COMSATS University Islamabad

PROJECT

Subject:

Numerical Computation

Instructor: Dr. Umair Umar

Prepared by:

Name	Registration NO.
Muhammad Usman	SP20-BSM-029
Umair Hassan	SP20-BSM-041

Date: June 20 ,2022

COMSATS University Islamabad

Table of Contents

About the members 3

Chapter 1: Method of non-linear equations 4

i: Bisection method 6

ii: Newton-Raphson method 9

iii: Secant method 11

iv: False position method 14

v: Fixed point method 15

Chapter 2: Methods for system of linear equations 16

i: Gauss Jacobi method 18

ii: Gauss seidel method 19

iii: Successive Over Relaxation methods 20

Chapter 3: Interpolation 21

i: Lagrange Interpolation 23

ii: Newton interpolation 24

iii: Newton divided difference interpolation 26

iv: Spline interpolation 28

Chapter 4: Integration 30

i: Trapezoidal Rule 32

ii: Simpson's (1/3) rule 35

References 37

About the members

This project was done under the supervision of Dr Umair Umar who was the instructor of the course Numerical Computations at the Department of Mathematics, CUI, and Islamabad. The project comprises of all the course contents studied in the semester of spring-2022.

Muhammad Usman alongside Umair Hassan worked upon the project I Usman have done my FSC from Punjab groups of college. For now, I am enrolled in the Undergraduate program of Mathematics at CUI Islamabad. While Umair Hassan completed FSC from Fazaia inter collage and completed my matriculation from Fazaia school of PAF. Now I am enrolled in the Undergraduate program of Mathematics at CUI Islamabad.

In numerical analysis, a numerical method is a mathematical tool designed to solve numerical problems. The implementation of a numerical method with an appropriate convergence check in a programming language is called a numerical algorithm.
It considers all the errors that may affect the result. It considers how much accuracy is required, estimate magnitude of round off. Determine how much iteration is required.
So, there are some methods in numerical analysis

1.Bisection Method
2.Regular false method
3.Newton Raphson method
4.Secant method
5.Fixed point method

1: BISECTION METHOD

The bisection method is an application of the Intermediate Value Theorem (IVT). As such, it is useful in proving the IVT. The IVT states that suppose you have a line segment (between points a and b, inclusive) of a continuous function, and that function crosses a horizontal line. Given these facts, then the intersection of the two lines—point x—must exist

Bisection Method

A numerical method in Mathematics to find a root of a given function
This method based on repeated application of intermediate value property
An equation f(x) =0 where f(x) is real continuous function.
It has at least one root between a and b if f (a) f (b) <0

Algorithm

Follow the below procedure to get the solution for the continuous function:

For any continuous function f(x)

- Find two points, say a and b such that a < b and f(a)* f(b) < 0
- Find the midpoint of a and b, say "t"
- t is the root of the given function if f(t) = 0; else follow the next step
- Divide the interval [a, b] – If f(t)*f(a) <0, there exist a root between t and a
  – else if f(t) *f (b) < 0, there exist a root between t and b
- Repeat above three steps until f(t) = 0.

The bisection method is an approximation method to find the roots of the given equation by repeatedly dividing the interval. This method will divide the interval until the resulting interval is found, which is extremely small.

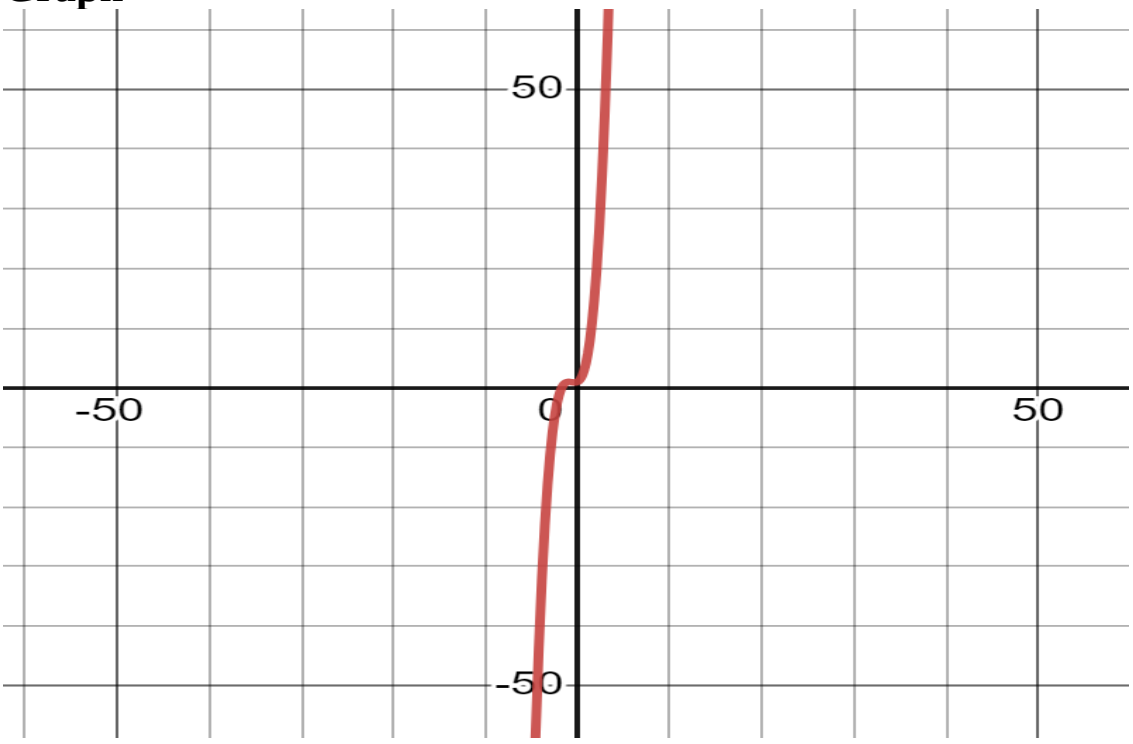
```
from math import sin,cos
def bisection(x0,x1,e):
    step = 1
    condition = True
    while condition:
        x2 = (x0+x1)/2
        print('%d : %d \t\t  %d \t\t %d \t\t %d \t\t %d \t\t %d ' %(step,x0 , x1 , f(x0) , f(x1), x2 , f(x2)))
        if f(x0) * f(x2) < 0:
            x1 = x2
        else:
            x0 = x2
        step = step +1
        condition = abs(f(x2)) > e
    print('root is :%0.8f '%x2)

def f(x):
    return x**5 - 10*x**2 + 13*x - 45

x0 = float(input('first guess: '))
x1 = float(input('second guess: '))
e  = float(input('tolerance: '))

if f(x0) * f(x1) > 0.0:
    print('given guess values do not bracket the root')
else:
    root = bisection(x0,x1,e)
```

Consider a problem

This method based on repeated application of intermediate value property

$f(x)=x^3 + 2x^2+x+1$

Graph



Iteration 1

Step 1:

Here f(0) = -1<0 and f(1)=3>0

∴ Now, Root lies between 0 and 1

Step 2:

$x\_m = (a+b)/2$

$X_0 = (0+1)/2 = 0.5$

Step 3

$f(x_0)$=f(0.5)=$(0.5)^3+2(0.5)^2+(0.5)-1$

= 0.125>0

2nd iteration

Here f(0) = -1<0 and f(0.5)=0.125>0

∴ Now, Root lies between 0 and 0.

