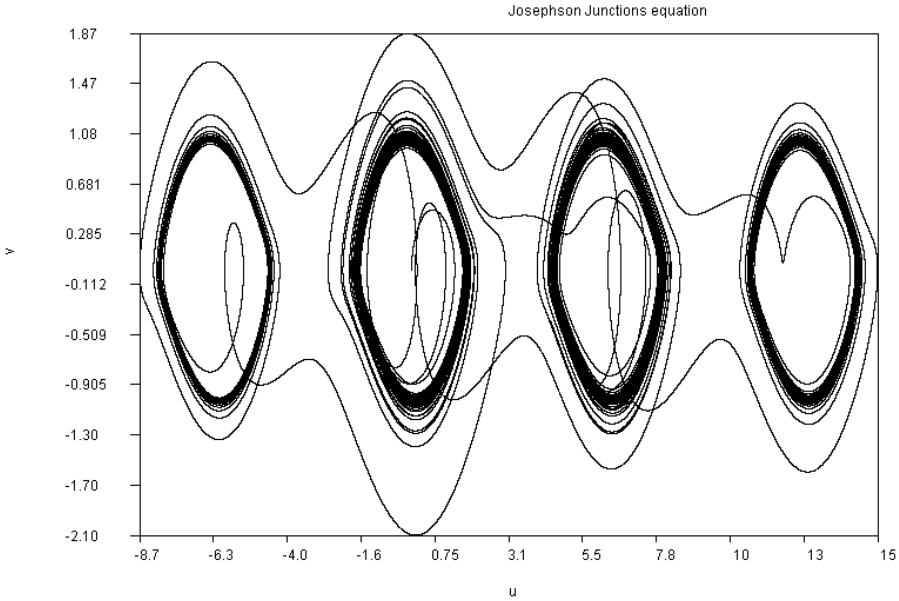
where β_c is the so-called McCumber parameter and Ω is the (normalized)

angular frequency of the driving current. $u = v = \Phi$ and $v = \frac{d\Phi}{dt}$, this can be written as

$$\frac{du}{dt} = v$$

$$\frac{dv}{dt} = \mathrm{A}\sin(\Omega\mathrm{t}) - \mathrm{r}v - \sin(\mathrm{u})$$

For differential equation solution parametric values will be taken as A=0.9045, r=0.5, Ω=0.47. Solve the differential equation for the initial conditions u=v=0 for 0<=t<=1200 s



PROBLEM 31

Planar motion of a three body celestial system can be expressed by the following equations:

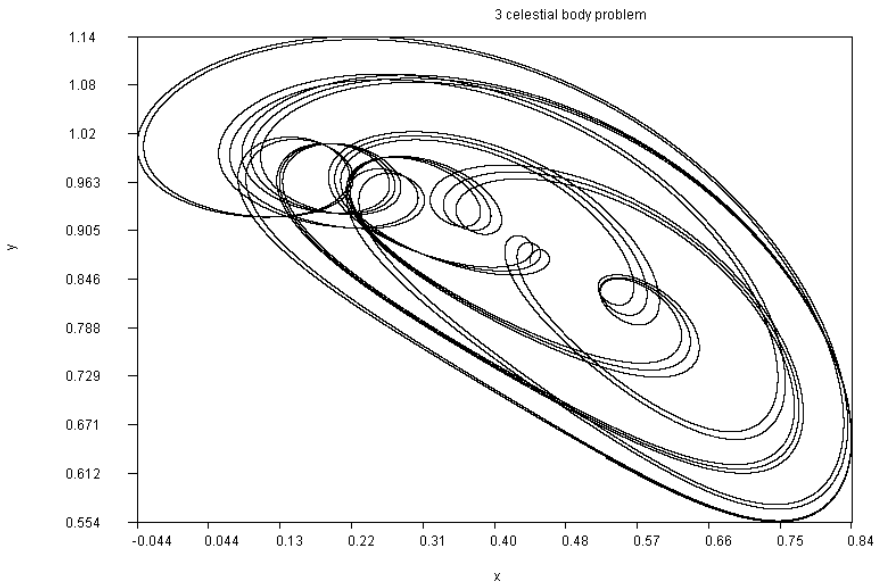
$$\frac{dx}{dt} = p + y$$

$$\frac{dy}{dt} = q - x$$

$$\frac{dp}{dt} = q - \frac{m(x+n)}{\left[(x+n)^2 + y^2\right]^{3/2}} - \frac{n(x-m)}{\left[(x+n)^2 + y^2\right]^{3/2}}$$

$$\frac{dq}{dt} = -p - \frac{my}{\left[(x+n)^2 + y^2\right]^{3/2}} - \frac{ny}{\left[(x+n)^2 + y^2\right]^{3/2}}$$

where the *p* and *q* are the canonical momenta of the coordinates *x* and *y*,
respectively. There are five equilibrium positions, called Lagrangian positions. One of the Lagrangian position is , L4, located at $(x_{average}, y_{average})$ = (0.48, 0.866). In order to solve the diferential equation of the motion initial starting point will be taken as: (x,y,p,q)={0.44,0.866,-0.866,0.48} as initial conditions. Solve the differential equations for 0<t<200. Take m=0.98 n=0.02



PROBLEM 32 For Falkner–Skan solution transformation for fluid laminar free convection, we get the following variables for describing expressions for the stream function ψ and dimensionless coordinate variable η:

$$\eta = \left(\frac{1}{4}Gr\right)^{1/4}\frac{y}{x} \qquad \psi = 4\nu\left(\frac{1}{4}Gr\right)^{1/4} f(\eta) \quad \textit{where}$$

$$Gr = \frac{g(\rho_\infty / \rho_w - 1)x^3}{\nu^2}$$

Velocity profile for the free convection boundary layer:

$$w_x = \frac{\partial\psi}{dy} = \frac{4\nu}{x}\left(\frac{1}{4}Gr\right)^{1/2} f'(\eta) \quad w_y = -\frac{\partial\psi}{dx} = \frac{\nu}{x}\left(\frac{1}{4}Gr\right)^{1/4}\left[\eta f'(\eta) - 3f(\eta)\right]$$

Where x is vertical dimension, y is horizontal dimension, Gr is Grashoff number, w_x and w_y are x and y are dimension velocities. □□is the stream function.

$\theta(\eta) = \dfrac{t - t_\infty}{t_w - t_\infty}$ is the dimensionless temperature. Under these condition falkner-skan solution is converted to the following differential equation set:

$$f'''(\eta) + 3f(\eta)f''(\eta) - 2(f'(\eta))^2 + \theta(\eta) = 0$$

$$\theta''(\eta) + 3\Pr f(\eta)\theta'(\eta) = 0$$

Where Pr is the Prandtl number. *The boundary conditions:*

$\eta = 0: \quad f(\eta) = f'(\eta) = 0 \quad \theta(\eta) = 1$

$\eta \to \infty: \quad f'(\eta) = 0 \quad \theta(\eta) = 0$

Solve the differential equation set.

PROBLEM 33 A compound pendulum consisting of a shaft with a center of mass at point G is shown in the

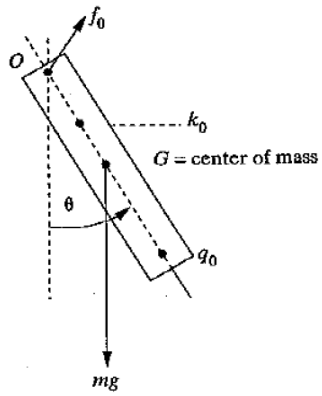


figure. For such a system diferential equation can be given as $J\dfrac{d^2\theta}{dt^2} + mgr\sin(\theta) = 0$. In this equation J is mass momentum of inertia and for the rod given $J = \dfrac{1}{3}ml^2$ and r is the center of gravity so $r = \dfrac{l}{2}$ Differential equation becomes $\dfrac{d^2\theta}{dt^2} + \dfrac{3mg}{2l}\sin(\theta) = 0$. Note that for small values of initial θ value $\sin(\theta)$ approximates to θ so differential equation can be solved analytically. Approximate analytical solution is: $\theta(t) = \theta_0 \cos\left(\left[\sqrt{\dfrac{3mg}{2l}}\right]t\right)$ Solve differential equation numerically and compare the results for big and small initial value of θ

PROBLEM 34 : Solve boundary value equation :

$$\frac{d^2y}{dt^2} = -\left(\frac{dy}{dt}\right)^2 - y + \ln(x)$$ x=1 y=0 and x=2 y=ln(2)

12 .0 EIGENVALUE AND EIGENVECTOR CALCULATIONS

12.1 INTRODUCTION TO EIGENVALUES

Non-zero solution of **[A]{X} = 0** equation is existed, but not necessarily a single solution. A set of {X} vectors can be existed for any given matrix [A]. The equation can also be written as follows for the unit matrix I

[A]{X} = [I]{λ} {X} = 0

[A- λI]{X}= 0

In this equation **λ** is called eigen-value and **{X}** is called eigen-vector. If the equation is written in the open format:

$$\begin{bmatrix} a_{11}-\lambda & a_{12} & a_{13} & \dots & a_{1n} \\ a_{11} & a_{22}-\lambda & a_{23} & \dots & a_{2n} \\ a_{11} & a_{11} & a_{33}-\lambda & \dots & a_{3n} \\ \dots & \dots & \dots & \dots. & \dots. \\ a_{1n} & a_{2n} & a_{3n} & \dots & a_{nn}-\lambda \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ \dots \\ X_5 \end{bmatrix} = 0$$

In this chapter, solution methods of eigenvalues and eigenvectors will be investigated. It should be note that eigenvalues could be complex numbers as well.

12.2 DOMINANT EIGENVALUE CALCULATION (POWER METHOD)

Finding biggest eigenvalue:

Assume that A is n x n matrix with eigenvalues $\lambda_1, \lambda_2, \lambda_3,.., \lambda_n$ with an associated collection of linearly independent eigenvectors $x^1, x^2, x^3,\dots, x^n$. It is also assumed that A has one eigenvalue λ_1 which is largest in magitude so that

$|\lambda_1| > |\lambda_2| > |\lambda_3| > |\lambda_4| \dots > |\lambda_{n-1}| > |\lambda_n|$

It shoud be noted that a matrix A does not have to have linealy independent eigenvectors. When it does not, convergence of the power method may fail. Eigen value with the maximum absolute value is called dominant eigen value. If an eigenvalue is dominant it should be either a real value or it should have an eigen vector consist of real values. If x^k is the eigenvector corresponds to eigenvalue λ_k , The following equations can be written.

$A x^k = \lambda_k x^k$, $1 <= k < n$.

Where { x^1 , x^2 , x^3 , x^4 ,…, x^n } is eigenvector with n components. If A is a symmetric matrix, with n linearly independent eigenvalues. If y^0 is any vector the fact that { x^1 , x^2 , x^3 , x^4 ,…, x^n } is linearly independent implies that constants c_k exist with

$$y^0 = \sum_{k=1}^{n} c_k x^k$$

If a series of y^k vectors are constructed

$y^{k+1} = Ay^k \quad k \geq 0$

$$y^1 = A\left[\sum_{k=1}^{n} c_k x^k\right] = \sum_{k=1}^{n} c_k A x^k = \sum_{k=1}^{n} c_k \lambda_k x^k$$

Generalizing the equation will yield

$$y^i = \sum_{k=1}^{n} c_k \lambda_k^i x^k \quad i \geq 0$$

remembering that λ_1 is the eigenvalue with the largest magnitude. If λ_1^i is taken out of the paranthesis, the equation becomes

$$y^i = \lambda_1^i \left[c_1 x^1 + \sum_{k=1}^{n} c_k \left(\frac{\lambda_k}{\lambda_1}\right)^i x^k\right] \qquad i \geq 0$$

It is known that

$$\left|\frac{\lambda_k}{\lambda_1}\right| < 1$$

Therefore when $i \to \infty$ $\left|\frac{\lambda_k}{\lambda_1}\right|^i \to 0$

$$y^i \cong \lambda_1^i c_1 x^1 , \quad i \gg 1$$

If vector y is normalised :

$$y^{k+1} = \frac{Ay^k}{\|Ay^k\|}$$

Vector y became the eigenvector of the dominant eigenvalue. When the eigenvector is known, eigenvalue can be calculated easily.

$$Ax = \lambda x$$

By multilying both side of te equation with the transpose matrix x^T

$$x^T Ax = \lambda x^T x$$

And

$$\lambda = \frac{x^T Ax}{x^T x}$$

When k is is large, it is known that y^k is approximating to the dominant eigenvalue . In this case

$$\mu_k = \frac{(y^k)^T Ay^k}{(y^k)^T y^k} \qquad k \geq 0$$

It can be written that with interpolation process, μ approximate to the dominant eigenvalue. Deviation of the μ_k from the eigen value is

$$r^k \cong (\mu_k I - A)y^k$$

In this equation when $r^k = 0$ μ_k becomes eigenvalue and y^k becomes eigenvector. In other words if $\|r^k\|$ is small iteration process can be finalize. In the following program computer code from the power method is given

Example : Find the biggest eigenvalue of the following matrix by using power method

$$A = \begin{bmatrix} 1 & 2 \\ 2 & 5 \end{bmatrix}$$

Steps of process is shown in spreadheet solution. Starting vector is taken as y={1,1}

1	2	1
2	5	1

3	3
7	7
	7

1	2	0.428571
2	5	1

2.428571	2.428571
5.857143	5.857143
	5.857143

1	2	0.414634
2	5	1

2.414634	2.414634
5.829268	5.829268
	5.829268

1	2	0.414226
2	5	1

2.414226	2.414226
5.828452	5.828452
	5.828452

1	2	0.414214
2	5	1

2.414214	2.414214
5.828428	5.828428
	5.828428

1	2	0.414214
2	5	1

2.414214	2.414214
5.828427	5.828427

5.828427

λ= 5.828427

Example : Here is a second example:

$$A=\begin{bmatrix} 10 & 4 & -6 \\ 4 & -6 & 8 \\ -6 & 8 & 14 \end{bmatrix}$$ Find the Biggest eigenvalue by power method. Starting vector is again taken as

y={1,1,1}

3.556	-1.778	0	1	1.778	1.778	
-1.778	3.556	-1.778	1	0	0	
0	-1.778	3.556	1	1.778	1.778	
					1.778	
3.556	-1.778	0	1	3.556	3.556	
-1.778	3.556	-1.778	0	-3.556	3.556	
0	-1.778	3.556	1	3.556	3.556	
				3.556	3.556	1
3.556	-1.778	0	1	5.334	5.334	
-1.778	3.556	-1.778	-1	-7.112	7.112	
0	-1.778	3.556	1	5.334	5.334	
				5.334	7.112	1
3.556	-1.778	0	0.75	4.445	4.445	
-1.778	3.556	-1.778	-1	-6.223	6.223	
0	-1.778	3.556	0.75	4.445	4.445	
				4.445	6.223	-0.125
3.556	-1.778	0	0.714286	4.318	4.318	
-1.778	3.556	-1.778	-1	-6.096	6.096	
0	-1.778	3.556	0.714286	4.318	4.318	
				4.318	6.096	-0.020408163
3.556	-1.778	0	0.708333	4.296833	4.296833	
-1.778	3.556	-1.778	-1	-6.07483	6.074833	
0	-1.778	3.556	0.708333	4.296833	4.296833	
				4.296833	6.074833	-0.003472222
3.556	-1.778	0	0.707317	4.29322	4.29322	
-1.778	3.556	-1.778	-1	-6.07122	6.07122	
0	-1.778	3.556	0.707317	4.29322	4.29322	
				4.29322	6.07122	-0.000594884
3.556	-1.778	0	0.707143	4.2926	4.2926	
-1.778	3.556	-1.778	-1	-6.0706	6.0706	
0	-1.778	3.556	0.707143	4.2926	4.2926	
				4.2926	6.0706	-0.000102041
3.556	-1.778	0	0.707113	4.292494	4.292494	
-1.778	3.556	-1.778	-1	-6.07049	6.070494	
0	-1.778	3.556	0.707113	4.292494	4.292494	
				4.292494	6.070494	-1.75067E-05
3.556	-1.778	0	0.707108	4.292475	4.292475	
-1.778	3.556	-1.778	-1	-6.07048	6.070475	
0	-1.778	3.556	0.707108	4.292475	4.292475	
				4.292475	6.070475	-3.00365E-06
3.556	-1.778	0	0.707107	4.292472	4.292472	
-1.778	3.556	-1.778	-1	-6.07047	6.070472	
0	-1.778	3.556	0.707107	4.292472	4.292472	
				4.292472	6.070472	-5.15345E-07
3.556	-1.778	0	0.707107	4.292472	4.292472	
-1.778	3.556	-1.778	-1	-6.07047	6.070472	

0	-1.778	3.556	0.707107

4.292472	4.292472	
4.292472	6.070472	-8.84191E-08

λ= 6.070472

Finding smallest eigenvalue:

If Power method for dominant eigenvalue problem applied in inverse way, smallest magnitude eigenvalue can be obtained.