$X_1 = ((0+0.5))/2 = 0.25$

$f(x_1)=f(0.25)=(0.25)^3+2(0.25)^2+(0.25)-1=-0.6094<0$

Error approximation

$|\epsilon|=|(0{,}25-0.5)/1.25| * 100=20\%$

None of significant digit are at least correct in the estimate root of x=1.5 because approximate error is greater than 5%

More iteration conducted show in table

Approximate root of the equation $x3+2x2+x-1=0$ using Bisection method is 0.4656 (After 12 iterations)

Table

n	a	f(a)	b	f(b)	c=a+b2	f(c)	Update
1	**0**	**-1**	**1**	**3**	**0.5**	**0.125**	**b=c**
2	**0**	**-1**	**0.5**	**0.125**	**0.25**	**-0.6094**	**a=c**
3	**0.25**	**-0.6094**	**0.5**	**0.125**	**0.375**	**-0.291**	**a=c**
4	**0.375**	**-0.291**	**0.5**	**0.125**	**0.4375**	**-0.0959**	**a=c**
5	**0.4375**	**-0.0959**	**0.5**	**0.125**	**0.4688**	**0.0112**	**b=c**
6	**0.4375**	**-0.0959**	**0.4688**	**0.0112**	**0.4531**	**-0.0432**	**a=c**
7	**0.4531**	**-0.0432**	**0.4688**	**0.0112**	**0.4609**	**-0.0162**	**a=c**
8	**0.4609**	**-0.0162**	**0.4688**	**0.0112**	**0.4648**	**-0.0026**	**a=c**
9	**0.4648**	**-0.0026**	**0.4688**	**0.0112**	**0.4668**	**0.0043**	**b=c**
10	**0.4648**	**-0.0026**	**0.4668**	**0.0043**	**0.4658**	**0.0009**	**b=c**
11	**0.4648**	**-0.0026**	**0.4658**	**0.0009**	**0.4653**	**-0.0008**	**a=c**
12	**0.4653**	**-0.0008**	**0.4658**	**0.0009**	**0.4656**	**0**	**b=c**

2: REGULA FALSE METHOD

The Regula–Falsi Method is a numerical procedure for calculating the roots of a polynomial f. (x). In the Bisection Method, a value x replaces the midpoint and acts as a fresh estimate of a root of f.(x). The goal is to accelerate convergence.

It is method for solving equation in one unknown. It is quite like bisection method algorithm. It was developed because bisection method converges slowly.

Algorithm

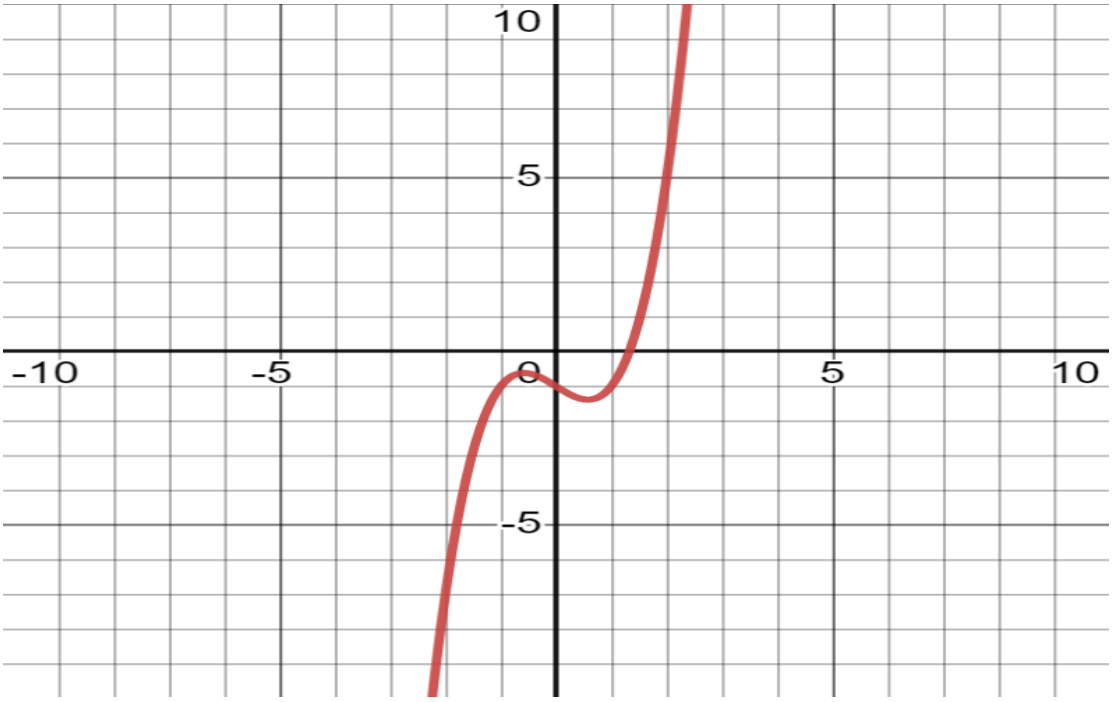
- Choose two guesses for root, such that f(a)f(b)<0
- Estimate root Xm=af(b)-bf(a)/f(b)-f(a)
- Find absolute error |error|=|$(\boldsymbol{X\_m\text{^}new - x\_m\text{^}old})/(\boldsymbol{x\_m\text{^}new})$|*100
- If we get |error|<10-3 then stop however estimated root is 10-3

```
from math import sin ,cos
def reg_falsi(f,x1,x2,tol=1.0e-6,maxfpos=100):
    if f(x1) * f(x2)<0:
        for fpos in range(1,maxfpos+1):
            xh = x2 - (x2-x1)/(f(x2)-f(x1)) * f(x2)
            if abs(f(xh)) < tol:
                break
            elif f(x1) * f(xh) < 0:
                x2 = xh
            else:
                x1 = xh
    else:
        print('No roots exists within the given interval')
    return xh, fpos
y = lambda x: x**5 - 10*x**2 + 13*x - 45
x1 = float(input('enter x1: '))
x2 = float(input('enter x2: '))
r, n = reg_falsi(y,x1,x2)
print('The root = %f at %d false position'%(r,n))
```

$$m = \frac{f(b) - f(a)}{b - a}$$

Find a root of an equation f(x)=x3-x-1 using False Position method

Graph



Solution:
Here x3-x-1=0

Let f(x)=x3-x-1

Here

x	0	1	2
f(x)	-1	-1	5

1st iteration :

Here f(1)=-1<0 and f(2)=5>0

∴ Now, Root lies between x0=1 and x1=2

$x2=x0 - f(x0)\cdot() \frac{x1-x0}{f(x1)-f(x0)}$

x2=1-(-1)·2-1/5-(-1)

x2=1.16667

f(x2)=f(1.16667)=-0.5787<0

Approximate root of the equation x3-x-1=0 using False Position method is 1.32464

Table

n	x0	f(x0)	x1	f(x1)	x2	f(x2)	Update
1	**1**	**-1**	**2**	**5**	**1.16667**	**-0.5787**	**x0=x2**
2	**1.16667**	**-0.5787**	**2**	**5**	**1.25311**	**-0.28536**	**x0=x2**
3	**1.25311**	**-0.28536**	**2**	**5**	**1.29344**	**-0.12954**	**x0=x2**
4	**1.29344**	**-0.12954**	**2**	**5**	**1.31128**	**-0.05659**	**x0=x2**
5	**1.31128**	**-0.05659**	**2**	**5**	**1.31899**	**-0.0243**	**x0=x2**
6	**1.31899**	**-0.0243**	**2**	**5**	**1.32228**	**-0.01036**	**x0=x2**
7	**1.32228**	**-0.01036**	**2**	**5**	**1.32368**	**-0.0044**	**x0=x2**
8	**1.32368**	**-0.0044**	**2**	**5**	**1.32428**	**-0.00187**	**x0=x2**
9	**1.32428**	**-0.00187**	**2**	**5**	**1.32453**	**-0.00079**	**x0=x2**
10	**1.32453**	**-0.00079**	**2**	**5**	**1.32464**	**-0.00034**	**x0=x2**

Compare the Bisection Method with the Regula-Falsi Method

Both methods are similar in that two starting points are required where the signs of the function are different. For *regular falsi,* get the next iterate by joining the line between the points of the function and finding the intersection. The bisection method takes the more primitive bisector of the abscissa. Both methods converge when the function is continuous.
As a rule of thumb, the *regular falsi* method will converge faster provided the function is nearly linear at the zero. That's good. However, with the bisection, you can compute precisely how many iterations are required for a specified accuracy. That's good. Remarkably, however, you can construct functions where either method is faster.

3: Newton Raphson Method

The Newton-Raphson method is a way to quickly find a good approximation for the root of a real-valued function f (x) = 0 f(x) = 0 f(x)=0. It uses the idea that a continuous and differentiable function can be approximated by a straight-line tangent to it.

DISADVANTAGES

- Its convergence is not guaranteed. ...
- Division by zero PROBLEMS can occur.
- Root jumping might take place thereby not getting intended solution.
- Inflection point issue might occur.
- Symbolic derivative is required.
- In case of multiple roots, this method converges slowly.

Algorithm

- We have a function
- Taking derivatives of the function
- Remember derivative must not =0
- Our root is always in +ve and –ve point
- We take only one initial guess between these points
- Formula:Xn+1=Xn-f (xn) \f'(xn)
- X new will be the assign as Xn

```
from math import sin,cos
def newton(fn,dfn,x,tol,maxiter):
    for i in range(maxiter):
        xnew = x - fn(x)/dfn(x)
        print('%d \t\t %0.6f \t\t %0.6f' %( i+1 , x , xnew ))
        if abs(xnew-x)<tol:
            break

        x = xnew
    return xnew, i
y = lambda x: x**5 - 10*x**2 + 13*x - 45
dy = lambda x : 5*x**4 - 20*x + 13
x, n = newton(y, dy,3.5, 0.000001, 100)
print('the root is %.3f at %d iterations.'%(x,n+1))
```

Consider a Question

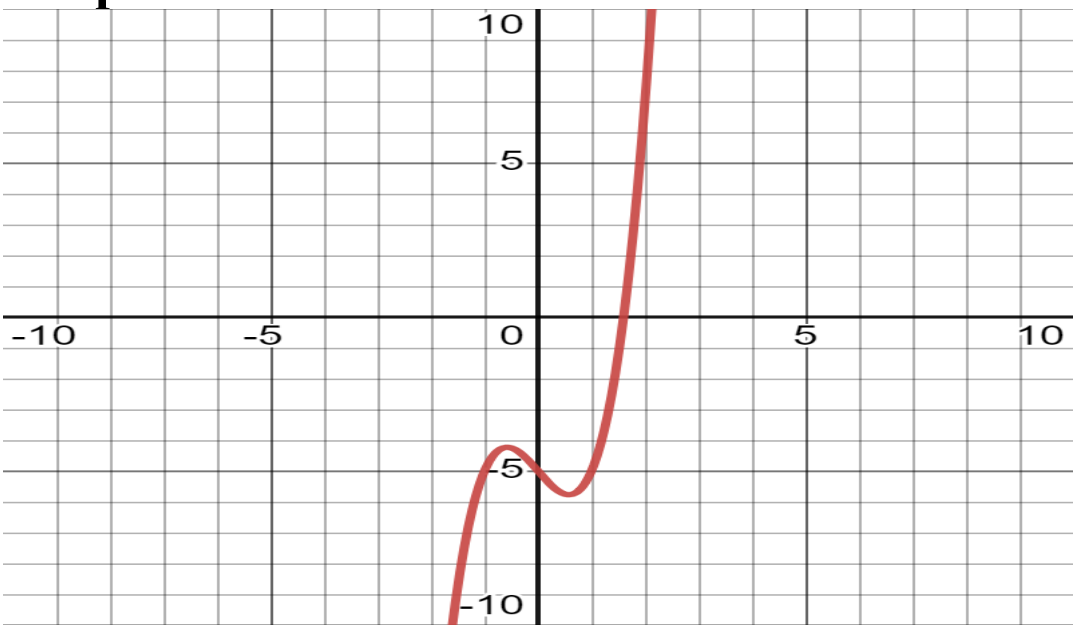
***f*(*x*)=2x^3-2*x*-5**

Find a root of an equation *f*(*x*)=2*x*3-2*x*-5 using Newton Raphson method

Solution:

Here 2x^3-2*x*-5=0

Graph



Let $f(x)=2x^3-2x-5$
$\therefore f'(x)=6x2-2$
Here

x	0	1	2
$f(x)$	**-5**	**-5**	**7**

Here $f(1)=-5<0$ and $f(2)=7>0$
∴ Root lies between 1 and 2
$x0=1+22=1.5$

1*st* iteration :

$f(x0)=f(1.5)=2\cdot1.53-2\cdot1.5-5=-1.25$
$f'(x0)=f'(1.5)=6\cdot1.52-2=11.5$
$x1=x0-f(x0)f'(x0)$
$x1=1.5--1.2511.5$
$x1=1.6087$
Approximate root of the equation 2x3-2x-5=0 using Newton Raphson method is 1.6006

Table

n	x0	f(x0)	f'(x0)	x1	Update
1	**1.5**	**-1.25**	**11.5**	**1.6087**	**x0=x1**
2	**1.6087**	**0.1089**	**13.52741**	**1.60065**	**x0=x1**
3	**1.60065**	**0.00062**	**13.37239**	**1.6006**	**x0=x1**
4	**1.6006**	**0**	**13.37149**	**1.6006**	**x0=x1**

4: Secant Method

The secant method is a root-finding algorithm that uses a succession of roots of secant lines to better approximate a root of a function f. The secant method can be thought of as a finite-difference approximation of Newton's method.
The tangent line to the curve of y = f(x) with the point of tangency (x0, f(x0) was used in Newton's approach. The graph of the tangent line about x = α is essentially the same as the graph of y = f(x) when x0 ≈ α. The root of the tangent line was used to approximate α.

Consider employing an approximating line based on 'interpolation'. Let's pretend we have two root estimations of root α, say, x0 and x1. Then, we have a linear function

$$q(x) = a_0 + a_1 x$$
$$using\ q(x_0) = f(x_0), q(x_1) = f(x_1)$$

Formula: $\boldsymbol{q(x) = \frac{(x_1-x)f(x_0)+(x-x_0)f(x_1)}{x_1-x_0}}$

Algorithm

- We have a function f(x).
- We must find root .
- Our root is always in +ve and –ve point
- We take only one initial guess between these points