X 1/X

(X)*(1/X)=1

If biggest lambda(X)

$$A * A^{-1} = I$$

$$y^{k+1} = \frac{Ay^k}{\|Ay^k\|}$$

$$\mu_k = \frac{(y^k)^T A y^k}{(y^k)^T y^k} \quad k \geq 0$$

When k is large **$1/\mu_k$ becomes eigenvalue**.

Example : Find the smalest eigenvalue of $\boldsymbol{A} = \begin{bmatrix} \mathbf{1} & \mathbf{2} \\ \mathbf{2} & \mathbf{5} \end{bmatrix}$ by using inverse power method.

Inverse of the A

$$A^{-1} = \begin{bmatrix} 5 & -2 \\ -2 & 1 \end{bmatrix}$$

5	-2	1
-2	1	1

3	3
-1	1
	3

5	-2	1
-2	1	-0.33333

5.666667	5.666667
-2.33333	2.333333
	5.666667

5	-2	1
-2	1	-0.41176

5.823529	5.823529
-2.41176	2.411765
	5.823529

5	-2	1
-2	1	-0.41414

5.828283	5.828283
-2.41414	2.414141
	5.828283

5	-2	1
-2	1	-0.41421

5.828423	5.828423
-2.41421	2.414211
	5.828423

5	-2	1
-2	1	-0.41421

5.828427	5.828427
-2.41421	2.414213
	5.828427

λ=1/5.828427= 0.171573

Example : Find the smalest eigenvalue of $A = \begin{bmatrix} 3.556 & -1.778 & 0 \\ -1.778 & 3.556 & -1.778 \\ 0 & -1.778 & 3.556 \end{bmatrix}$

by using inverse power method.

Inverse of the matrix A

$$A^{-1} = \begin{bmatrix} 0.42182227 & 0.281215 & 0.14060 \\ 0.28121484 & 0.56243 & 0.28121 \\ 0.14060742 & 0.281215 & 0.42182 \end{bmatrix}$$

0.421822272	0.281215	0.140607	1	0.843645	0.843645	
0.281214848	0.56243	0.281215	1	1.124859	1.124859	
0.140607424	0.281215	0.421822	1	0.843645	0.843645	
					1.124859	
0.421822272	0.281215	0.140607	0.75	0.703037	0.703037	
0.281214848	0.56243	0.281215	1	0.984252	0.984252	
0.140607424	0.281215	0.421822	0.75	0.703037	0.703037	
				0.984252	0.984252	-0.125
0.421822272	0.281215	0.140607	0.714286	0.68295	0.68295	
0.281214848	0.56243	0.281215	1	0.964165	0.964165	
0.140607424	0.281215	0.421822	0.714286	0.68295	0.68295	
				0.964165	0.964165	-0.02041
0.421822272	0.281215	0.140607	0.708333	0.679603	0.679603	
0.281214848	0.56243	0.281215	1	0.960817	0.960817	
0.140607424	0.281215	0.421822	0.708333	0.679603	0.679603	
				0.960817	0.960817	-0.00347
0.421822272	0.281215	0.140607	0.707317	0.679031	0.679031	
0.281214848	0.56243	0.281215	1	0.960246	0.960246	
0.140607424	0.281215	0.421822	0.707317	0.679031	0.679031	
				0.960246	0.960246	-0.00059
0.421822272	0.281215	0.140607	0.707143	0.678933	0.678933	
0.281214848	0.56243	0.281215	1	0.960148	0.960148	
0.140607424	0.281215	0.421822	0.707143	0.678933	0.678933	
				0.960148	0.960148	-0.0001
0.421822272	0.281215	0.140607	0.707113	0.678916	0.678916	
0.281214848	0.56243	0.281215	1	0.960131	0.960131	
0.140607424	0.281215	0.421822	0.707113	0.678916	0.678916	
				0.960131	0.960131	-1.8E-05
0.421822272	0.281215	0.140607	0.707108	0.678913	0.678913	
0.281214848	0.56243	0.281215	1	0.960128	0.960128	
0.140607424	0.281215	0.421822	0.707108	0.678913	0.678913	
				0.960128	0.960128	-3E-06
0.421822272	0.281215	0.140607	0.707107	0.678913	0.678913	
0.281214848	0.56243	0.281215	1	0.960128	0.960128	
0.140607424	0.281215	0.421822	0.707107	0.678913	0.678913	
				0.960128	0.960128	-5.2E-07
0.421822272	0.281215	0.140607	0.707107	0.678913	0.678913	
0.281214848	0.56243	0.281215	1	0.960128	0.960128	
0.140607424	0.281215	0.421822	0.707107	0.678913	0.678913	
				0.960128	0.960128	-8.8E-08
0.421822272	0.281215	0.140607	0.707107	0.678913	0.678913	
0.281214848	0.56243	0.281215	1	0.960128	0.960128	
0.140607424	0.281215	0.421822	0.707107	0.678913	0.678913	
				0.960128	0.960128	-1.5E-08
0.421822272	0.281215	0.140607	0.707107	0.678913	0.678913	
0.281214848	0.56243	0.281215	1	0.960128	0.960128	
0.140607424	0.281215	0.421822	0.707107	0.678913	0.678913	
				0.960128	0.960128	-2.6E-09

λ=1/0.960128= 1.041528

Computation of intermediate eigenvalues by power method

Using power method already given largest and smallest eigenvalue. The remaining eigenvalues can also be found by using power method. The procedure is called deflation. **Hotelling deflation** for symmetric matrices will be described in here. According to this procedure deflated matrix A_i to be used in finding the eigenvector X_i and this eigenvector will be used to define new defleted matrix A_{i+1} acording to relation

$$[A_{i+1}] = [A_{i+1}] - \lambda_i \{X^i\}\{X^i\}^T \quad i = 1,2,..,n-1$$

Eigenvector $\{X^i\}$ has been normalized as. $\{X^i\}\{X^i\}^T = 1$This process can be defined as $k = \frac{1}{|X^i|}$

$$\{X^i\} = k\{X^i\}$$

An example case calculation by using excel code

$$A = \begin{bmatrix} 12 & -6 & -6 \\ 6 & 16 & 2 \\ -6 & 2 & 16 \end{bmatrix}$$

POWER METHOD

A			y	X	X/Xmax
12	6	-6	1	18	1
6	16	2	0.5	13	0.722222222
-6	2	16	-0.5	-13	-0.722222222
				18	

A			y	X	X/Xmax
12	6	-6	1	20.66666667	1
6	16	2	0.7222222	16.11111111	0.779569892
-6	2	16	-0.722222	-16.1111111	-0.779569892
			TOTAL	20.66666667	RESİDUE 2.666666667

A			y	X	X/Xmax
12	6	-6	1	21.35483871	1
6	16	2	0.7795699	16.91397849	0.79204431
-6	2	16	-0.77957	-16.9139785	-0.79204431
				21.35483871	0.688172043

A			y	X	X/Xmax
12	6	-6	1	21.50453172	1
6	16	2	0.7920443	17.08862034	0.794652056
-6	2	16	-0.792044	-17.0886203	-0.794652056
				21.50453172	0.149693012

A			y	X	X/Xmax
12	6	-6	1	21.53582467	1
6	16	2	0.7946521	17.12512878	0.795192617
-6	2	16	-0.794652	-17.1251288	-0.795192617
				21.53582467	0.031292948

A			y	X	X/Xmax
12	6	-6	1	21.54231141	1
6	16	2	0.7951926	17.13269664	0.795304474
-6	2	16	-0.795193	-17.1326966	-0.795304474
				21.54231141	0.006486736

A			y	X	X/Xmax
12	6	-6	1	21.54365369	1
6	16	2	0.7953045	17.13426264	0.795327612
-6	2	16	-0.795304	-17.1342626	-0.795327612
				21.54365369	0.001342282

A					
12	6	-6	1	21.54393134	1
6	16	2	0.7953276	17.13458656	0.795332397
-6	2	16	-0.795328	-17.1345866	-0.795332397
				21.54393134	0.000277654

λ1=		**21.543654**

	1	0.7953324	-0.795	1		
K^2=				0.7953324	2.265107245	0.4414802
				-0.795332		

K= 0.664439764

X1=

1		0.664439764
0.7953324	* 0.6644398	0.528450471
-0.795332		-0.528450471

0.6644398	0.6644398	0.5284505	-0.528
0.5284505			
-0.52845			

0.4414802	0.35112351	-0.351123506
0.351123506	0.2792599	-0.2792599
-0.35112351	-0.2792599	0.2792599

0.4414802	0.3511235	-0.351124			9.511096543	7.56448322	-7.564483216
0.3511235	0.2792599	-0.27926	*	21.543654 =	7.564483216	6.01627857	-6.016278572
-0.351124	-0.27926	0.2792599			-7.56448322	-6.01627857	6.016278572

12	6	-6		9.5110965	7.56448322	-7.56448322		2.488903457	-1.564483	1.5644832
6	16	2	-	7.5644832	6.01627857	-6.01627857	=	-1.564483216	9.9837214	8.0162786
-6	2	16		-7.564483	-6.01627857	6.016278572		1.564483216	8.0162786	9.9837214

A

2.4889035	-1.564483	1.5644832	1	2.488903457	0.127215395
-1.564483	9.9837214	8.0162786	1	16.43551678	0.840069048
1.5644832	8.0162786	9.9837214	1	19.56448322	1
				19.56448322	

2.4889035	-1.564483	1.5644832	0.1272154	0.566836127	0.033506943
-1.564483	9.9837214	8.0162786	0.840069	16.20426757	0.957870264
1.5644832	8.0162786	9.9837214	1	16.91697528	1
				16.91697528	

2.4889035	-1.564483	1.5644832	0.0335069	0.149306812	0.008428414
-1.564483	9.9837214	8.0162786	0.9578703	17.5269674	0.989402588
1.5644832	8.0162786	9.9837214	1	17.71469735	1
				17.71469735	

2.4889035	-1.564483	1.5644832	0.0084284	0.037556983	0.002094851
-1.564483	9.9837214	8.0162786	0.9894026	17.88101228	0.997366052
1.5644832	8.0162786	9.9837214	1	17.92823431	1
				17.92823431	

2.4889035	-1.564483	1.5644832	0.0020949	0.009334649	0.000519106
-1.564483	9.9837214	8.0162786	0.9973661	17.97042604	0.999347305
1.5644832	8.0162786	9.9837214	1	17.9821629	1
				17.9821629	

2.4889035	-1.564483	1.5644832	0.0005191	0.002313135	0.000128539
-1.564483	9.9837214	8.0162786	0.9993473	17.99267155	0.999838382
1.5644832	8.0162786	9.9837214	1	17.99557995	1
				17.99557995	

2.4889035	-1.564483	1.5644832	0.0001285	0.000572769	3.18225E-05
-1.564483	9.9837214	8.0162786	0.9998384	17.99818536	0.999959988
1.5644832	8.0162786	9.9837214	1	17.99890552	1
				17.99890552	

2.4889035	-1.564483	1.5644832
-1.564483	9.9837214	8.0162786
1.5644832	8.0162786	9.9837214

3.182E-05
0.99996
1

0.000141801
17.99955075
17.99972904

17.99972904

7.87794E-06
0.999990095
1

2.4889035	-1.564483	1.5644832
-1.564483	9.9837214	8.0162786
1.5644832	8.0162786	9.9837214

7.878E-06
0.9999901
1

3.51041E-05
17.99988878
17.99993292

17.99993292

1.95023E-06
0.999997548
1

λ2=		**17.999933**

12	6	-6
6	16	2
-6	2	16

1	0	0
0	1	0
0	0	1

C=

	12	6	-6
0.5	0	13	5
-0.5	0	5	13

1	0	0
-0.5	1	0
0.5	0	1

	12	6	-6
	0	13	5
0.38462	0	0	11.076923

1	0	0
-0.5	1	0
0.6923077	-0.38461538	1

inverse A

0.14583	-0.0625	0.090278
-0.0625	0.090278	-0.090278
0.0625	-0.03472	0.090278

0.14583	-0.0625	0.09028
-0.0625	0.09028	-0.0903
0.0625	-0.0347	0.09028

y

1
1
1

X

0.173611111
-0.0625
0.118055556

0.173611111

X/Xmax

1
-0.36
0.68

0.14583	-0.0625	0.09028
-0.0625	0.09028	-0.0903
0.0625	-0.0347	0.09028

1
-0.36
0.68

0.229722222
-0.15638889
0.136388889

0.229722222

1
-0.680773881
0.593712213

0.14583	-0.0625	0.09028
-0.0625	0.09028	-0.0903
0.0625	-0.0347	0.09028

1
-0.680774
0.5937122

0.24198072
-0.17755777
0.139737001

0.24198072

1
-0.733768262
0.577471631

0.14583	-0.0625	0.09028
-0.0625	0.09028	-0.0903
0.0625	-0.0347	0.09028

1
-0.733768
0.5774716

0.243826705
-0.18087582
0.14011092

1
-0.741821219
0.574633201

0.243826705

0.14583	-0.0625	0.09028
-0.0625	0.09028	-0.0903
0.0625	-0.0347	0.09028

1
-0.741821
0.5746332

0.244073768
-0.18134658
0.14013429

0.244073768

1
-0.742999058
0.574147279

0.14583	-0.0625	0.09028
-0.0625	0.09028	-0.0903
0.0625	-0.0347	0.09028

1
-0.742999
0.5741473

0.244103515
-0.18140904
0.140131319

0.244103515

1
-0.743164409
0.574065142

0.14583	-0.0625	0.09028
-0.0625	0.09028	-0.0903
0.0625	-0.0347	0.09028

1
-0.743164
0.5740651

0.244106434
-0.18141656
0.140129645

0.244106434

1
-0.743186296
0.574051419

0.14583	-0.0625	0.09028
-0.0625	0.09028	-0.0903
0.0625	-0.0347	0.09028

1
-0.743186
0.5740514

0.244106563
-0.18141729
0.140129166

0.244106563

1
-0.743188922
0.574049154

λ3=		**4.096571**

An example is given by using python code:

$$A=\begin{bmatrix} 12 & -6 & -6 \\ 6 & 16 & 2 \\ -6 & 2 & 16 \end{bmatrix}$$

Power method to calculate eigenvalues of the symmetric matrices, Python version

```
 """
M. Turhan ÇOBAN
POWER METHOD for eigenvalue calculations
22.01.2022
"""
from math import *

def norm(v):
    # vector 2 norm
    total=0.0
    n=len(v)
    for i in range(n):
        total=total+v[i]*v[i]
    return sqrt(total)

def norminf(a):
    # infinite matrix norm
    x=0.0
    max=0.0
    n=len(a)
    for i in range(n):
      x=abs(a[i])
```

```
    if x>max:
        max=x
    return max

def multiply_m_v(left,right):
    #multiplication of one matrix with one vector
    m1=len(left[0])
    n1=len(left)
    m2=len(right)
    b = [0.0 for i in range(m2)]
    if n1 != m2:
        print("inner matrix dimensions must agree")
        for ii in range(n1):b[ii]=0
        return b
    for i in range(m1):
        b[i]=0
        for k in range(n1):
            b[i]+=left[i][k]*right[k]
    return b

def multiply_c_v(left,right):
    #multiplying a vector with a constant
    n=len(right);
    b = [0.0 for i in range(n)];
    for i in range(n):
        b[i]=left*right[i];
    return b

def multiply_c_m(left,right):
    #multiplying a matrix with a constant
     n=len(right);
     m=len(right[0]);
     b = [[0 for x in range(m)] for y in range(n)];
     for i in range(n):
         for j in range(m):
             b[i][j]=right[i][j]*left;
     return b

def V_VT(left):
    #multiplys a vector with a vector              transpose
    n=len(left);
    aa = [[0 for x in range(n)] for y in range(n)];
    for i in range(n):
        for j in range(n): aa[i][j]=left[i]*left[j];
    return aa

def  VT_X(left,right):
    #multiplys a vector transpose with a vector
    n=len(left)
    tot=0.0
    for i in range(n):
        l1=left[i]
        l2=right[i]
        tot=tot+left[i]*right[i]
    return tot

def substract(left,right):
    # substracting of two matrices
    n1=len(left)
    m1=len(left[0])
    n2=len(right)
    m2=len(right[0])
    nMax=0
    mMax=0
    if m1>=m2:
        mMax=m1
    else:
        mMax=m2
    if n1>=n2:
        nMax=n1
    else:
        nMax=n2
    b=[[0 for i in range(mMax)] for j in range(nMax)]
    for i in range(n1):
       for j in range(m1):
           b[i][j]=b[i][j]+left[i][j]
```

```
    for i in range(n2):
      for j in range(m2):
          b[i][j]=b[i][j]-right[i][j]
    return b

def biggest_eigen_power(A):
    eps=1e-15
    err=1
    m=100
    n=len(A)
    y=[0.0 for i in range(n)]
    r=[0.0 for i in range(n)]
    xnorm=0.0
    mu=1.0
    for i in range(n):
      y[i]=1.0
    k=0
    mueski=0.0
    x=[0.0 for i in range(n)]
    while err>eps:
        mueski=mu
        x=multiply_m_v(A,y)
        xnorm=norminf(x)
        for i in range(n):
            y[i]=x[i]/xnorm
        mu=VT_X(y,x)/VT_X(y,y)
        for i in range(n):
            r[i]=mu*y[i]-x[i]
        k=k+1
        err=abs((mu-mueski)/mu)
        #print("k=",k,"err=",err)
    return mu
# Gauss(DooLittle) LU decomposition
# python version
# Gauss(DoLittle) LU decomposition
def gaussLU(ai):
    #Gauss elimination with pivoting
  n=len(ai)
  x=[0 for i in range(n)]
  a=[[0 for i in range(n)] for j in range(n)]
  carpan=0
  toplam=0
  for i in range(n):
      for j in range(n):
          a[i][j]=ai[i][j]
  for k in range(n-1):
      for i in range(k+1,n):
          carpan=a[i][k]/a[k][k]
          a[i][k]=carpan
          for j in range(k+1,n):
              a[i][j]-=carpan*a[k][j]
  return a

# inverse matrix
def I(n):
    #unit matrix of dimension n
    B = [[0 for x in range(n)] for y in range(n)]
    for i in range(0,n):
        for j in range(0,n):
            if i==j: B[i][j]=1.0
    return B
# bi is vector
def back_substitution(ai,bi):
    #gauss LU linear system of  equation solution
    n=len(ai)
    x=[0 for i in range(n)]
    a=[[0 for i in range(n)] for j in range(n)]
    b=[0 for i in range(n)]
    for i in range(n):
        for j in range(n):
            a[i][j]=ai[i][j]
        b[i]=bi[i]
    for i in range(n):
        toplam=b[i]
        for j in range(i):
            toplam-=a[i][j]*b[j]
```

```
        b[i]=toplam
    x[n-1]=b[n-1]/a[n-1][n-1]
    for i in range((n-2),-1,-1):
        toplam=0
        for j in range(i+1,n):
            toplam+=a[i][j]*x[j]
        x[i]=(b[i]-toplam)/a[i][i]
    return x
# bi is matrix
def back_substitution_n(ai,bi):
    #gauss LU linear system of  equation solution
    n=len(ai)
    m=len(bi)
    x=[[0 for i in range(n)] for j in range(m)]
    a=[[0 for i in range(n)] for j in range(n)]
    b=[[0 for i in range(n)] for j in range(m)]
    for i in range(n):
        for j in range(n):
            a[i][j]=ai[i][j]
            b[i][j]=bi[i][j]
    for k in range(m):

        for i in range(n):
            toplam=b[i][k]
            for j in range(i):
                toplam =toplam - a[i][j]*b[j][k]
            b[i][k]=toplam
        x[n-1][k]=b[n-1][k]/a[n-1][n-1]
        for i in range((n-2),-1,-1):
            toplam=0
            for j in range(i+1,n):
                toplam+=a[i][j]*x[j][k]
            x[i][k]=(b[i][k]-toplam)/a[i][i]
    return x

def AXB(a,b):
    n=len(a)
    c=gaussLU(a)
    s=back_substitution(c,b)
    return s
def inverse(a):
    # Inverse matrix
    n=len(a)
    I1=I(n)
    c=gaussLU(a)
    s=back_substitution_n(c,I1)
    return s

def eigen_power(A):
    eps=1.0e-9
    err=1
    m=700
    n=len(A)
    y=[0 for i in range(n)]
    r=[0 for i in range(n)]
    lmbd=[0 for i in range(n)]
    xnorm=0.0
    mu=1.0
    A1=[[0 for i in range(n)] for j in range(n)]
    X=[[0 for i in range(n)] for j in range(n)]
    for i in range(n):
        for j in range(n):
            A1[i][j]=A[i][j]
    for jj in range(n-1):
        for i in range(n):
            y[i]=1.0
        # end for i in range(n):
        k=0
        mueski=0.0
        x=[0 for i in range(n)]
        while err>eps:
            mueski=mu
            x=multiply_m_v(A,y)
            xnorm=norminf(x)
            for i in range(n):
                y[i]=x[i]/xnorm
```

```
        mu=VT_X(y,x)/VT_X(y,y)
        for i in range(n):
            r[i]=mu*y[i]-x[i]
        k=k+1
        err=abs((mu-mueski)/mu)
    # end while
    err=1
    X[jj]=y
    lmbd[jj]=mu
    cc= VT_X(X[jj],X[jj])
    kk=1.0/norm(X[jj])
    X[jj+1]=multiply_c_v(kk,X[jj])
    A=substract(A,multiply_c_m(lmbd[jj], V_VT(X[jj+1])))
    # end  for jj in range(n-1):
  A2=inverse(A1)
  lmbd[jj+1]=1.0/biggest_eigen_power(A2)
  return lmbd

a=[[12.0,6.0,-6.0],[6.0,16.0,2.0],[-6.0,2.0,16.0]]
l=eigen_power(a)
print("lambda=\n",l)
```

```
runfile('D:/okul/SCO1/eigen_power.py', wdir='D:/okul/SCO1')
lambda=
 [21.544003744145073, 18.06001876460423, 4.45599625468247]
```

$$A = \begin{bmatrix} 3.556 & -1.778 & 0 \\ -1.778 & 3.556 & -1.778 \\ 0 & -1.778 & 3.556 \end{bmatrix}$$

```
a=[[3.556,-1.778,0.0],[-1.778,3.556,-1.778],[0.0,-1.778,3.556]]
l=eigen_power(a)
print("lambda=\n",l)
```

```
runfile('D:/okul/SCO1/eigen_power.py', wdir='D:/okul/SCO1')
inverse matrix:  [[0.421822272215973, 0.281214848143982, 0.140607424071991], [0.281214848143982, 0.562429696287964,
0.281214848143982], [0.140607424071991, 0.281214848143982, 0.421822272215973]]
lambda=
 [6.070471714460811, 1.041528286100637, 1.041528286100637]
```

$$\boldsymbol{A} = \begin{bmatrix} \mathbf{1} & \mathbf{2} \\ \mathbf{2} & \mathbf{5} \end{bmatrix}$$

```
a=[[1.0,2.0],[2.0,5.0]]
l=eigen_power(a)
print("lambda=\n",l)
```

```
runfile('D:/okul/SCO1/eigen_power.py', wdir='D:/okul/SCO1')
inverse matrix:  [[5.0, -2.0], [-2.0, 1.0]]
lambda=
 [5.828427124764781, 0.1715728752538099]
```

12.3 JACOBI METHOD AND HOUSEHOLDER TRANSFORMATION

Jacobi method is used to calculate eigen value problems for the symmetric matrices. For a symmetric matrix, transpose of the matrix is the same as the matrix itself.

$$\{A\}^T = \{A\}$$

By using this property of the symmetric matrices a transformation called similarity tranformation can be applied. This transformation converts the matrix into diagonal form. Eigenvalues can be calculated by using this form. Assume matrix P be a nonsingular matrix similarity transformation :

$$\{B\} = \{P\}^{-1}\{A\}\{P\}$$

After applying similarity transformation eigen values reamain the same, but eigenvectors change. Therefore eigenvalues of matrix B is the same as eigenvalues of the matrix A. Now assume that vector $[y]^k$ is the eigenvector of matrix $\{B\}$ corresponding eigenvalue λ_k. In this case :

$$\{B\}[y]^k = \lambda_k[y]^k$$

$$\{B\} = \{P\}^{-1}\{A\}\{P\}$$

If both side of the equation is multiplied with $\{P\}$:

$$\{P\}\{B\} = \{P\}\{P\}^{-1}\{A\}\{P\} = \{A\}\{P\}$$

and

$$\{A\}\{P\}[y]^k = \{P\}\{B\}[y]^k = \lambda_k\{B\}[y]^k$$

If $[y]^k$ is eigenvector of $\{B\}$, eigenvector of $\{A\}$ took a value like $[x]^k$. The relation between $[y]^k$ and $[x]^k$

$$[x]^k = \{P\}[y]^k$$

Eigenvector of {A} can be obtained from multiplication of eigenvector $\{B\}$ by $\{P\}$
Symmetric matrices can be converted to the diagonal form by applying orthogonal symmetry .Ortagonal ($\{P\}^{-1}=\{P\}^T$) In order to obtained this convertion angular convertion matrix, R(ϕ), can be applied Matrix R(ϕ) is a unit matrix except row p and column q.
$R_{pp}=R_{qq} = \cos(\phi)=c$
$R_{pq}= - R_{qp}= - \sin(\phi)=-s$
For example for n=8 p=3 q=5

$$R(\phi)=\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & R_{pp} & 0 & R_{pq} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & R_{qp} & 0 & R_{qq} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \cos(\phi) & 0 & -\sin(\phi) & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & \sin(\phi) & 0 & \cos(\phi) & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

In this equation ϕ is the rotation angle. Rotation matrix R has interesting properties, one of these is multiplication of rotation matrix and transpose value will be given a unit matrix.
$\{R(\phi)\}^T\{R(\phi)\}=I$
Inversion of the rotation matrix is also very simple.
$\{R(\phi)\}^{-1}=\{R(\phi)\}^T$
For the rotation matrix following property is also valid.
$\{R(\phi)\}^T=\{R(-\phi)\}$
If similarity transformation is applied by using R matrix
$\{B\}=\{R(\phi)\}^T\{A\}\{R(\phi)\}$
In this transform angle ϕ can be selected such that $B_{qp}=0$.
$B_{qp}=(A_{qq} - A_{pp}) \sin(\phi)\cos(\phi)+A_{pq} [\cos^2(\phi) - \sin^2(\phi)] = 0$
If $\sin(2\phi)=2 \sin(\phi)\cos(\phi)$
And remembering that
$\cos(2\phi)=\cos^2(\phi) - \sin^2(\phi)$
$B_{qp}=(A_{qq} - A_{pp}) \sin(2\phi)/2+A_{pq} \cos(2\phi) = 0$
ϕ becomes

$$\phi = \frac{1}{2} arctan\left(\frac{2A_{pq}}{A_{pp} - A_{qq}}\right)$$

By using this angle and R matrix similarity transformation is carried out. Aplication of Jacobi method is as follows: P=I is selected. In upper trianglar part of the matrix the value with the biggest magnitude A_{pq} value is found. By using p and q values (indices of the element) angle ϕ is calculated. {P} ={P}{R} tranformation and ve $\{A\} =\{R\}^T\{A\} \{R\}$ tranformation is applied. And biggest magnitude A_{pq} value is found in the upper triangular part of the matrix again and iteration continued until absolute value A_{pq} is close to zero (equal to ε) . Eigenvalues is taken as $\lambda_k=A_{kk}$, and eigen vector becomes $P=[x^1,x^2,\ldots,x^n]$.

One of the problem of Jacobi iteration is the excessive number of iteration steps. If A matrix is pre-transformed before the process above by using Householder transformation, total number of iteration reduces significantly. Householder transformation converts the matrix into upper Hessenberg matrix.

$$B=\begin{bmatrix} B_{11} & B_{12} & B_{13} & \dots & B_{1,n-1} & B_{1,n} \\ B_{21} & B_{22} & B_{23} & \dots & B_{2,n-1} & B_{2,n} \\ 0 & B_{32} & B_{33} & \dots & B_{3,n-1} & B_{3,n} \\ 0 & 0 & B_{43} & \dots & B_{4,n-1} & B_{4,n} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & B_{n,n-1} & B_{n,n} \end{bmatrix}$$

As it is seen from the matrix itself, upper triangular part plus one below the diagonal is existed in this form, rest of the matrix elements are zero. Householder transformations is as follows:

1. $Q = I$ is taken
2. $\alpha = sign(A_{k+1,k})\sqrt{\sum_{j=k+1}^{n} A_{j,k}^2}$ In this equation $sign(A_{k+1,k})$ is the sign of $(A_{k+1,k})$. $sign(x) = \frac{x}{|x|}$
2. $u^k=[0,\dots,0,A_{k+1,k} + \alpha, A_{k+2,k},\dots, A_{n,k}]^T$.
3. $U = I - \frac{2uu^T}{u^Tu}$
4. Q = Q P is applied.
5. $A=PAP^T$.
6. B=A

B is the upper Hessenberg matrix.

An example problem:

$$A=\begin{bmatrix} 4 & -2 & 6 & 4 \\ -2 & 3 & -1 & 3 \\ 6 & -1 & 22 & 13 \\ 4 & 3 & 13 & 46 \end{bmatrix} \quad (symmetric\ matrix)$$

find the eigenvalues:

Jacobi method to calculate eigenvalues of the symmetric matrices, Python version

```
# -*- coding: utf-8 -*-
"""
Created on Sat Aug 25 17:26:02 2018

@author: Mustafa Turhan Çoban
"""
from math import *;

def I(n):
    #unit matrix of dimension n
    B = [[0 for x in range(n)] for y in range(n)];
    for i in range(0,n):
        for j in range(0,n):
            if i==j: B[i][j]=1.0;
    return B;

def AT(A):
    #transpose matrix
    n=len(A);
    m=len(A[0]);
    B = [[0 for x in range(m)] for y in range(n)];
    for i in range(0,n):
        for j in range(0,m):
            B[i][j]=A[j][i];
    return B;
def multiply_m_m(left,right):
    #multiplying two matrisis
    n1=len(left);
    m1=len(left[0]);
    n2=len(right);
    m2=len(right[0]);
    b = [[0 for x in range(n1)] for y in range(m2)];
    if m1 != n2:
        print("warning : matrix sizes should be same");
        return b;
    for i in range(0,n1):
```

```
        for j in range(0,m2):
            for k in range(0,m1):
                b[i][j]+=left[i][k]*right[k][j];
    return b;

def eigen_jacobi(A):
    # eigen value of a symmetric matrix
    # Jacobi method
    eps=1e-15;
    err=1;
    m=100;
    max=-9.99e100;
    n=len(A);
    P= [[0.0 for x in range(n)] for y in range(n)];
    R= [[0.0 for x in range(n)] for y in range(n)];
    p=n;q=n;
    lambda1=[0.0 for x in range(n)];
    P=I(n);
    for i in range(n):
        for j in range(i+1,n):
            if abs(A[i][j]) > max: max=abs(A[i][j]);p=i;q=j;
    while abs(A[p][q]) > eps:
        fi=0.5*atan(2.0*A[p][q]/(A[p][p]-A[q][q]+1e-40));
        R=I(n);
        R[p][p]=cos(fi);
        R[q][q]=cos(fi);
        R[p][q]=-sin(fi);
        R[q][p]=sin(fi);
        P=multiply_m_m(P,R);
        A= multiply_m_m(AT(R), multiply_m_m(A,R));
        max=-9.99e100;
        for i in range(0,n):
            for j in range(i+1,n):
                if abs(A[i][j]) > max: max=abs(A[i][j]);p=i;q=j;
    for i in range(0,n):lambda1[i]=A[i][i];
    PP= [[0.0 for x in range(n)] for y in range(n+1)];
    PP[0]=lambda1;
    for i in range(1,n+1):
        for j in range(0,n):
            PP[i][j]=P[i-1][j];
    return PP;

a=[[4,-2,6,4],[-2,2,-1,3],[6,-1,22,13],[4,3,13,46]];
c=eigen_jacobi(a);
print("eigen values=\n",c[0]);
```

```
runfile('D:/okul/SCO1/eigen_jacobi.py', wdir='D:/okul/SCO1')
eigen values=
 [3.4693117897172057, 0.1774125248787215, 17.810856825590943, 52.54241885981311]
```

12.4 CHARACTERISTIC POLYNOMIAL AND FADDEEV-LEVERRIER METHOD

Characteristic matrix obtained by substracting of the matrix from eigenvalue matrix(unit matrix and eigenvalue multiplication). Characteristic polynomial obtained when the determinant of this matrix is taken. Roots of characteristic polynomial is given eigenvalues.

$$\Delta(\lambda)=\det\left\{I\lambda-\begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{11} & a_{22} & a_{23} & \dots & a_{2n} \\ a_{11} & a_{11} & a_{33} & \dots & a_{3n} \\ \dots & \dots & \dots & \dots & \dots \\ a_{1n} & a_{2n} & a_{3n} & \dots & a_{nn} \end{bmatrix}\right\} \qquad \Delta(\lambda)=\det\left\{\begin{bmatrix} \lambda-a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{11} & \lambda-a_{22} & a_{23} & \dots & a_{2n} \\ a_{11} & a_{11} & \lambda-a_{33} & \dots & a_{3n} \\ \dots & \dots & \dots & \dots & \dots \\ a_{1n} & a_{2n} & a_{3n} & \dots & \lambda-a_{nn} \end{bmatrix}\right\}$$

Process can be explained by an example problem. If matrix A is given as: $A=\begin{bmatrix}1 & 1\\ 6 & 2\end{bmatrix}$,

characteristic polynomial of this matrix will be:

$$\Delta(\lambda)=\det(\lambda I-A)=det\begin{bmatrix}\lambda-1 & 1\\ 6 & \lambda-2\end{bmatrix}=(\lambda-1)(\lambda-2)-1*6=0$$

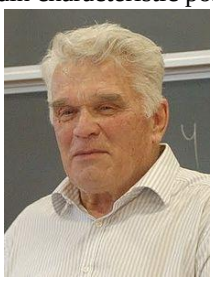
$\lambda^2-3\lambda+2-6=\lambda^2-3\lambda-4=0$

Roots of this polynomial can be obtained by using quadratic equation solution:

$$\lambda_{1,2} = \frac{3 \pm \sqrt{9+16}}{2} = \{4, -1\}$$

So the roots of the characteristic polynomial will give us eigenvalues. Method will be valid for non-symmetric matrices as well. In case of non symmetric matrices complex root finding methods such as Bairstow method should be utilised. In this section Leverrier method to obtain characteristic polynomial will be investigated.