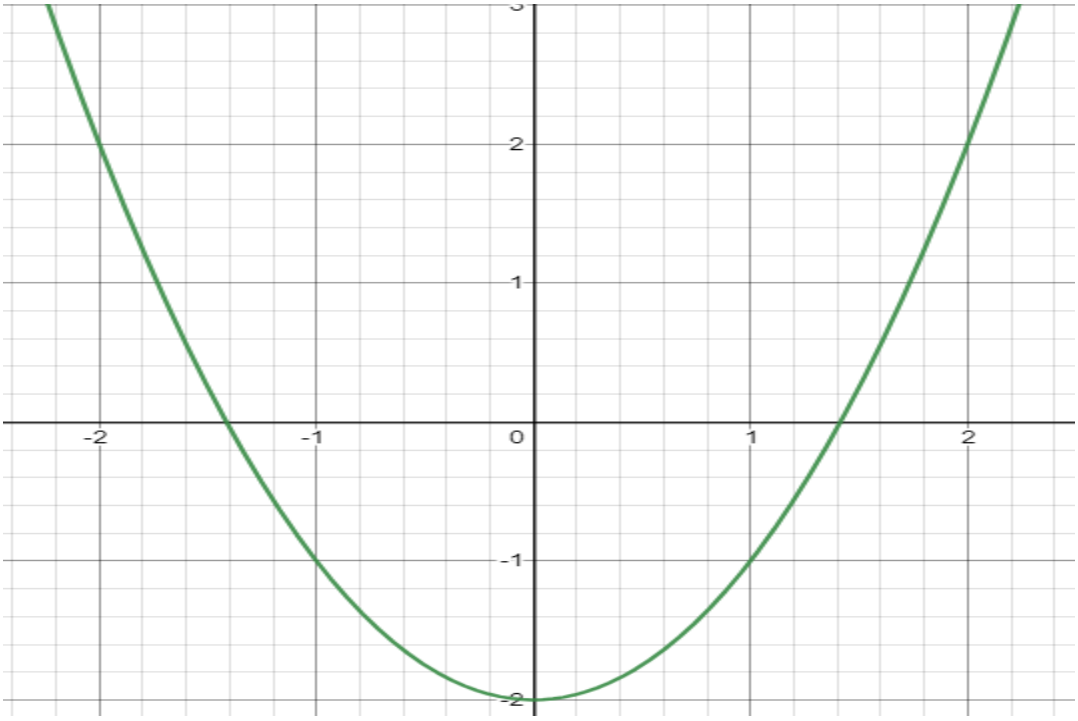
```
from math import sin
def secant(fn,x1,x2,tol,maxiter):
    for i in range(maxiter):
        xnew  = x2 - (x2-x1)/(fn(x2)-fn(x1))*fn(x2)
        print('\t%d \t\t %d \t\t %0.6f' %( x1,x2, xnew ))
        if abs(xnew-x2) < tol:
            break
        else:
            x1 = x2
            x2 = xnew
    else:
        print('warning: Maximum number of iterations is reached')
    return xnew, i+1
f = lambda x: x**5 - 10*x**2 + 13*x - 45

x1 = float(input('enter x1: '))
x2 = float(input('enter x2: '))
r, n = secant(f,x1,x2,1.0e-6,100)
print('Root = %f at %d iterations'%(r,n))
```

Remember denominator should not equal to zero

LET WE HAVE AN EXAMPLE

$$[f(x) = x^2 - 2]$$

Graph



$[f(x) = x\text{^}2 - 2]$

F(1)=-2
F(2)=0
[a,b]=[1,2]
Formula: $(\mathbf{af(b)} - \mathbf{bf(a)})/(\mathbf{f(b)} - \mathbf{f(a)})$
C=a($b\text{^}2 - 2$)-b($a\text{^}2 - 2$)\$b\text{^}2 - 2 - a\text{^}2 + 2$
TABLE

a	b	c	F(c)
1	**2**	**1.3333**	**-0.2223**
2	**1.3333**	**1.4000**	**-0.040**
1.3333	**1.4000**	**1.4146**	**-0.0010**
1.4000	**1.4146**	**1.4142**	**0.0000**
1.4146	**1.4142**	**1.4142**	**0.0000**

Secant Method Advantages and Disadvantages

The secant method has the following advantages:
It converges quicker than a linear rate, making it more convergent than the bisection method.

It does not necessitate the usage of the function's derivative, which is not available in several applications.
Unlike Newton's technique, which requires two function evaluations in every iteration, it only requires one.

The secant method has the following drawbacks:

The secant method may not converge.
The computed iterates have no guaranteed error bounds.
If f0(α) = 0, it is likely to be challenging. This means that when x = α, the x-axis is tangent to the graph of y = f(x).
Newton's approach is more easily generalized to new ways for solving nonlinear simultaneous systems of equations.

5: Fixed Point Method

The **fixed-point iteration** method in numerical analysis is used to find an approximate solution to algebraic and transcendental equations. The fixed-point iteration method uses the concept of a fixed point in a repeated manner to compute the solution of the given equation. A fixed point is a point in the domain of a function g such that g(x) = x. In the fixed-point iteration method, the given function is algebraically converted in the form of g(x) = x.

Algorithm

- Given an equation f(x) = 0
- Convert f(x) = 0 into the form x = g(x)
- Let the initial guess be x0
- xi+1= g(xi) i=1,2,3, …

```
import numpy as np
from math import *
def f(x):
    return x**5 - 10*x**2 + 13*x - 45
def g(x):
    return 1/sqrt(1+x)
def fixedPointIteration(x, e, n):
    step = 1
    flag = 1
    condition = True
    while condition:
        x1 = g(x)
        print('Iteration : %d, x1 = %0.6f and f(x1) = %0.6f' % (step, x1, f(x1)))
        x = x1
        step = step + 1
        if step > n:
            flag=0
            break
        condition = abs(f(x1)) > e
    if flag==1:
        print('\nRequired root is: %0.8f' % x1)
    else:
        print('\nNot Convergent.')
x = float(input('Enter Guess : '))
e = float(input('Tolerable Error : '))
n = int(input('No of Steps : '))
fixedPointIteration(x,e,n)
```

Restriction

f(a)f(b) <0

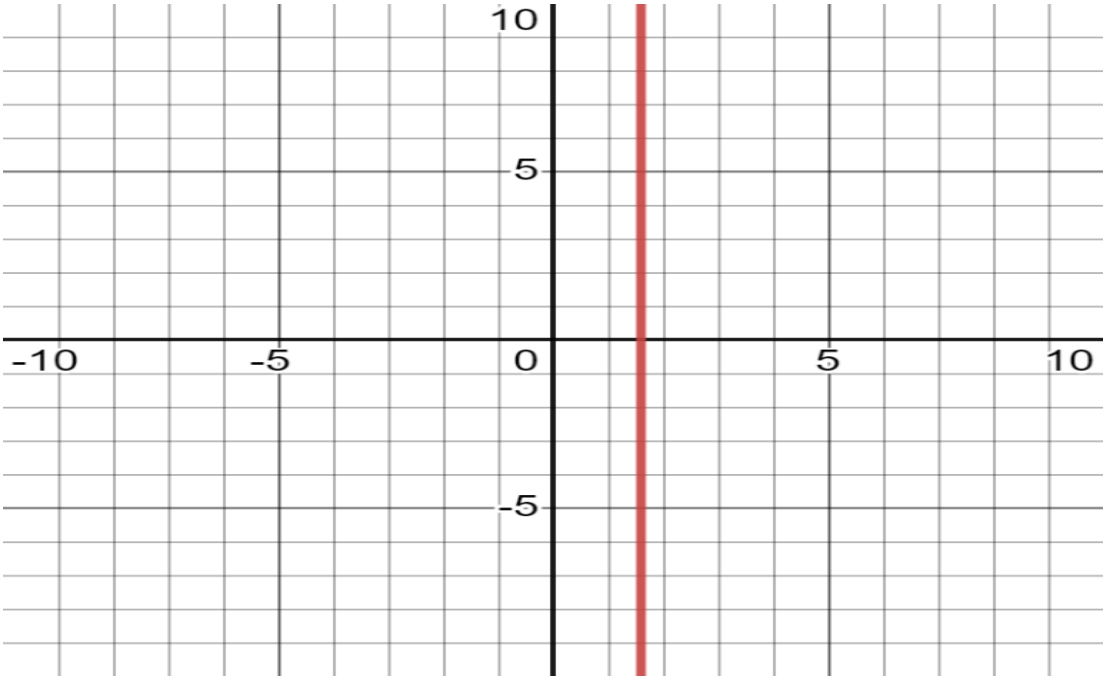
|g'(***x*_0**)|<1

Find the first approximate root of the equation $2x^3 - 2x - 5 = 0$ up to 4 decimal places.

Solution:

Given $f(x) = 2x^3 - 2x - 5 = 0$

Graph



As per the algorithm, we find the value of xo, for which we must find a and b such that f(a) < 0 and f(b) > 0

Now, f(0) = – 5

f(1) = – 5

f(2) = 7

Thus, a = 1 and b = 2

Therefore, xo = (1 + 2)/2 = 1.5

Now, we shall find g(x) such that |g'(x)| < 1 at x = xo

$2x^3 – 2x – 5 = 0$

$\Rightarrow x = [(2x + 5)/2]^{1/3}$

$g(x) = [(2x + 5)/2]^{1/3}$ which satisfies |g'(x)| < 1 at x = 1.5

Now, applying the iterative method xn,= g(xn – 1) for n = 1, 2, 3, 4, 5, …

For n = 1; $x1 = g(xo) = [\{2(1.5) + 5\}/2]^{1/3} = 1.5874$

For n = 2; $x2 = g(x1) = [\{2(1.5874) + 5\}/2]^{1/3} = 1.5989$

For n = 3; $x3 = g(x2) = [\{2(1.5989) + 5\}/2]^{1/3} = 1.60037$

For n = 4; $x4 = g(x3) = [\{2(1.60037) + 5\}/2]^{1/3} = 1.60057$

For n = 5; $x5 = g(x4) = [\{2(1.60057) + 5\}/2]^{1/3} = 1.60059$

For n = 6; x6 = g(x5) = [{2(1.60059) + 5}/2]$^{1/3}$ = 1.600597 ≈ 1.6006

The approximate root of 2x3 – 2x – 5 = 0 by the fixed-point iteration method is 1.6006.

System of linear equation

In the system if linear equations we will solve Jacobi and GuassSiedal methods and their convergence criteria.

Before this we should know the concept of norms

The I1 norm is given by ||X||1 =∑_(i = 1)^n▒ 〖|x|〗

The I2 normor Euclidean norm is given by ||x||2 = ∑_(i = 2)^n▒ 〖|x|〗

The Iinf norm or max norm is given by |x||inf =max┬(0 ≤ x ≤ 1) 〖|x|〗

Convergence Criteria

A sequence of vectors ∏_(k = 1)^inf▒Xk inRnis said to converge to x with respect to norm ||.||

if given any ε>0 and there exists an integer N (ε) such that

ε>0 and there exists an integer N (ε) such that
||xk –x||< ε for all k≥N (ε)

Jacobi Method

Algorithm

Set k=1

While k≤N do step 3-6

For i=1, 2…n

Xi = [$\sum_{j=1}^{n}$(−a XO) + bi

If |x-Xo|<Tol then stop

K=k+1

Then result x1, x2….

Stopping criteria for Jacobi is

```
from pprint import pprint
from numpy import array, zeros, diag, diagflat, dot
def jacobi(A,b,N=25,x=None):
    """Solves the equation Ax=b via the Jacobi iterative method."""
    if x is None:
        x = zeros(len(A[0]))
    D = diag(A)
    R = A - diagflat(D)
    for i in range(N):
        x = (b - dot(R,x)) / D
    return x
A = array([[2.0,1.0],[5.0,7.0]])
b = array([11.0,13.0])
guess = array([1.0,1.0])
sol = jacobi(A,b,N=25,x=guess)
print ("A:")
pprint(A)
print ("B:")
pprint(b)
print ("X:")
pprint(sol)
```

Example

4x+y-z=5

-x+3y+z=-4

2x+2y+5z=1

First find system is strictly dominant

|4|>|1|+|-1|

|3|>|-1|+|1|

|5|>|2|+|2|

Set iteration

$x_{k+1}=\frac{1}{4}$(5-yk+zk)

$y_{K+1}=\frac{1}{3}$(-4+xk-zk)

$Z_{K+1}=\frac{1}{5}$(1-2yk-2zk)

Take initial guess x=0,y=0,z=0

1st iteration

$x_1=\frac{1}{4}$(5+0)=1.25

$x_2=\frac{1}{3}$(-4+0-0)=-1.33

$x_3=\frac{1}{5}$(1-2(0)-2(0))=0.22

Table:

N	x	y	z
1	1.25	-1.3333	0.2
2	1.6333	-0.9833	0.2333
3	1.5542	-0.8667	-0.06
4	1.4517	-0.7953	-0.075
5	1.4301	-0.8244	-0.0626
6	1.4405	-0.8358	-0.0422

7	1.4484	-0.8391	-0.0419
8	1.4482	-0.8357	-0.0451
9	1.4476	-0.8356	-0.045

While implementing Jacobi method then it is not that the the matrix alwaysstrickly dominant then we go for another method we find spectral radius.So for this spectral radius<1.

Spectral radiusthe spectral radius for a square matrix is defined simply as the largest absolute value of its eigenvalue.

We first do D-1(L+U)Xn+D-1b

D^{-1} (L+U)=T

$$D^{-1}=\begin{bmatrix} \frac{1}{VALUE\ OF\ THIS} & 0 & 0 \\ 0 & \frac{1}{VALUE\ OF\ THIS} & 0 \\ 0 & 0 & \frac{1}{VALUE\ OF\ THIS} \end{bmatrix}$$

L=lower triangular matrix

U=upper triangular matrix

After finding T we multiply it with lambda and calculate Eigen value and we will get all Eigen values and they are less than 1.

P(Tj)<1

While implementing Jacobi method then it is not that the matrix always trickly dominant then we go for another method we find spectral radius. So, for this spectral radius<1.
Spectral radius the spectral radius for a square matrix is defined simply as the largest absolute value of its eigenvalue.
We first do D-1(L+U)Xn+D-1b
$D\hat{}(-1)$ (L+U)=T

L=lower triangular matrix
U=upper triangular matrix
After finding T we multiply it with lambda and calculate Eigen value and we will get all Eigen values and they are less than 1.
P(Tj)<1

The Gauss-Seidel Method

Main idea of Gauss-Seidel

With the Jacobi method, the values of $x_i^{(k)}$ obtained in the k_{th} iteration remain unchanged until the entire $(k+1)_{th}$ iteration has been calculated. With the Gauss-Seidel method, we use the new values $x_i^{(k+1)}$ as soon as they are known. For example, once we have computed $x_1^{(k+1)}$ from the first equation, its value is then used in the second equation to obtain the new $x_2^{(k+2)}$ and so on.

Problem

Equations 2x+5y=16,3x+y=11 using Gauss Seidel method

Solution:

Total Equations are 2

2*x*+5*y*=16 ; 3*x*+*y*=11

The coefficient matrix of the given system is not diagonally dominant.
Hence, we re-arrange the equations as follows, such that the elements in the coefficient matrix are diagonally dominant.