Urbain Le Verrier



Ludvig Dmitrievich Faddeev

Algorithm of the method can be given as:

B_n=A and a_n=-trace(B_n) (trace process add the diagonal elements of the matrix)

$$trace(B_n) = \sum_{i=1}^{n-1} b_{ii}$$

for k=n-2 to 0

{ C_k=(B_{k+1}+a_{k+1}I)

B_k=AC_k

$$a_k = -\frac{trace(B_n)}{n-k}$$

}

Another interesting result of the Leverrier algorithm is that inverse matrix can be obtained from simple relation

$$A^{-1} = \frac{C_0}{a_0}$$

As it is seen from the algorithm, applying of the method is quite simple. In our example program, characteristic polynomial will be obtained by using Faddeev-LeVerrier method, and Once Eigen value matrix converted to characterisitic polynomial then whole eigen values can be solved by using Horner's synthetic division method and some root finding methods. Originally Horner's synthetic division method is combined with Newton-Raphson method to find roots, but some other more efficient root finding methods can also be employed.

Horners method utilises synthetic division process to solve roots of polynomials. Synthetic division process use recursive process to obtain a division for a polynomial. If

$$P(x) = f_n(x) = a_0 + a_1 x + a_1 x^2 + \cdots + a_n x^n$$

and if this polynomial divides with $(x - x_0)$ to form polynomial

$$Q(x) = f_{n-1}(x) = b_1 + b_2 x + b_3 x^2 + \cdots + b_{n-1} x^{n-1}$$

Where $P(x) = f_n(x) = b_0 + (x - x_0) f_{n-1}(x)$ or $P(x) = b_0 + (x - x_0) Q(x)$

$R = b_0$ is the remainder of the synthetic division process. Of course if the remainder is equal to zero x_0 would be the root of this polynomial. Synthetic division can be calculated as following iterative equation:

$$b_n = a_n$$

$$b_i = a_i + b_{i+1}$$

Differentiating of $f_n(x)$with respect to x gives

$$f'_n(x) = f_{n-1}(x) + (x - x_0) f'_{n-1}(x)$$

and at x=x_0

$$f_n(x) = b_0$$

$$f'_n(x) = f_{n-1}(x) = Q(x)$$

Therefore Newton - Raphson iterative method can be applied to find the roots by using iterative process

$$x_n = x_{n-1} - \frac{f_n(x_{n-1})}{f'_n(x_{n-1})} = x_{n-1} - \frac{b_0}{f'_n(x_{n-1})}$$

$$x_n = x_{n-1} - \frac{b_0}{Q(x_{n-1})}$$

As an example problem let us calculate eigenvalues of :

$$A = \begin{bmatrix} 3 & 1 & 5 \\ 3 & 3 & 1 \\ 4 & 6 & 4 \end{bmatrix} \ (Non-symmetric\ matrix)$$

$$B = \begin{bmatrix} 4 & -2 & 6 & 4 \\ -2 & 3 & -1 & 3 \\ 6 & -1 & 22 & 13 \\ 4 & 3 & 13 & 46 \end{bmatrix} \ (symmetric\ matrix)$$

After converting

The following programs will be used Leverier method to find charecteristic matrix then use Horner method to solve it in python code. Each code utilises a different curve fitting method

Program 12.4.1 Leverrier method with horner's synthetic division and Newton-Raphson root finding method (complex eigen values) python code : leverrier.py

```
from math import *

def f_poly(c1,x):
    # this function calculates the value of
    # c least square curve fitting function
    n=len(c1)
    if n!=0.0:
      ff=c1[n-1]
      for i in range(n-2,-1,-1):
        ff=ff*x+c1[i]
    else:
      ff=0,0
    return ff

def isSymmetric(a):
    b=True
    n=len(a)
    for i in range(n):
        for j in range(n):
            if a[i][j] != a[j][i]:
                b=False
    return b

def horner(a,x1):
    # roots of a polynomial by using synthetic division process
    # Horner method (utilises Newton-Raphson method)
    n=len(a)
    n1=n;
    nmax=200
    tolerance=1.0e-20
    n1=n+1
    ref=3.1
    y=[complex(0.0,0.0) for i in range(n1-2)]
    n2=n1-2
    for k in range(0,n2):
       n=len(a)
       x=x1;
       xold=3.4878;
       ref=abs(xold-x)
       b=[complex(0.0,0) for i in range(n)]
       c=[complex(0.0,0.0) for i in range(n-1)]
       b[n-1]=a[n-1]
       ii=0;
       while ref>tolerance and ii<nmax:
         # Synthetic division
         for i in range(n-2,-1,-1):
             b[i]=a[i]+b[i+1]*x
         for i in range(n-2,-1,-1):  c[i]=b[i+1]
         Q=f_poly(c,x)
         P=b[0];
         # Newton-Raphson root finding
         x=x-P/Q
```

```
            ref=abs(xold-x)
            ii=ii+1
            xold=x
        a=c;
        n=len(a)
        y[k]=x
    return y


#trace function
def trace(B):
   sum=0;
   n=len(B);
   for i in range(0,n):sum+=B[i][i];
   return sum;

def add(A,a):
   n=len(A);
   m=len(A[0]);
   w, h = 8, 5;
   B = [[0 for x in range(m)] for y in range(n)];
   for i in range(0,n):
      for j in range(0,m):
        if i!=j: B[i][j]=A[i][j];
        else:     B[i][j]=A[i][j]+a;
   return B;

def multiply_c_m(left,right):
   #multiplying a matrix with a constant
    n=len(right);
    m=len(right[0]);
    b = [[0 for x in range(m)] for y in range(n)];
    for i in range(0,n):
       for j in range(0,m):
          b[i][j]=right[i][j]*left;
    return b;
#end of multiplying a matrix with a constant double

def multiply_m_m(left,right):
   #multiplying two matrisis
   n1=len(left);
   m1=len(left[0]);
   n2=len(right);
   m2=len(right[0]);
   b = [[0 for x in range(n1)] for y in range(m2)];
   if m1 != n2:
     print("uyarı: matris boyutları aynı olmalıdır");
     return b;
   for i in range(0,n1):
      for j in range(0,m2):
         for k in range(0,m1):
            b[i][j]+=left[i][k]*right[k][j];
   return b;

# convert matrix to characteristic polynomial form
def polynomial_coefficients(A):
   n=len(A);
   m=len(A[0]);
   B = [[0 for x in range(m)] for y in range(n)];
   for i in range(0,n):
      for j in range(0,m):
         B[i][j]=A[i][j];
   a = [0.0 for y in range(n+1)];
   C = [[0.0 for x in range(m)] for y in range(n)];
   a[n]=1.0;
   a[n-1]=-trace(B);
   for k in range(n-2,-1,-1):
     C=add(B,a[k+1])
     B=multiply_m_m(A,C)
     a[k]=-trace(B)/(n-k)
   return a

def eigen_horner(A):
   b=polynomial_coefficients(A);
   if isSymmetric(A):
       x1=1.56435
```

```
    else:
        x1=complex(1.56435,0.1)
    return horner(b,x1)

A=[[3,1,5],[3,3,1],[4,6,4]]
a=polynomial_coefficients(A)
print("a=\n",a)
print("A=\n",A)
print(eigen_horner(A))
B=[[4,-2,6,4],[-2,3,-1,3],[6,-1,22,13],[4,3,13,46]]
print("B=\n",B)
print(eigen_horner(B))
```

```
 runfile('D:/okul/SCO1/leverier.py', wdir='D:/okul/SCO1')
a=
 [-40.0, 4.0, -10, 1.0]
A=
 [[3, 1, 5], [3, 3, 1], [4, 6, 4]]
[(7.71137312019964e-21+2j), (-8.423007366470451e-23-2j), (10+0j)]
B=
 [[4, -2, 6, 4], [-2, 3, -1, 3], [6, -1, 22, 13], [4, 3, 13, 46]]
[0.6982737832303404, 3.9170557010445037, 17.8405606144007, 52.54410990132446]
```

```
A=[[4,-2,6,4],[-2,2,-1,3],[6,-1,22,13],[4,3,13,46]];
print(eigen_horner(A));
```

```
runfile('D:/okul/SCO1/eigen_horner.py', wdir='D:/okul/SCO1')
[(0.1774125248787216+0j), (3.469311789717206-6.393786517153874e-23j), (17.810856825590946+1.6791691784404488e-23j),
(52.542418859813125-1.6382924242895428e-24j)]
```

Le Guerre root finding method is also specificaly developed to find roots of a polynomial. Method formulations is as follows:

$$g(x) = [f'(x)]^2 - \frac{nf(x)f''(x)}{n-1} \qquad (2.15.10)$$

$$x_{k+1} = x_k - \frac{nf(x_k)}{f'(x_k) \pm (n-1)\sqrt{g(x_k)}} \qquad (2.15.11)$$

Leverrier method with Horner's synthetic division and LeGuerre root finding method (complex eigen values) python code

```
 # -*- coding: utf-8 -*-
"""
Created on Sun Sep  2 20:11:52 2018

@author: Mustafa Turhan Çoban


"""
from math import *
from cmath import *
def f_poly(c1,x):
    # this function calculates the value of
    # c least square curve fitting function
    n=len(c1);
    if n!=0.0:
      ff=c1[n-1];
      for i in range(n-2,-1,-1):
        ff=ff*x+c1[i];
    else:
      ff=0,0;
    return ff;

def df_poly(c1,x):
    h=0.001
    return (-f_poly(c1,(x+2.0*h))+8.0*f_poly(c1,(x+h))-8.0*f_poly(c1,(x-h))+f_poly(c1,(x-2.0*h)))/(12.0*h);

def d2f_poly(c1,x):
    dx=0.001
    y1=-1.0/12.0*f_poly(c1,(x-2.0*dx))+4.0/3.0*f_poly(c1,(x-dx));
    y2=-5.0/2.0*f_poly(c1,x)+4.0/3.0*f_poly(c1,(x+dx))-1.0/12.0*f_poly(c1,(x+2.0*dx));
    return (y1+y2)/dx;

def trace(B):
```

```
    sum=0;
    n=len(B);
    for i in range(0,n):sum+=B[i][i];
    return sum;

def topla(A,a):
    n=len(A);
    m=len(A[0]);
    B = [[0 for x in range(m)] for y in range(n)];
    for i in range(0,n):
      for j in range(0,m):
        if i!=j: B[i][j]=A[i][j];
        else:     B[i][j]=A[i][j]+a;
    return B;

def carp_m_m(left,right):
    #multiplying two matrisis
    n1=len(left);
    m1=len(left[0]);
    n2=len(right);
    m2=len(right[0]);
    b = [[0 for x in range(n1)] for y in range(m2)];
    if m1 != n2:
      print("warning : matrix sizes should be same");
      return b;
    for i in range(0,n1):
      for j in range(0,m2):
        for k in range(0,m1):
          b[i][j]+=left[i][k]*right[k][j];
    return b;
def eigen_LeGuerre(A):
   #eigen value result is complex use Horner method with synthetic division
   b=polynomial_coefficients(A);
   return LeGuerre_SD(b);

def polynomial_coefficients(A):
    n=len(A);
    m=len(A[0]);
    B = [[0 for x in range(m)] for y in range(n)];
    for i in range(0,n):
      for j in range(0,m):
        B[i][j]=A[i][j];
    a = [0.0 for y in range(n+1)];
    C = [[0.0 for x in range(m)] for y in range(n)];
    a[n]=1.0;
    a[n-1]=-trace(B);
    for k in range(n-2,-1,-1):
       C=topla(B,a[k+1]);
       B=carp_m_m(A,C);
       a[k]=-trace(B)/(n-k);
    return a;
def LeGuerre_SD(a):
  # roots of a polynomial by using synthetic division process
  # complex roots
   n=len(a);
   x1=complex(1.56435,1.76353);
   x=complex(1.56435,1.87363);
   n1=n;
   nmax=500;
   tolerance=1.0e-15;
   n1=n+1;
   ref=3.1;
   y=[complex(0.0,0.0) for i in range(n1-2)];
   n2=n1-2;
   for k in range(0,n2):
         n=len(a);
         m=n-1;
         x=x1;
         xold=1.4878;
         ref=abs(xold-x);
         b=[complex(0.0,0) for i in range(n)];
         c=[complex(0.0,0.0) for i in range(n-1)];
         b[n-1]=a[n-1];
         ii=0;
         while ref>tolerance and ii<nmax:
            for i in range(n-2,-1,-1):
```

```
                b[i]=a[i]+b[i+1]*x;
            for i in range(n-2,-1,-1):  c[i]=b[i+1];
            dfx=df_poly(a,x);
            fx=f_poly(a,x);
            d2fx=d2f_poly(a,x);
            G=dfx/fx;
            H=G*G-d2fx/fx;
            y1=sqrt((m*H-G*G)*(m-1.0));
            z1=G+y1;
            z2=G-y1;
            zbool=z1.real*z1.real+z1.imag*z1.imag> z2.real*z2.real+z2.imag*z2.imag;
            if zbool: z=z1;
            else: z=z2;
            aa=n/z;
            x=x-aa;
            ref=abs(xold-x);
            ii=ii+1;
            xold=x;
        if n==2: y[k]=-a[0]/a[1];
        else:    y[k]=x;
        a=c;
        n=len(a);
    return y;

A=[[4,1,1],[0,2,2],[-2,0,9]];
b= eigen_LeGuerre(A);
print("b=",b);
```

```
runfile('D:/okul/SCO1/LeGuerre.py', wdir='D:/okul/SCO1')
b= [(1.7783225824834867+1.0465662590966393e-240j), (4.817630421052081-1.0052689051413437e-152j), (8.404046996464423-
2.010537810284278e-152j)]
```

```
A=[[8,-1,-5],[-4,4,-2],[18,-5,-7]];
print(eigen_horner(A));
```

```
runfile('D:/okul/SCO1/eigen_horner.py', wdir='D:/okul/SCO1')
[(1+0j), (1.9999999999999998+4j), (2-4j)]
```