3*x*+*y*=11
2*x*+5*y*=16

From the above equations
x=13(11-*y*)
y=15(16-2*x*)

Solution steps are

1*st* Approximation

*x*1=13[11-(0)]=13[11]=3.666667

*y*1=15[16-2(3.666667)]=15[8.666667]=1.733333

2nd Approximation

*x*2=13[11-(1.733333)]=13[9.266667]=3.088889

*y*2=15[16-2(3.088889)]=15[9.822222]=1.964444

3rd Approximation

*x*3=13[11-(1.964444)]=13[9.035556]=3.011852

*y*3=15[16-2(3.011852)]=15[9.976296]=1.995259

4th Approximation

$x4=13[11-(1.995259)]=13[9.004741]=3.00158$

$y4=15[16-2(3.00158)]=15[9.99684]=1.999368$

5th Approximation

$x5=13[11-(1.999368)]=13[9.000632]=3.000211$

$y5=15[16-2(3.000211)]=15[9.999579]=1.999916$

6th Approximation

$x6=13[11-(1.999916)]=13[9.000084]=3.000028$

$y6=15[16-2(3.000028)]=15[9.999944]=1.999989$

Solution By Gauss Seidel Method.
$x=3.000028\cong 3$
$y=1.999989\simeq 2$

Successive over-relaxation(SOR)

The method of successive over-relaxation (SOR) is a version of the Gauss–Seidel method for solving a linear system of equations in numerical linear algebra that results in faster convergence. Any slowly converging iterative process can benefit from a strategy like this.
It was created in 1950 by David M. Young Jr. and Stanley P. Frankel at the same time for the goal of solving linear problems on digital computers automatically. Before Young and Frankel's study, over-relaxation techniques were used. Lewis Fry Richardson's method, as well as R. V. Southwell's methods, are two examples. These approaches, on the other hand, were created for human calculators and required considerable experience to achieve convergence to the solution, rendering them useless for programming on digital computers. These aspects are discussed in the thesis of David M. Young Jr

The iterations of the SOR method

Total Equations are 3
10x+2y-z=7 ; x+8y+3z=-4 ; -2x-y+10z=9

10x+2y-z=7
x+8y+3z=-4
-2x-y+10z=9

From the above equations, First write down the equations for Gauss Seidel method
$X\_(K+1)=1/10(〖7-2Y〗\_K+Z\_K)$
$Y\_(K+1)=1/8$ (-4-$X\_(K+1)$ - $3Z\_K$)
$Z\_(K+1)=1/10$ (9+$2X\_(K+1)$ +$Y\_(K+1)$)
Now multiply the right-hand side by the parameter w and add to it the vector x_k from the previous iteration multiplied by the factor of (1-w)

$xk+1=(1-w)\cdot xk+w\cdot 1/10$ ($〖7-2Y〗\_K+Z\_K$)

$yk+1=(1-w)\cdot yk+w\cdot 1/8$ (-4-$X\_(K+1)$ - $3Z\_K$)

$zk+1=(1-w)\cdot zk+w\cdot 1/10$ (9+$2X\_(K+1)$ +$Y\_(K+1)$)

4. Initial gauss (x,y,z)=(0,0,0) and w=1.25

Solution steps are
1*st* Approximation

x_1=(1-1.25)·0+1.251/10[7-2(0)+(0)]=(-0.25)·0+1.25·110[7]=0+0.875=0.875

y_1=(1-1.25)·0+1.25·1/8[-4-(0.875)-3(0)]=(-0.25)·0+1.25·18[-4.875]=0±0.7617=-0.7617

z_1=(1-1.25)·0+1.25·1/10[9+2(0.875)+(-0.7617)]=(-0.25)·0+1.25·110[9.9883]=0+1.2485=1.2485

Iterations are tabulated as below
Iteration x y Iterations are tabulated as below
Table

Iteration	x	y	z
1	**0.875**	**-0.7617**	**1.2485**
2	**1.0027**	**-1.1765**	**0.9165**
3	**1.033**	**-0.9219**	**1.0389**
4	**0.9771**	**-1.0342**	**0.9803**
5	**1.0118**	**-0.9841**	**1.0099**
6	**0.9943**	**-1.0077**	**0.9951**
7	**1.0027**	**-0.9962**	**1.0024**
8	**0.9987**	**-1.0018**	**0.9988**
9	**1.0007**	**-0.9991**	**1.0006**
10	**0.9997**	**-1.0004**	**0.9997**
11	**1.0002**	**-0.9998**	**1.0001**

INTERPOLATION
NEWTON'S FORWARD INTERPOLATION

Formula:

$$f(a+hu) = f(a) + u\,\Delta f(a) + \frac{u(u-1)}{2!}\,\Delta^{2f(a)} +,,, + \frac{[u(u-1)(u-2)\ldots u-n+1]}{n!}$$

This formula is particularly useful for interpolating the values of f(x) near the beginning of the set of values given. h is called the interval of difference and u $= \frac{(x-a)}{h}$ here a is the first term.

Find Solution of an equation $2x^3 - 4x + 1$ using Newton's Forward Difference formula
x1 = 2 and x2 = 4
x = 2.1
Step value (h) = 0.25
Finding f(2)

Solution:
Equation is $f(x)=2x^3 - 4x + 1$

The value of table for *x* and *y*

x	**2**	**2.25**	**2.5**	**2.75**	**3**	**3.25**	**3.5**	**3.75**	**4**
y	**9**	**14.7812**	**22.25**	**31.5938**	**43**	**56.6562**	**72.75**	**91.4688**	**113**

Newton's forward difference interpolation method to find solution

Newton's forward difference table is

x	y	Δ*y*	Δ2*y*	Δ3*y*	Δ4*y*
2	9				
		5.7812			
2.25	14.7812		1.6875		
		7.4688		0.1875	
2.5	22.25		1.875		0
		9.3438		0.1875	
2.75	31.5938		2.0625		0
		11.4062		0.1875	
3	43		2.25		0
		13.6562		0.1875	
3.25	56.6562		2.4375		0
		16.0938		0.1875	
3.5	72.75		2.625		0
		18.7188		0.1875	
3.75	91.4688		2.8125		
		21.5312			
4	113				

The value of x at you want to find the f(x):x=2.1

h=x1-x0=2.25-2=0.25

p=x-x0h=2.1-20.25=0.4

Newton's forward difference interpolation formula is

$y(x)=y_0+p\Delta y_0+\frac{p(p-1)}{2!}\Delta 2y_0+\frac{p(p-1)(p-2)}{3!}(\Delta 3y_0)+\frac{p(p-1)(p-2)(p-3)}{4!}(\Delta 4y_0)$

$y(2.1)=9+0.4\times 5.7812+\frac{0.4(0.4-1)}{2}(1.6875)\times+\frac{0.4(0.4-1)(0.4-2)}{6}(0.1875)+\frac{0.4(0.4-1)(0.4-2)(0.4-3)}{24}(0)$

y(2.1)=9+2.3125-0.2025+0.012+0

y(2.1)=11.122

Solution of newton's forward interpolation method y(2.1)=11.122

Newton Backward Difference

This is another way of approximating a function with an n^{th} degree polynomial passing through (n+1) equally spaced points.

As a particular case, lets again consider the linear approximation to f(x)

To reduce the number of numerical computations required to compute a large number of interpolated values using the existing interpolation formula,

Example:

Find Solution of an equation $\mathbf{x^3 - x + 1}$ using Newton's Backward Difference formula
x1 = 2 and x2 = 4
x = 3.75
Step value (h) = 0.5
Finding f(2)

Solution:
Equation is f(x)=$x^3 - x + 1$.