Shengfeng Li**[96]** root finding algorithm algorithm:

$$y_k = x_k - \frac{f(x_k)}{f'(x_k)}$$

$$x_{k+1} = x_k - \frac{(f(x_k) - f(y_k))f(x_k)}{(f(x_k) - 2f(y_k))f'(x_k)}$$

```
"""
  Leverrier charecteristic polynomial finding
  f(x)=1.0*x^n+a_n-1*x^n-1+..+a0

  combined with Horner's method of synthetic division

  combined with Shenfeng Li root finding
  Leverrier-Shenfeng  eigenvalue solution method
  # A fourth order root finding methods

  # reference : Fourth-order iterative method without calculating
  # the higher derivatives for nonlinear equations, Shenfeng Li
  # Journal of Algorithms & computational Technology Vol 13:1-8
  # DOI:10.1177/1748302619887686

  M. Turhan ÇOBAN
  24.01.2022
"""
from math import *

def f_poly(c1,x):
    # this function calculates the value of
    # least square curve fitting function
    # polynomial coefficients: c1
    # f(x)=c1[0]+c1[1]*x+c1[2]*x*x+...+c1[n]*x^n
    n=len(c1)
    if n!=0.0:
```

```
        ff=c1[n-1]
        for i in range(n-2,-1,-1):
            ff=ff*x+c1[i]
    else:
        ff=0,0
    return ff

def df_poly(c1,x):
    # this function calculates the derivative of
    # least square curve fitting function
    # polynomial coefficients: c1
    # f(x)=c1[1]+2c1[2]*x+...+n*c1[n]*x^(n-1)
    n=len(c1)
    if n!=0.0:
        ff=(n-1)*c1[n-1]
        for i in range(n-2,-1,-1):
            ff=ff*x+c1[i]*i
    else:
        ff=0,0
    return ff/x

def d2f_poly(c1,x):
    # this function calculates the second derivative of
    # least square curve fitting function
    # polynomial coefficients: c1
    # f(x)=c1[1]+2c1[2]*x+...+n*c1[n]*x^(n-1)
    n=len(c1)
    if n!=0.0:
        ff=(n-1)*(n-2)*c1[n-1]
        for i in range(n-2,0,-1):
            ff=ff*x+c1[i]*i*(i-1)
    else:
        ff=0,0
    return ff/x

def isSymmetric(a):
    b=True
    n=len(a)
    for i in range(n):
        for j in range(n):
            if a[i][j] != a[j][i]:
                b=False
    return b

def Shenfeng(a,x1):
    # roots of a polynomial by using synthetic division process
    # Horner method (utilises Shenfeng Li root finding  method)
    n=len(a)
    n1=n;
    nmax=200
    tolerance=1.0e-20
    n1=n+1
    ref=3.1
    y=[complex(0.0,0.0) for i in range(n1-2)]
    n2=n1-2
    for k in range(0,n2):
        n=len(a)
        x=x1;
        xold=3.4878;
        ref=abs(xold-x)
        b=[complex(0.0,0.0) for i in range(n)]
        c=[complex(0.0,0.0) for i in range(n-1)]
        b[n-1]=a[n-1]
        ii=0;
        while ref>tolerance and ii<nmax:
            # Synthetic division
            for i in range(n-2,-1,-1):
                b[i]=a[i]+b[i+1]*x
            for i in range(n-2,-1,-1):  c[i]=b[i+1]
            Q=f_poly(c,x)
            P=b[0]
            dQ=df_poly(c,x)
            d2Q=df_poly(c,x)
            # Shenfeng root finding
            # y=x-fk/dfk
            # x=x-(fk-fyk)fk/((fk-2fyk)dfk)
```

```
            fk=P
            dfk=Q
            y1=x-fk/dfk
            fyk=f_poly(c,y1)
            if fk!=0:
                x=x-(fk-fyk)*fk/((fk-2.0*fyk)*dfk)
            else:
                break
            ref=abs(xold-x)
            ii=ii+1
            xold=x
        a=c;
        n=len(a)
        y[k]=x
    return y


#trace function
def trace(B):
   sum=0;
   n=len(B);
   for i in range(0,n):sum+=B[i][i];
   return sum;
## Adding a constant to a matrix
def add(A,a):
   n=len(A);
   m=len(A[0]);
   w, h = 8, 5;
   B = [[0 for x in range(m)] for y in range(n)];
   for i in range(0,n):
      for j in range(0,m):
        if i!=j: B[i][j]=A[i][j];
        else:     B[i][j]=A[i][j]+a;
   return B;

def multiply_c_m(left,right):
   #multiplying a matrix with a constant
    n=len(right);
    m=len(right[0]);
    b = [[0 for x in range(m)] for y in range(n)];
    for i in range(0,n):
       for j in range(0,m):
          b[i][j]=right[i][j]*left;
    return b;
#end of multiplying a matrix with a constant double

def multiply_m_m(left,right):
   # multiplying two matrises
   n1=len(left)
   m1=len(left[0])
   n2=len(right)
   m2=len(right[0])
   b = [[0 for x in range(n1)] for y in range(m2)];
   if m1 != n2:
     print("uyarı: matris boyutları aynı olmalıdır");
     return b;
   for i in range(0,n1):
      for j in range(0,m2):
          for k in range(0,m1):
              b[i][j]+=left[i][k]*right[k][j];
   return b;

# convert matrix to characteristic polynomial form
def polynomial_coefficients(A):
   # Leverrier charecteristic polynomial finding
   n=len(A);
   m=len(A[0]);
   B = [[0 for x in range(m)] for y in range(n)];
   for i in range(0,n):
      for j in range(0,m):
         B[i][j]=A[i][j];
   a = [0.0 for y in range(n+1)];
   C = [[0.0 for x in range(m)] for y in range(n)];
   a[n]=1.0;
   a[n-1]=-trace(B);
   for k in range(n-2,-1,-1):
```

```
    C=add(B,a[k+1])
    B=multiply_m_m(A,C)
    a[k]=-trace(B)/(n-k)
  return a

# calculate eigenvalues
def eigen_Shenfeng(A):
  b=polynomial_coefficients(A);
  if isSymmetric(A):
      x1=1.56435
  else:
      x1=complex(1.56435,0.1)
  return Shenfeng(b,x1)

# main method
A=[[4,1,1],[0,2,2],[-2,0,9]];
a=polynomial_coefficients(A)
print("a=\n",a)
print("A=\n",A)
print(eigen_Shenfeng(A))
A=[[3,1,5],[3,3,1],[4,6,4]]
a=polynomial_coefficients(A)
print("a=\n",a)
print("A=\n",A)
print(eigen_Shenfeng(A))
B=[[4,-2,6,4],[-2,3,-1,3],[6,-1,22,13],[4,3,13,46]]
b=polynomial_coefficients(A)
print("B=\n",B)
print(eigen_Shenfeng(B))
```

```
runfile('D:/okul/SCO1/eigen_Shenfeng.py', wdir='D:/okul/SCO1')
a=
 [-72.0, 64.0, -15, 1.0]
A=
 [[4, 1, 1], [0, 2, 2], [-2, 0, 9]]
[(1.7783225824834874+5.435551211630808e-21j), (4.817630421052083-1.3168563040299001e-20j),
(8.40404699646443+1.026260616815705e-21j)]
a=
 [-40.0, 4.0, -10, 1.0]
A=
 [[3, 1, 5], [3, 3, 1], [4, 6, 4]]
[(2.734723870874897e-20+2j), (-6.94511260352448e-21-2j), (10+7.792877041947547e-21j)]
B=
 [[4, -2, 6, 4], [-2, 3, -1, 3], [6, -1, 22, 13], [4, 3, 13, 46]]
[0.6982737832303404, 3.9170557010445033, 17.840560614400697, 52.54410990132446]
```

Program can also be utilised to find the roots of a polynomial

```
# main method
a=[1,3,3,1]
x1=1.2
print(Shenfeng(a,x1))
```

```
runfile('D:/okul/SCO1/eigen_Shenfeng.py', wdir='D:/okul/SCO1')
[-1.000003858948008, -0.9999983398474983, -0.9999998288041945]
```

```
# main method
a=[1,3,3,1]
x1=complex(1.2,1.0)
print(Shenfeng(a,x1))
```

```
runfile('D:/okul/SCO1/eigen_Shenfeng.py', wdir='D:/okul/SCO1')
[(-0.9999819297059646-7.42756028756881e-06j), (-1.0000030182232078+2.320794070394966e-05j), (-1.00001858954043-
1.4217830169109477e-05j)]
```

```
# main method
a=[1,2,3,4,5]
x1=complex(1.2,1.0)
print(Shenfeng(a,x1))
```

```
runfile('D:/okul/SCO1/eigen_Shenfeng.py', wdir='D:/okul/SCO1')
[(0.13783227490298997+0.6781543891053363j), (-0.5378322749029898+0.35828468634512795j), (-0.53783227490299-
0.3582846863451278j), (0.13783227490298983-0.6781543891053365j)]
```

In [48], Abbasbandy gives a third order iterative method, called **Abbasbandy's method**(AM) provisionally,

which is expressed as

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} - \frac{1}{2}\frac{f(x_n)f(x_n)f''(x_n)}{f'(x_n)f'(x_n)f'(x_n)} - \frac{1}{6}\frac{f(x_n)f(x_n)f(x_n)f'''(x_n)}{f'(x_n)f'(x_n)f'(x_n)f'(x_n)}$$

```
"""
  Leverrier charecteristic polynomial finding
  f(x)=1.0*x^n+a_n-1*x^n-1+..+a0
  combined with Abbasbandy root finding
  Leverrier-Abbasbandy eigenvalue solution method
  M. Turhan ÇOBAN
  22.01.2022
"""
from math import *

def f_poly(c1,x):
    # this function calculates the value of
    # least square curve fitting function
    # polynomial coefficients: c1
    # f(x)=c1[0]+c1[1]*x+c1[2]*x*x+...+c1[n]*x^n
    n=len(c1)
    if n!=0.0:
      ff=c1[n-1]
      for i in range(n-2,-1,-1):
        ff=ff*x+c1[i]
    else:
      ff=0,0
    return ff

def df_poly(c1,x):
    # this function calculates the derivative of
    # least square curve fitting function
    # polynomial coefficients: c1
    # f(x)=c1[1]+2c1[2]*x+...+n*c1[n]*x^(n-1)
    n=len(c1)
    if n!=0.0:
      ff=(n-1)*c1[n-1]
      for i in range(n-2,-1,-1):
        ff=ff*x+c1[i]*i
    else:
      ff=0,0
    return ff/x

def d2f_poly(c1,x):
    # this function calculates the second derivative of
    # least square curve fitting function
    # polynomial coefficients: c1
    # f(x)=c1[1]+2c1[2]*x+...+n*c1[n]*x^(n-1)
    n=len(c1)
    if n!=0.0:
      ff=(n-1)*(n-2)*c1[n-1]
      for i in range(n-2,0,-1):
        ff=ff*x+c1[i]*i*(i-1)
    else:
      ff=0,0
    return ff/x

def isSymmetric(a):
    b=True
    n=len(a)
    for i in range(n):
       for j in range(n):
          if a[i][j] != a[j][i]:
             b=False
    return b

def Abbasbandy(a,x1):
    # roots of a polynomial by using synthetic division process
    # Horner method (utilises Newton-Raphson method)
    n=len(a)
    n1=n;
    nmax=200
    tolerance=1.0e-20
    n1=n+1
```

```
    ref=3.1
    y=[complex(0.0,0.0) for i in range(n1-2)]
    n2=n1-2
    for k in range(0,n2):
        n=len(a)
        x=x1;
        xold=3.4878;
        ref=abs(xold-x)
        b=[complex(0.0,0) for i in range(n)]
        c=[complex(0.0,0.0) for i in range(n-1)]
        b[n-1]=a[n-1]
        ii=0;
        while ref>tolerance and ii<nmax:
          # Synthetic division
          for i in range(n-2,-1,-1):
              b[i]=a[i]+b[i+1]*x
          for i in range(n-2,-1,-1):  c[i]=b[i+1]
          Q=f_poly(c,x)
          P=b[0]
          dQ=df_poly(c,x)
          d2Q=df_poly(c,x)
          # Abbasbandy root finding
          # x=x-fx/dfx-0.5*(fx*fx*d2fx)/(dfx*dfx*dfx)-0.1666667*(fx*fx*fx*d3fx)/(dfx*dfx*dfx*dfx)
          x=x-P/Q-0.5*(P*P*dQ)/(Q*Q*Q)-0.16666666*(P*P*P*d2Q)/(Q*Q*Q*Q)
          ref=abs(xold-x)
          ii=ii+1
          xold=x
        a=c;
        n=len(a)
        y[k]=x
    return y


#trace function
def trace(B):
   sum=0;
   n=len(B);
   for i in range(0,n):sum+=B[i][i];
   return sum;

def add(A,a):
   n=len(A);
   m=len(A[0]);
   w, h = 8, 5;
   B = [[0 for x in range(m)] for y in range(n)];
   for i in range(0,n):
      for j in range(0,m):
        if i!=j: B[i][j]=A[i][j];
        else:     B[i][j]=A[i][j]+a;
   return B;

def multiply_c_m(left,right):
   #multiplying a matrix with a constant
    n=len(right);
    m=len(right[0]);
    b = [[0 for x in range(m)] for y in range(n)];
    for i in range(0,n):
       for j in range(0,m):
          b[i][j]=right[i][j]*left;
    return b;
#end of multiplying a matrix with a constant double

def multiply_m_m(left,right):
   #multiplying two matrisis
   n1=len(left);
   m1=len(left[0]);
   n2=len(right);
   m2=len(right[0]);
   b = [[0 for x in range(n1)] for y in range(m2)];
   if m1 != n2:
     print("uyarı: matris boyutları aynı olmalıdır");
     return b;
   for i in range(0,n1):
      for j in range(0,m2):
          for k in range(0,m1):
              b[i][j]+=left[i][k]*right[k][j];
```

```
  return b;

# convert matrix to characteristic polynomial form
def polynomial_coefficients(A):
  n=len(A);
  m=len(A[0]);
  B = [[0 for x in range(m)] for y in range(n)];
  for i in range(0,n):
    for j in range(0,m):
      B[i][j]=A[i][j];
  a = [0.0 for y in range(n+1)];
  C = [[0.0 for x in range(m)] for y in range(n)];
  a[n]=1.0;
  a[n-1]=-trace(B);
  for k in range(n-2,-1,-1):
    C=add(B,a[k+1])
    B=multiply_m_m(A,C)
    a[k]=-trace(B)/(n-k)
  return a

def eigen_Abbasbandy(A):
  b=polynomial_coefficients(A);
  if isSymmetric(A):
      x1=1.56435
  else:
      x1=complex(1.56435,0.1)
  return Abbasbandy(b,x1)

A=[[4,1,1],[0,2,2],[-2,0,9]];
a=polynomial_coefficients(A)
print("a=\n",a)
print("A=\n",A)
print(eigen_Abbasbandy(A))
A=[[3,1,5],[3,3,1],[4,6,4]]
a=polynomial_coefficients(A)
print("a=\n",a)
print("A=\n",A)
print(eigen_Abbasbandy(A))
B=[[4,-2,6,4],[-2,3,-1,3],[6,-1,22,13],[4,3,13,46]]
print("B=\n",B)
print(eigen_Abbasbandy(B))
```

```
runfile('D:/okul/SCO1/eigen_abbasbandy.py', wdir='D:/okul/SCO1')
a=
 [-72.0, 64.0, -15, 1.0]
A=
 [[4, 1, 1], [0, 2, 2], [-2, 0, 9]]
[(1.778322582483487+0j), (4.817630421052085-8.376218261069452e-82j), (8.404046996464427-1.4413851728993445e-27j)]
a=
 [-40.0, 4.0, -10, 1.0]
A=
 [[3, 1, 5], [3, 3, 1], [4, 6, 4]]
[(-9.46415628388234e-21+2j), (-4.097485474232352e-23-2j), (10+0j)]
B=
 [[4, -2, 6, 4], [-2, 3, -1, 3], [6, -1, 22, 13], [4, 3, 13, 46]]
[0.6982737832303404, 3.9170557010445037, 17.8405606144007, 52.54410990132446]
```

Halley's root finding method.

$$x_{n+1} = x_n - \frac{2f(x_n)f'(x_n)}{2f'(x_n)f'(x_n) - f(x_n)f''(x_n)}$$

```
"""
  Leverrier charecteristic polynomial finding
  f(x)=1.0*x^n+a_n-1*x^n-1+..+a0
  combined with Halley root finding
  Leverrier-Halley eigenvalue solution method
  M. Turhan ÇOBAN
  22.01.2022
"""
from math import *

def f_poly(c1,x):
    # this function calculates the value of
```

```
    # least square curve fitting function
    # polynomial coefficients: c1
    # f(x)=c1[0]+c1[1]*x+c1[2]*x*x+...+c1[n]*x^n
    n=len(c1)
    if n!=0.0:
      ff=c1[n-1]
      for i in range(n-2,-1,-1):
        ff=ff*x+c1[i]
    else:
      ff=0,0
    return ff

def df_poly(c1,x):
    # this function calculates the derivative of
    # least square curve fitting function
    # polynomial coefficients: c1
    # f(x)=c1[1]+2c1[2]*x+...+n*c1[n]*x^(n-1)
    n=len(c1)
    if n!=0.0:
      ff=(n-1)*c1[n-1]
      for i in range(n-2,-1,-1):
        ff=ff*x+c1[i]*i
    else:
      ff=0,0
    return ff/x

def d2f_poly(c1,x):
    # this function calculates the second derivative of
    # least square curve fitting function
    # polynomial coefficients: c1
    # f(x)=c1[1]+2c1[2]*x+...+n*c1[n]*x^(n-1)
    n=len(c1)
    if n!=0.0:
      ff=(n-1)*(n-2)*c1[n-1]
      for i in range(n-2,0,-1):
        ff=ff*x+c1[i]*i*(i-1)
    else:
      ff=0,0
    return ff/x

def isSymmetric(a):
    b=True
    n=len(a)
    for i in range(n):
       for j in range(n):
          if a[i][j] != a[j][i]:
             b=False
    return b

def halley(a,x1):
    # roots of a polynomial by using synthetic division process
    # Horner method (utilises Newton-Raphson method)
    n=len(a)
    n1=n;
    nmax=200
    tolerance=1.0e-20
    n1=n+1
    ref=3.1
    y=[complex(0.0,0.0) for i in range(n1-2)]
    n2=n1-2
    for k in range(0,n2):
       n=len(a)
       x=x1;
       xold=3.4878;
       ref=abs(xold-x)
       b=[complex(0.0,0) for i in range(n)]
       c=[complex(0.0,0.0) for i in range(n-1)]
       b[n-1]=a[n-1]
       ii=0;
       while ref>tolerance and ii<nmax:
         # Synthetic division
         for i in range(n-2,-1,-1):
            b[i]=a[i]+b[i+1]*x
         for i in range(n-2,-1,-1):  c[i]=b[i+1]
         Q=f_poly(c,x)
         P=b[0]
```

```
            dQ=df_poly(c,x)
            # Halley root finding
            # x=x-2.0*fx*dfx/(2.0*dfx*dfx-fx*d2fx)
            x=x-2.0*P*Q/(2.0*Q*Q-P*dQ)
            ref=abs(xold-x)
            ii=ii+1
            xold=x
        a=c;
        n=len(a)
        y[k]=x
    return y


#trace function
def trace(B):
   sum=0;
   n=len(B);
   for i in range(0,n):sum+=B[i][i];
   return sum;

def add(A,a):
   n=len(A);
   m=len(A[0]);
   w, h = 8, 5;
   B = [[0 for x in range(m)] for y in range(n)];
   for i in range(0,n):
      for j in range(0,m):
        if i!=j: B[i][j]=A[i][j];
        else:     B[i][j]=A[i][j]+a;
   return B;

def multiply_c_m(left,right):
   #multiplying a matrix with a constant
    n=len(right);
    m=len(right[0]);
    b = [[0 for x in range(m)] for y in range(n)];
    for i in range(0,n):
       for j in range(0,m):
          b[i][j]=right[i][j]*left;
    return b;
#end of multiplying a matrix with a constant double

def multiply_m_m(left,right):
   #multiplying two matrisis
   n1=len(left);
   m1=len(left[0]);
   n2=len(right);
   m2=len(right[0]);
   b = [[0 for x in range(n1)] for y in range(m2)];
   if m1 != n2:
     print("uyarı: matris boyutları aynı olmalıdır");
     return b;
   for i in range(0,n1):
      for j in range(0,m2):
         for k in range(0,m1):
            b[i][j]+=left[i][k]*right[k][j];
   return b;

# convert matrix to characteristic polynomial form
def polynomial_coefficients(A):
   n=len(A);
   m=len(A[0]);
   B = [[0 for x in range(m)] for y in range(n)];
   for i in range(0,n):
      for j in range(0,m):
         B[i][j]=A[i][j];
   a = [0.0 for y in range(n+1)];
   C = [[0.0 for x in range(m)] for y in range(n)];
   a[n]=1.0;
   a[n-1]=-trace(B);
   for k in range(n-2,-1,-1):
     C=add(B,a[k+1])
     B=multiply_m_m(A,C)
     a[k]=-trace(B)/(n-k)
   return a
```

```
def eigen_halley(A):
  b=polynomial_coefficients(A);
  if isSymmetric(A):
     x1=1.56435
  else:
     x1=complex(1.56435,0.1)
  return halley(b,x1)

A=[[4,1,1],[0,2,2],[-2,0,9]];
a=polynomial_coefficients(A)
print("a=\n",a)
print("A=\n",A)
print(eigen_halley(A))
A=[[3,1,5],[3,3,1],[4,6,4]]
a=polynomial_coefficients(A)
print("a=\n",a)
print("A=\n",A)
print(eigen_halley(A))
B=[[4,-2,6,4],[-2,3,-1,3],[6,-1,22,13],[4,3,13,46]]
print("B=\n",B)
print(eigen_halley(B))
```

```
runfile('D:/okul/SCO1/eigen_halley.py', wdir='D:/okul/SCO1')
a=
 [-72.0, 64.0, -15, 1.0]
A=
 [[4, 1, 1], [0, 2, 2], [-2, 0, 9]]
[(1.7783225824834872+0j), (4.817630421052082+0j), (8.40404699646443+0j)]
a=
 [-40.0, 4.0, -10, 1.0]
A=
 [[3, 1, 5], [3, 3, 1], [4, 6, 4]]
[(6.118459962316172e-21+2j), (-1.4554068988711813e-22-2j), (10+0j)]
B=
 [[4, -2, 6, 4], [-2, 3, -1, 3], [6, -1, 22, 13], [4, 3, 13, 46]]
[0.6982737832303404, 3.9170557010445033, 17.8405606144007, 52.54410990132446]
```

12.4 BUILT IN EIGENVALUE CALCULATION METHODS IN PYTHON

In NumPy sub programs roots and poly is existed. By using poly characteristic polynomial can be created. It should be note that poly polynomial coeffiecints are in inverse direction compare to **Leverrier** method. In Leverrier polynomial coefficients are given as

$$f(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + .. + a_n x^n$$

$$[a_0, a_1, a_2.., a_n]$$

In NumPy, it is given as

$$f(x) = a_n + a_{n-1} x + a_{n-2} x^2 + a_{n-3} x^3 + .. + a_0 x^n$$

$$[a_0, a_1, a_2.., a_n]$$

After finding characteristic polynomial, roots method will find the eigenvalues

```
# -*- coding: utf-8 -*-
"""
Created on Thu Dec 30 09:20:49 2021

@author:M. Turhan ÇOBAN
Eigen values by using build-in NumPy
numpy.roots(p)  : to find roots of polynomial
numpy.poly(A) : to find polynomial coefficients
  f(x)=a0*x^n+a1*x^n-1+...+an
  inverse polynomial coefficients compare to Leverrier method
"""
import numpy as np

A=[[3,1,5],[3,3,1],[4,6,4]]
a=np.poly(A)
print("a = \n",a)
```

```
x=np.roots(a)
print("eigen values : \n",x)
B=[[4,-2,6,4],[-2,3,-1,3],[6,-1,22,13],[4,3,13,46]]
b=np.poly(B)
print("b = \n",b)
x=np.roots(b)
print("eigen values : \n",x)
```

```
runfile('D:/okul/SCO1/eigen_numpy.py', wdir='D:/okul/SCO1')
a =
 [  1. -10.   4. -40.]
eigen values :
 [ 1.00000000e+01+0.j -4.16333634e-17+2.j -4.16333634e-17-2.j]
b =
 [ 1.000e+00 -7.500e+01  1.265e+03 -4.519e+03  2.564e+03]
eigen values :
 [52.5441099  17.84056061  3.9170557   0.69827378]
```

A second available method is scipy.linalg.eigen, this method directly gives eigen values and vectors.

```
# -*- coding: utf-8 -*-
"""
Created on Thu Dec 30 09:20:49 2021

@author:M. Turhan ÇOBAN
Eigen values by using build-in scipy.linalg.eig
"""
import numpy as np
import scipy.linalg as la

A=[[3,1,5],[3,3,1],[4,6,4]]
x=la.eig(A)
print("eigen values : \n",x[0])
B=[[4,-2,6,4],[-2,3,-1,3],[6,-1,22,13],[4,3,13,46]]
x=la.eig(B)
print("eigen values : \n",x[0])
```

```
runfile('D:/okul/SCO1/eigen_scipy.py', wdir='D:/okul/SCO1')
eigen values :
 [1.00000000e+01+0.j 1.11022302e-16+2.j 1.11022302e-16-2.j]
eigen values :
 [52.5441099 +0.j 17.84056061+0.j  0.69827378+0.j  3.9170557 +0.j]
```

```
# -*- coding: utf-8 -*-
"""
Created on Thu Dec 30 09:20:49 2021

@author:M. Turhan ÇOBAN
Eigen values by using build-in scipy.linalg.eig
"""
import numpy as np
import scipy.linalg as la

A=[[3,1,5],[3,3,1],[4,6,4]]
x=la.eig(A)
print("eigen vectors : \n",x[1])
B=[[4,-2,6,4],[-2,3,-1,3],[6,-1,22,13],[4,3,13,46]]
x=la.eig(B)
print("eigen vectors : \n",x[1])
```

```
runfile('D:/okul/SCO1/eigen_scipy.py', wdir='D:/okul/SCO1')
eigen vectors :
 [[ 0.5767572 +0.j          0.70710678+0.j          0.70710678-0.j        ]
 [ 0.35246273+0.j         -0.35355339-0.35355339j -0.35355339+0.35355339j]
 [ 0.73696754+0.j         -0.35355339+0.35355339j -0.35355339-0.35355339j]]
eigen vectors :
 [[-0.12314029  0.27495958 -0.7282488   0.61553829]
 [-0.04153592 -0.17773297 -0.66904113 -0.72046494]
 [-0.4075391   0.85006077  0.13024455 -0.30715619]
 [-0.90389331 -0.41255897  0.07123215  0.08773797]]
```

12.5 SINGULAR VALUE DECOMPOSITION

Let A be an m x n matrix such that the number of rows m is greater than or equal to the number of columns n. Then there exists:
1) an m x n column orthogonal matrix **U** (a matrix Q is orthogonal if its transpose is equal to its inverse)
2) an n x n diagonal matrix **S,** with positive or zero elements, and

3) an n x n orthogonal matrix V such that:

$$A = USV^T$$

This is the **Singular Value Decomposition (SVD) of matrix A.** If a matrix can be decomposed this way, its inverse value will be existed as:

$$A^{-1} = V[diag(s_1^{-1}, s_2^{-1}, \dots, s_n^{-1}\ )]U^T$$

It should ve note that because of the orhogonal properties of U and V inverses are equal to transpose values. Then a system of equation can be simply solve as $x=A^{-1}b$
Now let us look at these processes in more details with an example

$$A = \begin{bmatrix} 4 & 0 \\ 3 & 5 \end{bmatrix}$$

The singular values of A are the square roots of the eigenvalues of A^TA

$$A^T = \begin{bmatrix} 4 & 3 \\ 0 & -5 \end{bmatrix}$$

$$A^TA = \begin{bmatrix} 4 & 3 \\ 0 & -5 \end{bmatrix}\begin{bmatrix} 4 & 0 \\ 3 & 5 \end{bmatrix} = \begin{bmatrix} 25 & -15 \\ -15 & 25 \end{bmatrix}$$

Eigenvalues of A^TA is

$$\lambda = \begin{bmatrix} 40 \\ 10 \end{bmatrix}$$

Root of the eigenvalues constructs the matrix S, by using eigenvalues as the diagonal of the matrix. And then the inverse can be easily computed by inverting the eigenvalues.

$$S = \begin{bmatrix} \sqrt{40} & 0 \\ 0 & \sqrt{10} \end{bmatrix} = \begin{bmatrix} 6.3245 & 0 \\ 0 & 3.1622 \end{bmatrix}$$

$$S^{-1} = \begin{bmatrix} 1/6.3245 & 0 \\ 0 & 1/3.1622 \end{bmatrix} = \begin{bmatrix} 0.1581 & 0 \\ 0 & 0.3162 \end{bmatrix}$$

The eigenvectors corresponding to these eigenvalues are:

$$[A^TA - \lambda I][X^1] = \begin{bmatrix} 25-40 & -15 \\ -15 & 25-40 \end{bmatrix}\begin{bmatrix} x_1^1 \\ x_2^1 \end{bmatrix} = 0$$

$$[A^TA - \lambda I][X^2] = \begin{bmatrix} 25-10 & -15 \\ -15 & 25-10 \end{bmatrix}\begin{bmatrix} x_1^2 \\ x_2^2 \end{bmatrix} = 0$$

$$[X] = \begin{bmatrix} 0.707106781186548 & 0.707106781186548 \\ -0.707106781186548 & 0.707106781186548 \end{bmatrix}$$

The eigenvector values will be normalized by using Length factor

$$L = \sqrt{(x_1)^2 + (x_2)^2}$$

Then the normalized Value will be V vector

$$[V] = \begin{bmatrix} 0.707106781186548 & 0.707106781186548 \\ -0.707106781186548 & 0.707106781186548 \end{bmatrix}$$

The U vector will be calculated as

$$U = AVS^{-1} = \begin{bmatrix} 0.4472 & 0.8944 \\ 0.8944 & -0.4472 \end{bmatrix}$$

Another example

A	-19	-4.7	6.45
	-4.7	4.6	11.8
	6.45	11.8	-8.3

A^T*A	-19	-4.7	6.45	*	-19	-4.7	6.45	=	424.6925	143.79	-231.545
	-4.7	4.6	11.8		-4.7	4.6	11.8		143.79	182.49	-73.975
	6.45	11.8	-8.3		6.45	11.8	-8.3		-231.545	-73.975	249.7325

λ_1	640.9145
λ_1	134.9733
λ_3	134.9733

S	25.31629	0	0	S^(-1)	0.0395	0	0					
	0	11.6178	0		0	0.086075	0					
	0	0	11.6178		0	0	0.086075					
										total	sqrt(total)	
x	0.783418	-0.02661	0.620926	x^2	0.613743	0.000708	0.385549			1	1	
	0.330646	0.863801	-0.38016		0.109327	0.746153	0.14452			1	1	
	-0.52624	0.503129	0.685515		0.27693	0.253139	0.469931			1	1	
V=	0.783418	-0.02661	0.620926	VT	0.783418	0.330646	-0.52624					
	0.330646	0.863801	-0.38016		-0.02661	0.863801	0.503129					
	-0.52624	0.503129	0.685515		0.620926	-0.38016	0.685515					
U=AVS-1	-19	-4.7	6.45		0.783418	-0.02661	0.620926			-19.8332	-0.30913	-5.58927
	-4.7	4.6	11.8	*	0.330646	0.863801	-0.38016	=		-8.37074	10.03547	3.422
	6.45	11.8	-8.3		-0.52624	0.503129	0.685515			13.32247	5.845259	-6.17067
	-19.8332	-0.30913	-5.58927		0.0395	0	0			-0.78342	-0.02661	-0.4811
	-8.37074	10.03547	3.422	*	0	0.086075	0	=		-0.33065	0.863801	0.294548
	13.32247	5.845259	-6.17067		0	0	0.086075			0.526241	0.503129	-0.53114
$A=USV^T$	-0.78342	-0.02661	-0.4811		25.31629	0	0			-19.8332	-0.30913	-5.58927
	-0.33065	0.863801	0.294548	*	0	11.6178	0	=		-8.37074	10.03547	3.422
	0.526241	0.503129	-0.53114		0	0	11.6178			13.32247	5.845259	-6.17067
	-19.8332	-0.30913	-5.58927		0.783418	0.330646	-0.52624			-19	-4.7	6.45
	-8.37074	10.03547	3.422	*	-0.02661	0.863801	0.503129	=		-4.7	4.6	11.8
	13.32247	5.845259	-6.17067		0.620926	-0.38016	0.685515			6.45	11.8	-8.3

A program is developed to calculate singular value decomposition by using jacobi method. The example problem solved in this program as follows:

$$\begin{bmatrix} 3 & 2 & -1 \\ 2 & -2 & 4 \\ -1 & 0.5 & -1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ -2 \\ 0 \end{bmatrix}$$