The value of **x** and **y**

Table:

x	2	2.5	3	3.5	4
y	7	14.125	25	40.375	61

Newton's backward difference interpolation method to find solution

Newton's backward difference table is

Table

2	7				
		7.125			
2.5	14.125		3.75		
		10.875		0.75	
3	25		4.5		0
		15.375		0.75	
3.5	40.375		5.25		
		20.625			
4	61				

The value of x at you want to find the $f(x)$: $x = 3.75$

h=x_1-x_0=2.5-2=0.5

$p = \frac{x-x_n}{h}$=$\frac{3.75-40}{5}$=-0.5

Newton's backward difference interpolation formula is

$$y(x) = y_n + p\nabla y_n + \frac{p(p+1)}{2!}(\nabla^{2}y_n) + \frac{p(p+1)(p+2)}{3!}(\nabla^{3}y_n) + \frac{p(p+1)(p+2)(p+3)}{4!}(\nabla^{4}y_n)$$

$$y(3.75) = 61 + (-0.5) \times 20.625 + \frac{-0.5(-0.5+1)}{2}(5.25) + \frac{-0.5(-0.5+1)(-0.5+2)}{6}(0.75) + \frac{-0.5(-0.5+1)(-0.5+2)(-0.5+3)}{24}(0)$$

$y(3.75)=61-10.3125-0.6562-0.0469+0$
$y(3.75)=49.9844$
Solution of newton's backward interpolation method y(3.75)=49.9844

Newton's divided difference interpolation formula

This formula is called Newton's Divided Difference Formula. Once we have the divided differences of the function f relative to the tabular points then we can use the above formula to compute f(x) at any non-tabular point.

$$\mathbf{f(x) = f[x_0] + (x - x_0) f[x_0, x_1] + (x - x_0)(x - x_1) f[x_0, x_1, x_2] + \ldots + (x - x_0)(x - x_1) \ldots (x - x_{k-1}) f[x_0, x_1, \ldots, x_k].}$$

Example : Compute **f(0.3)** for the data

x	0	1	3	4	7
f	**1**	**3**	**49**	**129**	**813**

using Newton's divided difference formula.
Solution : Divided difference table

x_i	f_i				
0	**1**				
		2			
1	**3**		**7**		
		23		**3**	
3	**49**		**19**		
		80		**3**	
4	**129**		**37**		
		228			

7 813

Now Newton's divided difference formula is

$$f(x) = f[x_0] + (x - x_0) f[x_0, x_1] + (x - x_0)(x - x_1) f[x_0, x_1, x_2] + (x - x_0)(x - x_1)(x - x_2) f[x_0, x_1, x_2, x_3]$$

$$f(0.3) = 1 + (0.3 - 0)\,2 + (0.3)(0.3 - 1)\,7 + (0.3)(0.3 - 1)(0.3 - 3)\,3$$

$$= 1.831$$

Since the given data is for the polynomial $\mathbf{f(x) = 3x3 - 5x2 + 4x + 1}$ the analytical value is **f(0.3) = 1.831**
The analytical value is matched with the computed value because the given data is for a third-degree polynomial and there are five data points available using which one can approximate any data exactly up to fourth degree polynomial.

Spline Interpolation

Spline interpolation divided into 3 types

- **linear spline**
- **quadratic spline**
- **cubic spline**

Linear Spline Interpolation

Define:
Spline interpolation is a form of interpolation where the interpolant is a special type of piecewise polynomial called a spline.
Given $(x_0,y_0),\ldots\ldots\ldots(x_n,y_n)$ interpolate data to linear spline. Values of x should be in particular order like

$x_0<x_1<\ldots\ldots\ldots x_n$

Values dependent on data point.

$$F(x) = y_i + \frac{y_{i+1} - y_i}{x_{i+1} - x_i}(x - x_i) \quad [x_i < x < x_{i+1}]$$

Draw back

It is not able to use information from other data points .It depends on data points. Derivatives is not continuous , even first derivative in continuous.

Question

Find value of velocity at 16 by using linear spline interpolation formula?

Time	velocity

0	**0**
10	**227.04**
20	**517.35**
15	**362.78**
22.5	**602.97**

First, we arrange in order

Time	**velocity**
0	**0**
10	**227.04**
15	**362.78**
20	**517.35**
22.5	**602.97**

Now we must find v(16) lie in interval [15,20], so

F(16)=$y_{15}+\frac{y_{20}-y_{15}}{20-15}$(x-15)

F(16)=$362.78+\frac{517.35-362.78}{20-15}$(16-15)

F(16)=393.7 m/sec

Quadratic Spline

$$y_1 = a_1x^2 + b_1x + c_1 \qquad x0 < x < x1$$

$$y_2 = a_2x^2 + b_2x + c_2 \qquad x1 < x < x2$$

$$y_n = a_nx^2 + b_nx + c_n \qquad x_{n-1} < x < x_n$$

There are three unknown coefficients in every quadratic spline (a_1,b_1,c_1). Thus there is needed of 3n linear equation derived in the following steps.

1) All given points are part of spline interpolation

There is total of 2n linear equations derived on this step

2) The first derivative of two spline sharing an intermediate point are equal.

   There is total of n-1 linear equation derived on this step

3) The second derivative of 1st spline is assumed to be zero.

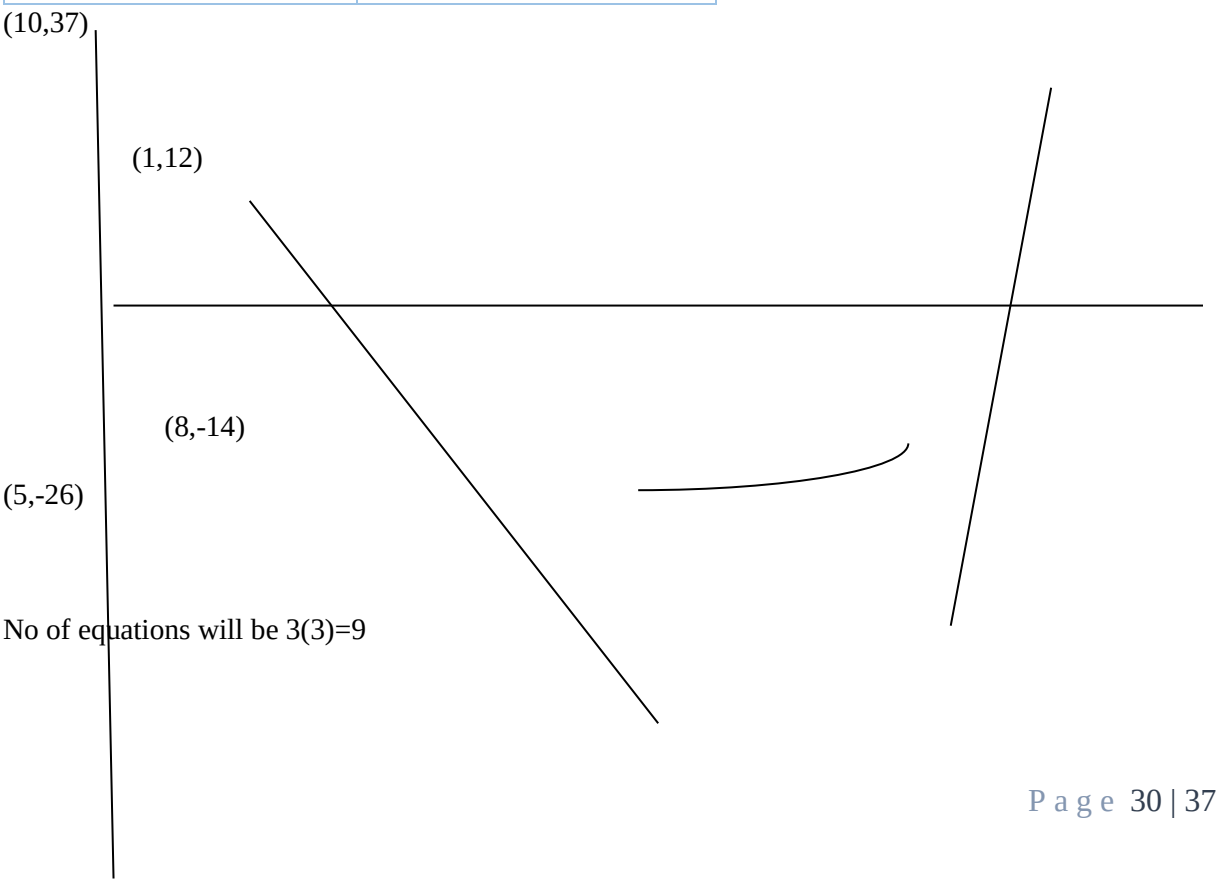
   There is only one equation derived on this step that is $2a_1=0$

   Because of this assumption the first spline is a line.