Program 9.8.2 Singular value decomposition linear system of equation solving method by using Jacobi eigenvalue method (Python code)

```
# -*- coding: utf-8 -*-
"""
Created on Sun Aug 26 07:28:01 2018

@author: Mustafa Turhan Çoban
"""
from math import *;
from eigen_jacobi import *;
def SVD(A):
    #singular value decomposition
    B=AT(A);
    ATA=multiply_m_m(B,A);
    c=eigen_jacobi(ATA);
    n=len(c[0]);
    n1=len(c);
    dummy=[0.0 for x in range(n1)];
    V = [[0.0 for x in range(n)] for y in range(n)];
    U = [[0.0 for x in range(n)] for y in range(n)];
    xx = [[0.0 for x in range(n)] for y in range(n)];
    x=[0.0 for x in range(n)];
    for i in range(n):
        for k in range(i+1,n):
            if c[0][i]<c[0][k]:
                for j in range(n1):
                    dummy[j]=c[j][k];
                    c[j][k]=c[j][i]
                    c[j][i]=dummy[j];
    S = [[0.0 for x in range(n)] for y in range(n)];
    SI = [[0.0 for x in range(n)] for y in range(n)];
    for i in range(0,n):
        x[i]=c[0][i];
        S[i][i]=sqrt(x[i]);
        SI[i][i]=1.0/S[i][i];
    L=[0.0 for x in range(n)];
```

```
    for i in range(n):
        for j in range(n):
            xx[i][j]=c[i+1][j];
    for j in range(0,n):
        for i in range(0,n):
            L[j]+=xx[i][j]*xx[i][j];

    for i in range(0,n):
        for j in range(0,n):
            V[i][j]=xx[i][j]/L[j];
    VT=AT(V);
    U=multiply_m_m(A,V);
    U=multiply_m_m(U,SI);
    UT=AT(U);
    #AA=multiply_m_m(U,multiply_m_m(S,VT));
    #BB=multiply_m_m(V,multiply_m_m(SI,UT));
    #CC=multiply_m_m(AA,BB);
    x=[S,V,U];
    return x;

def invB(B):
    #inverse matrix (n!=m)
    x=SVD(B);
    S=x[0];
    V=x[1];
    U=x[2];
    SI = invS(S);
    UT=AT(U);
    return multiply_m_m(V,multiply_m_m(SI,UT));

def multiply_m_v(left,right):
    #multiplication of one matrix with one vector
    m1=len(left[0]);
    n1=len(left);
    m2=len(right);
    b = [0.0 for i in range(m2)];
    if n1 != m2:
        print("inner matrix dimensions must agree");
        for ii in range(n1):b[ii]=0;
        return b;
    for i in range(m1):
        b[i]=0;
        for k in range(n1):
            b[i]+=left[i][k]*right[k];

def multiply_v_m(left,right):
    #multiplication of one vector with one matrix
    m2=len(right[0]);
    n2=len(right);
    m1=len(left);
    b = [0.0 for i in range(m1)];
    if n2 != m1:print("inner matrix dimensions must agree");
    for ii in range(n2): b[ii]=0;
    for i in range(m2):
        b[ii]=0;
        for k in range(m1):
            b[i]+=right[i][k]*left[k];
    return b;

def invS(S):
    n=len(S);
    SI = [[0.0 for x in range(n)] for y in range(n)];
    for i in range(0,n):
        SI[i][i]=1.0/S[i][i];
    return SI;

def solve(A,B):
    #solution of the system of equation by using SVD
    AI=invB(A);
    X=multiply_v_m(B,AI);
    return X;

A=[[3,2,-1],[2,-2,4],[-1,0.5,-1]];
C=SVD(A);
S=C[0];
V=C[1];
```

```
U=C[2];
print("S=\n",S);
print("V=\n",V);
print("U=\n",U);
B=[1,-2,0];
X=solve(A,B);
print("X=",X);
```

```
runfile('D:/okul/SCO1/SVD.py', wdir='D:/okul/SCO1')
Reloaded modules: eigen_jacobi
S=
 [[5.126196943738645, 0.0, 0.0], [0.0, 3.734641762863597, 0.0], [0.0, 0.0, 0.1567028942925037]]
V=
 [[0.3470226793918069, 0.8813019270954525, -0.3207525109575496], [-0.4502858087968988, 0.45657424497255894,
0.7673217377501177], [0.8226894616949058, -0.12184774156946693, 0.5552793688681098]]
U=
 [[-0.13307975651355772, 0.9850749406229755, 0.10916565285351665], [0.9530212898129558, 0.0969475577465563,
0.28696967123762784], [-0.2721032884210384, -0.1422270452608084, 0.9516991478544303]]

X= [1.000000000000028, -2.000000000000068, -2.0000000000000497]
```

Built in SVD in python (Same matrix above)

```
# Singular-value decomposition
from scipy.linalg import svd
# define a matrix
A=[[3,2,-1],[2,-2,4],[-1,0.5,-1]]
print(A)
# SVD
U, s, VT = svd(A)
print("U=\n",U)
print("s =\n",s)
print("VT =\n",VT)
```

```
runfile('D:/okul/SCO1/SVD1.py', wdir='D:/okul/SCO1')
[[3, 2, -1], [2, -2, 4], [-1, 0.5, -1]]
U=
 [[ 0.13307976  0.98507494  0.10916565]
 [-0.95302129  0.09694756  0.28696967]
 [ 0.27210329 -0.14222705  0.95169915]]
s =
 [5.12619694 3.73464176 0.15670289]
VT =
 [[-0.34702268  0.45028581 -0.82268946]
 [ 0.88130193  0.45657424 -0.12184774]
 [-0.32075251  0.76732174  0.55527937]]
```

If the interpolating polynomial is written as

$$\boldsymbol{y = c_0 + c_1 x + c_2 x^2 + \cdots + c_N x^N}$$

then the C_i's are required to satisfy the linear equation

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^N \\ 1 & x_1 & x_1^2 & \cdots & x_1^N \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & x_{N-1} & x_{N-1}^2 & \cdots & x_{N-1}^N \\ 1 & x_N & x_N^2 & \cdots & x_N^N \end{bmatrix} \begin{Bmatrix} c_0 \\ c_1 \\ \cdots \\ c_{N-1} \\ c_N \end{Bmatrix} = \begin{Bmatrix} y_0 \\ y_1 \\ \cdots \\ y_{N-1} \\ y_N \end{Bmatrix}$$

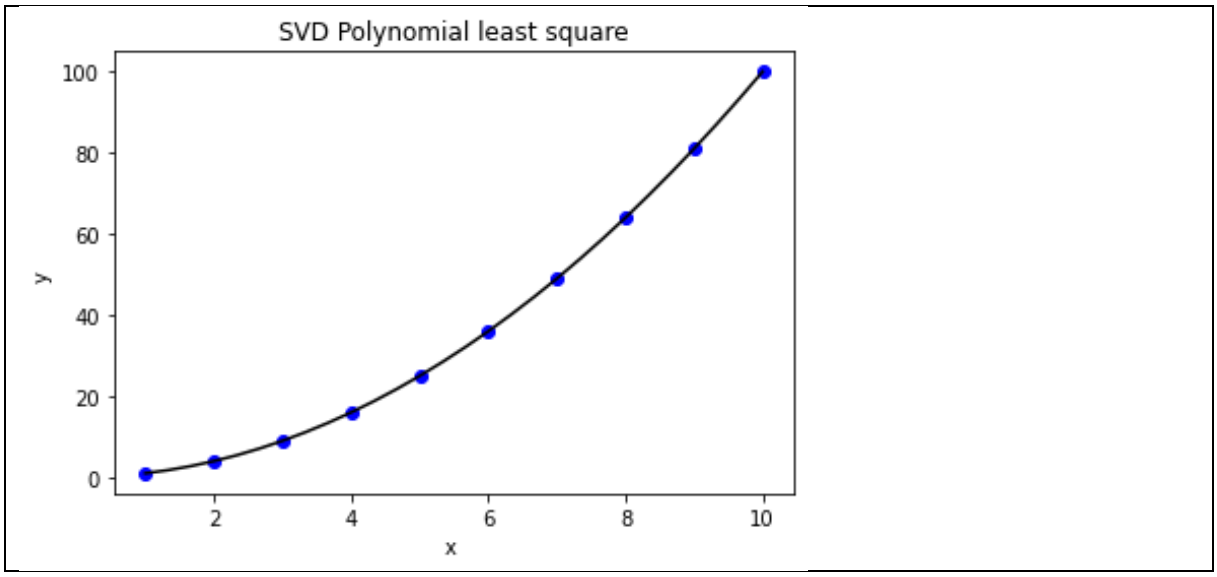
This is a *Vandermonde matrix.* The matrix can be solved by using some of the Matrix solving methods. Limitation of Vandermonde method is the requirement that number of data should be the same as number of polyomial coefficients. But if singular value decomposition method is selected as matrix solution method, this requirement is not valıd anymore, therefore vandermonde interpolation can give the same result with least square method with any degree of polynomial selection desired. Matrix becomes:

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^M \\ 1 & x_1 & x_1^2 & \cdots & x_1^M \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & x_{N-1} & x_{N-1}^2 & \cdots & x_{N-1}^M \\ 1 & x_N & x_N^2 & \cdots & x_N^M \end{bmatrix} \begin{Bmatrix} c_0 \\ c_1 \\ \cdots \\ c_{M-1} \\ c_M \end{Bmatrix} = \begin{Bmatrix} y_0 \\ y_1 \\ \cdots \\ y_{N-1} \\ y_N \end{Bmatrix}$$

```
# -*- coding: utf-8 -*-
"""
Created on Sat Aug 25 09:31:06 2018

@author: Mustafa Turhan Çoban
SINGULAR VALUE DECOMPOSITION LEAST SQUARE CURVE FITTING
poly_least_sqr.py
"""
from math import *
from gauss import *;
import numpy as np;
import matplotlib.pyplot as plt;

def PolynomialLSQ(xi,yi,n):
    #Polynomial least square
    m=len(xi);
    print("l=",m);
    np1=n+1;
    print("np1=",np1);
    A = [[0 for x in range(np1)] for y in range(np1)];
    B =[0 for x in range(np1)];
    X =[0 for x in range(np1)];
    for i in range(np1):
        for j in range(np1):
            if i==0 and j==0:
                A[i][j]=m;
            else:
                for k in range(m):
                    pp=pow(xi[k],(i+j));
                    A[i][j]= A[i][j]+pow(xi[k],(i+j));
        for k in range(m):
            if i==0 and j==0:
                B[i]=  B[i]+yi[k];
            else:
                B[i]= B[i]+pow(xi[k],i)*yi[k];
    X=gauss(A,B);
    return X;

def funcPolynomialLSQ(e,x):
    # this function calculates the value of
    # least square curve fitting function
    n=len(e);
    if n!=0.0:
        ff=e[n-1];
        for i in range(n-2,-1,-1):
            ff=ff*x+e[i];
    else:
        ff=0;
    return ff;

def listPolynomialLSQ(E,xi,aradegersayisi):
     #aradegersayisi:  x--o--o--x--o--o--x zincirinde x deneysel noktalar ise
                 # ara değer sayisi 2 dir
     n=len(xi);
     nn=(n-1)*(aradegersayisi+1)+1;
     z= [[0 for x in range(nn)] for y in range(2)];
     dx=0;
     k=0;
     for i in range(n-1):
         z[0][k]=xi[i];
         z[1][k]=funcPolynomialLSQ(E,z[0][k]);
         k=k+1;
         for j in range(aradegersayisi):
             dx=(xi[i+1]-xi[i])/(aradegersayisi+1.0);
             z[0][k]=z[0][k-1]+dx;
             z[1][k]=funcPolynomialLSQ(E,z[0][k]);
             k=k+1;
```

```
        z[0][k]=xi[i+1];z[1][k]=funcPolynomialLSQ(E,z[0][k]);
    return z;


a=np.loadtxt('a.txt')
print("a",a)
n=len(a)
x =[0 for x in range(n)]
y =[0 for x in range(n)]
for i in range(n):
    x[i]=a[i][0]
    y[i]=a[i][1]
print("x",x)
print("y",y)
E=PolynomialLSQ(x,y,2);
print("E=",E);
Z=listPolynomialLSQ(E,x,2)
print("X=\n",Z[0])
print("Y=\n",Z[1])
plt.title('SVD Polynomial least square')
plt.xlabel('x ')
plt.ylabel('y ')
plt.plot(x,y,'bo',Z[0],Z[1],'k')
```

```
runfile('D:/okul/SCO1/poly_least_sqr.py', wdir='D:/okul/SCO1')
Reloaded modules: gauss
a [[  1.   1.]
 [  2.   4.]
 [  3.   9.]
 [  4.  16.]
 [  5.  25.]
 [  6.  36.]
 [  7.  49.]
 [  8.  64.]
 [  9.  81.]
 [ 10. 100.]]
x [1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0]
y [1.0, 4.0, 9.0, 16.0, 25.0, 36.0, 49.0, 64.0, 81.0, 100.0]
l= 10
np1= 3
E= [0.0, -0.0, 1.0]
X=
 [1.0, 1.3333333333333333, 1.6666666666666665, 2.0, 2.3333333333333335, 2.666666666666667, 3.0,
3.3333333333333335, 3.666666666666667, 4.0, 4.333333333333333, 4.666666666666666, 5.0,
5.333333333333333, 5.666666666666666, 6.0, 6.333333333333333, 6.666666666666666, 7.0,
7.333333333333333, 7.666666666666666, 8.0, 8.333333333333334, 8.666666666666668, 9.0,
9.333333333333334, 9.666666666666668, 10.0]
Y=
 [1.0, 1.7777777777777777, 2.7777777777777772, 4.0, 5.4444444444444455, 7.1111111111111125, 9.0,
11.111111111111112, 13.444444444444446, 16.0, 18.777777777777775, 21.77777777777777, 25.0,
28.444444444444443, 32.11111111111111, 36.0, 40.11111111111111, 44.444444444444436, 49.0,
53.77777777777777, 58.77777777777777, 64.0, 69.44444444444446, 75.11111111111113, 81.0,
87.11111111111113, 93.44444444444447, 100.0]
```



PROBLEMS

PROBLEM 1 The general form for the three-dimensional stress field is given by

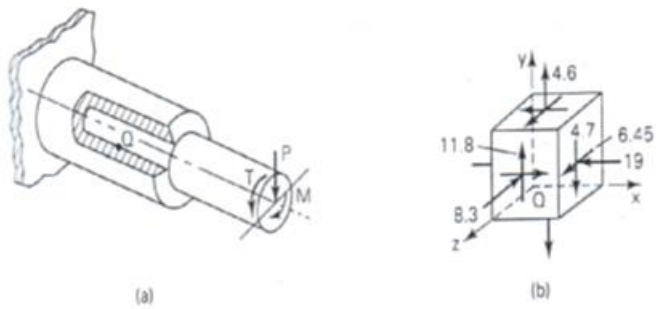
$$\begin{bmatrix} \sigma_{xx} & \sigma_{xy} & \sigma_{xz} \\ \sigma_{yx} & \sigma_{yy} & \sigma_{yz} \\ \sigma_{zx} & \sigma_{zy} & \sigma_{zz} \end{bmatrix}$$

We would like to calculate principle stresses. The principle stresses are eigenvalues of the the three-dimensional stress field. Calculate the principle stresses for

$$\begin{bmatrix} 10 & 14 & 25 \\ 14 & 7 & 15 \\ 25 & 15 & 16 \end{bmatrix}$$

PROBLEM 2

A steel shaft is placed inside of a iron base as shown in the figure . T torsion and M beding moment is applied to the shaft. Under this forces stress matrix at point Q is as follows (MPa)



-19	*-4.7*	*6.45*
-4.7	**4.6**	**11.8**
6.45	**11.8**	**-8.3**

Calculate the principle stresses at point Q

Note : Principle stresses are the eigenvalues of the matrix

PROBLEM 3

$$\boldsymbol{A} = \begin{bmatrix} \mathbf{1} & \mathbf{2} \\ \mathbf{3} & \mathbf{5} \end{bmatrix}$$

Calculate the eigenvalues of the matrix

PROBLEM 4

$$\boldsymbol{A} = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{-1} \\ \mathbf{-1} & \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{-1} & \mathbf{1} \end{bmatrix}$$

Calculate the eigenvalues of the matrix

PROBLEM 5

$$\boldsymbol{A} = \begin{bmatrix} \mathbf{3} & \mathbf{-2} & \mathbf{0} \\ \mathbf{2} & \mathbf{1} & \mathbf{4} \\ \mathbf{1} & \mathbf{-1} & \mathbf{2} \end{bmatrix}$$

Calculate the eigenvalues of the matrix

PROBLEM 6

$$A = \begin{bmatrix} 4 & -1 & 3 & 9 & -5 \\ -1 & 2 & 7 & 12 & 7 \\ 3 & 7 & 0 & -11 & 8 \\ 2 & -4 & 14 & 1 & -5 \\ 11 & -8 & 6 & -2 & 13 \end{bmatrix}$$ Calculate the eigenvalues of the matrix

PROBLEM 7

$$\boldsymbol{A} = \begin{bmatrix} \mathbf{8} & \mathbf{-1} & \mathbf{-5} \\ \mathbf{-4} & \mathbf{4} & \mathbf{-2} \\ \mathbf{18} & \mathbf{-5} & \mathbf{-7} \end{bmatrix}$$

calculate eigenvalues and eigenvectors of he mtrix

PROBLEM 8

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{\sigma_x} & \boldsymbol{\tau_{xy}} & \boldsymbol{\tau_{xz}} \\ \boldsymbol{\tau_{yx}} & \boldsymbol{\sigma_y} & \boldsymbol{\tau_{yz}} \\ \boldsymbol{\tau_{zx}} & \boldsymbol{\tau_{zy}} & \boldsymbol{\sigma_z} \end{bmatrix} = \begin{bmatrix} \mathbf{10} & \mathbf{4} & \mathbf{-6} \\ \mathbf{4} & \mathbf{-6} & \mathbf{8} \\ \mathbf{-6} & \mathbf{8} & \mathbf{14} \end{bmatrix} \boldsymbol{MPa}$$

Normal and shear stresses are given. Find the principle stresses

PROBLEM 9

Solve the following system of equation by using Singular value decomposition method.

$$\begin{bmatrix} 1 & 2 \\ 2 & 3 \\ 4 & 5 \\ 6 & 7 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

PROBLEM 10 Solve the following system of equation by using SVD and Gauss elimination methods, compare the results

$$\begin{bmatrix} 8 & -1 & -5 \\ -4 & 4 & -2 \\ 18 & -5 & -7 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

PROBLEM 11 Solve the roots of polynomial $f(x) = x^4 - 3x^3 + 2x^2 - x + 1$ by using companion matrix and eigenvalue.

PROBLEM 12 Solve the roots of polynomial $f(x) = x^3 - 3x^2 + 3x - 1$ by using companion matrix and eigenvalue.

13. REFERENCES

1. Chapra, Steven C.; Canale, Raymod P., “Numerical Methods for Engineers”, sixth Edition, 2010,McGraw-Hill, ISBN 978-007-126759-5
2. Pres, W. H.,Teulkolsky, S. A.,Vetterling, W. T., Flannery B. P.; “Numerical Recipes in C”, 1992, Cambridge University Pres, ISBN 0-521-43108-5
3. Borse, G.J.;”Numerical Methods with MATLAB” 1997,PWS Publishing company, ISBN 0-534-93822-1
4. Ralston, A.;Rabinowitz, P;”A First Course in Numerical Analysis”, second Edition, 1978, McGraw-Hill, ISBN 007-051158-6
5. Atkinson, K.E.;”An Introduction to Numerical Analysis”, Second edition, 1989, John Wiley & Sons, ISBN 0-471-62489-6
6. Burden, R.L.,Faires,D.F.;”Numerical Analysis”, Eight Edition, 2005,Thomson Brooks/Cole, ISBN 0-534-39200-8
7. Mathews,J.H.,Fink,K.D.;”Numerical Methods Using MATLAB”,1999,Third Edition, Prentice Hall, ISBN 0-13-270042-5
8. Schilling,R.J.,Haris, S. L.”Applied Numerical Methods for Engineers”,2000, Brooks/Cole, ISBN 0-534-37014-4
9. Dahlquist, G.,Anderson N.;”Numerical Methods,1974,Prentice Hall, ISBN 0-13-627315-7
10. Mathews J. H.,”Numerical Methods for Mathematics, Science and Engineering”, Second Edition, 1992, Prentice-Hall, ISBN 0-13-625047-5
11. Michalewicz, Z., “Genetic Algorithms+Data Structures = Evaluation Programs”, second extended edition, 1992, Springer-Verlag, ISBN 3-540-58090-5
12. Karaoğlu, B.,”Sayisal Fizik”,2004, Seyir Yayincilik, ISBN 975-92544-9-2
13. Bayram, M,”Nümerik Analiz”, 2002, “Aktif Yayinevi”, ISBN 975-6755-50-4
14. Tapramaz, R.,”Sayisal Çözümleme”, 2002, Literatür Yayinlari, ISBN 975-8431-82-X
15. Scheid, F.,”SCHAUM’S OUTLINES Nümerik Analiz”, 1988,Nobel Yayim Dağitim, ISBN 975-591-078-6
16. Fox, L., Mayers D. F., “Computing Methods for Scientist and Engineers”, 1968, Oxford University Press
17. Johnson, R. L.,”Numerical Methods A Software Aproach”, 1982, John Willey & Sons
18. Mitchell, M,” An Introduction to Genetic Algorithms”1999, A Bradford Book The MIT Pres, ISBN 0-262-13316-4
19. Arnold, D. N. , “A Concise Introduction to Numerical Analysis”, 1991
20. Nash, J.C. ,”A Compact Methods for Computers Linear Algebra and Function Minimisation”1990, Adam Hilger, ISBN 0-85274-318-1
21. Chenney, W.,Kinkaid, D.;”Numerical Mathematics and Computing”, Third Edition, Brooks/Cole publishing
22. Çoban, M. Turhan, “Java Programlama Kılavuzu, Herkes için ”, 2000, Alfa yayinevi, ISBN 975-316-631-1

23. Deitel & Deitel, Java How to Program, Second Edition, Prentice Hall, http://www.deitel.com, second edition,1998, ISBN 0-13-906249-1

22. Naughton, Patrick, t Schildt, Herbert ;"Java 1.1 The Complete Reference", Second Adition, 1998, Mc Graw Hill, ISBN 0-07-882436-2
23. Zukowski, John; "Mastering Java 1.2 (Renamed Java 2 by sun microsystems)",1998, Sybex publishing, ISBN 0-7821-2180-2
24. Learning Java, Covers Java 1.3, Patrick Niemeyer & Jonathan Knudsen, O'Reilly , 2000, ISBN 1-56592-718-4
25. Java Swing, Robert Eckstein, Marc Loy & Dave Wood, O'Reilly , 1998, ISBN 1-56592-455-X
26. JDBC Database Access with Java, The Java Series, Graham Hamilton, Rick Cattell, Maydene Fisher, Addison Wesley, 1997, ISBN 0-201-30995-5
27. Java Algorithms, Scott Robert Ladd, Mc Graw-Hill, 1998, ISBN 0-07-913696-6
28. Fundamentals of Database Systems, Elmasri & Navathe, Benjamin/Cummings, 1994, ISBN 0-8053-1753-8
29. Thinking in Java, Bruce Eckel, Prentice Hall, 2000, ISBN 013-659723-8
30. VisAD graphic package, http://www.ssec.wisc.edu/~billh/visad.html
31. Java Development kit ve 3D graphic package, http://java.sun.com
32. ALGLIN-NET internet site, (Gauss- Hermit integration) http://www.alglib.net/integral/gq/ghermite.php
33. ALGLIN-NET internet site, (Gauss-Laguerre integration) http://www.alglib.net/integral/gq/glaguerre.php
34. Gander, W. ,W. Gautschi, "Adaptive Quadrature - Revisited", BIT, Vol. 40, 2000, page. 84-101. http://www.inf.ethz.ch/personal/gander
35. H.J. Korsch H.-J. Jodl T. Hartmann,"Chaos, A Program Collection for the PC", Springer publishers, 2008, ISBN 978-3-540-74866-3
36. Forman S. Acton, "Numerical Methods that work", Mathematical Association of America, 1990, ISBN 0-88385-450-3
37. Alfio Quarteroni, Riccardo Sacco, Fausto Saleri, "Numerical Mathematics", Springer and Verlag, 1991, ISBN 0-387-98959-5
38. Perviz Moin, "Fundamentals of Enginering Numerical Analysis", Cambridge publications, 2010, ISBN 978-0-521-88432-7
39. J.H. Verner, Numerically optimal Runge--Kutta pairs with interpolants. Numerical Algorithms. 53, (2010) pp. 383--396.
40. J.H. Verner, Improved Starting methods for two-step Runge--Kutta methods of stage-order p-3, Applied Numerical Mathematics, 10, (2006) pp. 388--396.
41. J.H. Verner, Starting methods for two-step Runge--Kutta methods of stage-order 3 and order 6, J. Computational and Applied Mathematics, 185, (2006) pp. 292--307.
42. T. Macdougall and J.H. Verner, Global error estimators for Order 7,8 Runge--Kutta pairs, *Numerical Algorithms* 31, (2002) pp. 215--231.
43. Z. Jackiewicz and J.H. Verner, Derivation and implementation of two-step Runge-Kutta pairs. *Japan Journal of Industrial and Applied Mathematics* 19 (2002), pp. 227--248.
44. R.W. Lewis, P. Nithiarasu, K. N. Seetheramu, Fundamentals of the Element Method for Heat and Fluid Flow, John Willey & Sons, 2004, ISBN 0-470-84788-3
45. Computational Methods for Algebraic Spline Surfaces , Tor Dokken, Bert Jüttler, 2000, ISBN 3-540-23274-5, Springer Berlin Heidelberg New York
46. Peter Lancaster, Kestutis Salkauskas, Curve and Surface Fitting, an introduction, Academic Press, Harcourt Brace Jovanovich Publishers, 1986, ISNB 0-12-436060-2
47. Bength Fornberg, "Generation of Finite Difference Formulas on Arbitrary Spaced Grids", Mathematics of Computation, Volume 51, Number 184, October 1988, pages 699-706
48. S. Abbasbandy, Improving Newton-Raphson method for nonlinear equations by modified Adomian decomposition method, Appl. Math. Comput. 145 (2003), pp. 887-893.
49. Shenfeng Li, Rujing Wang, Two fourth order Iterative Methods Based on Continued Fraction for Root-finding Problems, World Academy of Science, Engineering and Technology 60 2011
50. Hasan, Mohammed A., On the Derivation of Higher Order Root-finding Methods, Proceedings of the 2007 American Control Conference, ThA07.3, New York City, USA, July 11-13 2007
51. Xin-She Yang.: A New Metaheuristic Bat-Inspired Algorithm. Nature Inspired Cooperative Strategies for Optimization (NICSO 2010). Studies in Computational Intelligence Vol. 284 (2010) 65-74.

52. Xin-She Yang, Engineering Optimisation, An introduction with Metaheuristic applications, John Willey & Sons , 2010, ISBN 978-0-470-58246-6
53. Z. W. Geem, J. H. Kim, and G. V. Loganathan, "A new heuristic optimization algorithm: Harmony search", *Simulation*, 76, 60-68 (2001).
54. Ashrafi, S.M., Dariane, A.B., Performance evaluation of an improved harmony search algorithm for numerical optimization: Melody Search (MS). Eng. Appl. Artif. Intel. (2012), http://dx.doi.org/10.1016/j.engappai.2012.08.005
55. Parikshit Yadav, Rajesh Kumar, S.K. Panda, C.S. Chang, "An Intelligent Tuned Harmony Search algorithm for optimization", Information Sciences 196(2012) 47-72 elsevier publ.
56. Yang, X. S. and Deb, S., 2009. 'Cuckoo search via Levy flights', *Proceeings of World Congress on Nature & Biologically Inspired Computing* (NaBIC 2009,India), IEEE Publications, USA, pp. 210-214.
57. [1] Civicioglu P., Transforming geocentric cartesian coordinates to geodetic coordinates by using differantial search algorithm, Computers & Geosciences 46, 229-247, 2012
58. [2] Vito., T., Elio, T., Kevin, M.P., et al., Swarm cognition: an interdisciplinary approach to the study of self-organising biological collectives, Swarm Intelligence 5, 3-18, 2011
59. [1] Wu, B., Qian, C., Ni, W., Fan, S., Hybrid harmony search and artificial bee colony for global optimization problems, Computers and Mathematics with Applications 64, 2621-2634, 2012
60. [2] Pan, Q., Suganhtan, P.H., Tasgetiren M.F., et al., A self – adaptive global best harmony search algorithm for continuous optimization problems, Appl. Math. Comput. 216, 830-848, 2010
61. [3] Kennedy, J., Eberhart, R.C., Shi, Y., Swarm Intelligence, Morgan Kaufmann Publishers, San Francisco, 2001
62. [1] J, Sun., B, Feng., W, Xu., "Particle Swarm Optimization with Particles Having Quantum Behavior", *IEEE Proc. Of Congress on Evolutionary Computation*, 2004
63. [2] J, Sun., W, Xu., B, Feng., "Global Search Strategy of quantum Behaved Particle Swarm Optimization", *IEEE Proc.of Conference on Cybernatics and Intelligent Systems*, Singapore, 2004
64. [3] van den Bergh, F., A, Engelbrecht., A new locally convergent particle swarm optimizer. In Proceedings of the IEEE Conference on systems, Man, and Cybernatics Hammamet, Tunisia, 1-6, 2002
65. [4] He, S., Q, Wu., J, Wen, J. Saunders., R, Baton., A particle swarm optimizer with passive congregation, Biosystems 78, 135-147, 2004
66. [5] Angeline, P.J., Evolutionary optimization versus particle swarm optimization: Phlosophy and performance differences. Proceedings of the 7th Annual Conference on Evoltionary Programming, San Diego, USA , 601-610, 1998a
67. [6] Xie, X.F., W.J. Zhang ve Z.L. Yang, Adaptive particle swarm optimization on individual level , 6th International Conference on Signal Processing, 1215-1218, 2002a
68. [7] He, Q., C, Han., An improved particle swarm optimization algorithm with disturbance term. Comput. Intell. Bioinfo. 4115, 100-108, 2006
69. [8] Padhye, N., Topology optimization of compliant mechanism using multi-objective particle swarm optimization. Proceedings of the Conference Companion on Genetic and Evolutionary Computation, 2008
70. [9] Jiao , B., Z. Lian and X. Gu, A dynamic inertia weight particle swarm optimization algorithm. Chaos, Solitons Fract. 37, 698-705
71. [10] Engelbracht A., Heteregenous particle swarm optimization. Proceedings of the 7th International conference on Swarm Intelligence, 191-202, 2010
72. [11] Chuang, L.Y., S.W. Tsai., C.H. Yang., Chaotic catfish particle warm optimization for solving global numerical optimization problems, Appl. Math. Comput. 217, 6900-6916, 2011

73. [12] Clerc, M., Kennedy, J., “The particle swarm: explosion, stability, and convergence in a multi-dimensional complex space.” IEEE Transactions on Evolutionary Computation, vol. 6, no.1, pp.58-73, 2002

74. [13] Sun, J., Fang, W., Wu, X.J., Palade, V., Xu, W.B., Quantum-behaved particle swarm optimization: analysis of the individual particle’s behavior and parameter selection, Evolutionary Computation, doi:10.1162/EVCO_a_00049.

75. [14] Su, J., Fang, W., Palade V., Wu, X., Xu, W., Quantum behaved particle swarm optimization with Gaussian distributed local attractor point, Applied Mathematics and Computation 218, 3763-3775, 2011

76. [15] Xi, M., Sun,J., Xu, W., An improved quantum behaved particle swarmoptimization algorithm with weighted mean best position, Applied Mathematics and Computation 205, 751-759, 2008

77. [16] Coelho, L.D.S., A quantum particle swarm optimizer with chaotic mutation operator, Chaos, Solitons & Fractals 37, 1409-1418, 2008

78. [17] Liu B., Wang, L., Jin Y.H., Tang, F., Huang D.X., Directing orbits of chaotic systems by particle swarm optimization, Chaos, Solitons & Fractals 29(2) , 454-461, 2006

79. [18] Zaslavski ,G.M., “The Simplest case of a strange attractor ”,*Phys. Lett. A* 69 (3), 145-147, 1978

80. [19] Yang, X., Shi, P., Shen, W., Jiang, K., Pang, S., Multiobjective Quantum-behaved Particle Swarm Optimization with Entopy Based Density Assesment and Chaotic Mutation Operator, Journal of Computational Information Systems 9: 10 , 3873-3881, 2013

81. [20] Tavazoei, M.S., Haeri, M., Comparison of different one dimensional masps as chaotic search pattern in chaos optimization algorithms, Appl. Math. Comput. 187, 1076-1085, 2007

82. [21] Hilborn, R.C., Chaos and nonlinear dynamics: an introduction for scientists and engineers, 2nd ed. New York: Oxford Univ. Press; 2004

83. [22] May, R.M., Simple mathematical models with very complicated dynamics, Nature 261, 459-467, 1976

84. [23] Li, Y., Deng, S., Xiao, D., A ovel Hash algorithm construction based on chaotic neural network, Neural. Comput. Applic. 20, 133-141, 2011

85. [24] Ott, E., Chaos in dynamical systems, Cambridge, U.K. : Cambridge University Press; 2002

86. [25] Tomida, A.G., Matlab Toolbox and GUI for Analyzing One-Dimensional Chaotic Maps, In: International Conference on Computational Sciences and Its Applications ICCSA, IEEE Press, 321-330, 2008

87. Heat Conduction, M. Necati Özışık, Second Edition, John Wiley & Sons, Inc. ISBN 0-471-53256-8

88. Fundamentals of the Finite Element Method for Heat and Fluid Flow, Roland W. Lewis, Perumal Nithiarasu, Kankanhally N. Seetharamu, Wiley publishers, ISBN 0-470-847-88-3

89. Generation of Finite Difference Formulas on Arbitrary Spaced Grids, Bength Fornberg, Mathematics of Computation, Volume 51, Number 184, October 1988, pp 699-706

90. On the rapid computation on various polylog-arithmetic constants, David H. Bailey, Peter B. Borwein, and simon Plouffe, Mathematics of Computation 66, no 218 (1997) 903-913

91. https://en.wikipedia.org/wiki/Artificial_neural_network

92. A. Askerzadeh, A novel metaheuristic method for solving constrained engineering optimization problems: Crow search algorithm, Computers and Structures, 169, 2016, pp. 1-12.

93. A. A. Heidari, S. Mirjalili, H. Faris, I Aljarah, Harris hawks optimisation:Algorithm and applications, Elsevier Future Generation Computer Systems, 97(2019) p. 849-872

94. A. Kaveh, A. Dadras, A novel meta-heuristic optimization algorithm: Thermal exchange optimization, Elsevier Advances in Engineering Software, 110(2017) 69-84

95. Romberg, W. (1955), "Vereinfachte numerische Integration", Det Kongelige Norske Videnskabers Selskab Forhandlinger, Trondheim, 28 (7): 30–36
96. Shengfeng Li, Fourth-order iterative method without calculating the higher derivatives for nonlinear equations, Journal of Algorithms & computational technology, Volume 13:1-8, DOI:10.1177/1748302619887686
97. F.W.J. Olver, The Evaluation of Zeros of High-Degree polynomials, Philosophical Transactions of the Royal Society of London. Series A. Mathematical and Pyhsical Scinces. Vol. 244. No. 885(Apr 3 1952) pp. 385-415