Question

Find f(7) for given set of data below using quadratic spline interpolation.

1	**12**
5	**-26**
8	**-14**
10	**37**



No of equations will be 3(3)=9

The given set has 4 set of points so .Thus there is need of n=4-1=3 quadratic function to perform interpolation.

$Y_1=a_1x^2+b_1x+c_1 1<x<5$

$Y_2=a_2x^2+b_2x+c_2 \quad 5<x<8$

$Y_{3=}a_3x^2+b_3x_+c_3 \quad 8<x<10$

The system has 3n =3(3)=9 number of unknown . 9 equations needed to solve these unknown.

algorithm

1) 2n=2(3)=6

Point (1,12) is only part of the 1st quadratic spline

$12=a_1(1)^2+b(1)+c_1$

$a_{1+}b_{1+}c_{1=}12$ …….eq 1

Point (5,-26) is part of 1st and 2nd quadratic spline.

$-26=a_125+5b_1+c_1$ …….eq2

$-26=a_225+5b_2+c_2$…….eq3

Points (8,-14) is part of 2nd and 3rd spline

$-14=64a_2+8b_2+c_2$ …….eq4

$-14=64a_3+8b_3+c_3$ …….eq5

Point (10,37) is part of 3rd and 4rth spline

$37=100a_3+10b_3+c_3$…….eq6

2) 1st derivatives of quadratic spline

The first interior point is (5,-26) first and second spline connected to it

$2a_1x+b_1=2a_2x+b_2$

$2a_1x+b_1-2a_2x+b_{2=0}$

$2a_1(5)+b_1-2a_2(5)+b_2=0$

$10a_1+b_1-10a_2+b_2=0$ …….eq7

The other interior point is (8,-14) and 2nd and 3rd spline connected to it

$2a_2x+b_2=2a_3x+b_3$

$2a_2x+b_2-2a_3x+b_{3=0}$

$2a_2(8)+b_1-2a_3(8)+b_3=0$

$16a_1+b_1-16a_2+b_2=0$ …….eq8

3) The second derivative is assumed to be zero

$2a_{1=0}$

$a_{1=0}$ …….eq9

Now

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 25 & 5 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 25 & 5 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 64 & 8 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 64 & 8 & 1= \\ 0 & 0 & 0 & 0 & 0 & 0 & 100 & 10 & 1 \\ 10 & 1 & 0 & -10 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 16 & 1 & 0 & -16 & -1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 12 \\ -26 \\ -26 \\ -14 \\ -14 \\ 37 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Cubic spline

Example:

Calculate Cubic Splines

Y	-5	-4	3

y(0.5)

Solution:

y	**-5**	**-4**	**3**

Cubic spline formula is

$$f(x)=\frac{(xi-x)^3}{6h}M_{i-1}+\frac{(x-xi-1)^3}{6h}M_i+\frac{x_i-1}{h}(y_{i-1}-\frac{h2}{6}M_{i-1})+\frac{x-x_{i-1}}{h}(M_i-\frac{h2}{6}M_i)$$

We have, $M_{i-1}+4M_i+M_{i+1}=\frac{6}{h^2}(y_{i-1}+4y_i+y_{i+1})$,,,,, Eq (2)

Here h=1,n=2

M_0=0,M_2=0

Substitute i=1 in equation (2)

$$M0+4M1+M2=\frac{6}{h^2}\left(y_0-2y_1+y_2\right)$$

⇒0+4M_1+0=61·(-5-2·-4+3)

⇒4M_1=36

⇒M_1=9

Substitute i=1 in equation (1), we get cubic spline in 1*st* interval $\left[x_0,x_1\right]$=[0,1]

$$f_1(x)=\frac{(x1-x)^3}{6h}M0+\frac{(x-x0)^3}{6h}M1+\frac{x1-x}{h}(y_0-\frac{h^2}{6}M0)+\frac{x-x0}{h}(y_1-\frac{h^2}{6}M1)$$

$$f_1(x)=\frac{(1-x)^3}{6}\cdot 0+\frac{(x-0)^3}{6}\cdot 9+\frac{1-x}{1}(-5-\frac{1}{6}\cdot 0)+\frac{x-0}{1}(-4-\frac{1}{6}\cdot 9)$$

$$f_1(x)=12\left(3x^3-x-10\right),\text{for } 0\le x\le 1$$

Substitute i=2 in equation (1), we get cubic spline in 2*nd* interval $\left[x_1,x_2\right]$=[1,2]

$$f_2(x)=\frac{(x2-x)^3}{6h}M1+\frac{(x-x1)^3}{6h}M2+\frac{x2-x}{h}(y_1-\frac{h^2}{6}M1)+\frac{x-x1}{h}(y_2-\frac{h^2}{6}M2)$$

$f_1(x) = \frac{(2-x)^3}{6} \cdot 9 + \frac{(x-1)^3}{6} . 0 + \frac{(2-x)}{1} \left(-4 - \frac{1}{6} \cdot 9\right) + +(x-1)1(3 - \frac{1}{6} \cdot 0)$

$f_2(x) = \frac{1}{2}(-3x^3 + 18x^2 - 19x - 4)$, for $1 \leq x \leq 2$

For $y(0.5)$, $0.5 \in [0,1]$, so substitute $x=0.5$ in $f_1(x)$, we get

$f_1(0.5) = -5.0625$

The Numerical integral

Area bounded by curve f(x) and x-axis between limits a&b . It is denoted by

$I = \int_a^b f(x)dx$ .....eq 1

Divide interval (a,b) into n equal interval with length h (step size) $h = \frac{b-a}{n}$

$a = x_0$

$x_1 = x_0 +$

$x_n = x_n + h$

Equation 1 can be evaluated using

\

Trapezoid Rule

$\int_a^b f(x)dx = h[(\frac{y_0+y_n}{2}) + y_1 + y_2 + \ldots\ldots\ldots y_n]$

Trapezoid rule Formula

$$Area = \int_a^b ydx \approx \frac{1}{2}h[y_0 + 2(y_1 + y_2 + \cdots y_{n-1}) + y_n]$$

$$where\ h = \frac{[b-a]}{n}$$

Question:

$\int_0^6 \frac{dx}{1+x^2}$

h=$\frac{b-a}{n}$=$\frac{6-0}{6}$=1

x_0=0(lower limit)

x_1=x_0+h=0+1=1

x_0	0
x_1	1
x_2	2
x_3	3
x_4	4
x_5	5
x_6	6

Now put all values in formula

$\int_0^6 \frac{dx}{1+x^2}$=h[($\frac{y_0+y_n}{h}$)+$y_1$+$y_2$+…….$y_n$]……eq 1

X	**0**	**1**	**2**	**3**	**4**	**5**	**6**
y	**1**	**0.5**	**0.2**	**0.1**	**0.05**	**0.038**	**0.027**

$\int_0^6 \frac{dx}{1+x^2}$=1[($\frac{1+0.027}{2}$)+0.5+0.2+0.1+0.05+0.038+0.027=1.4285]

Difference between Trapezoidal rule and Riemann Sums rule

In trapezoidal rule, we use trapezoids to approximate the area under the curve whereas in Riemann sums we use rectangles to find area under the curve, in case of integration.

Simpson $\frac{1}{3}$ rule

Simpson's 1/3rd rule is an extension of the trapezoidal rule in which the integrand is approximated by a second-order polynomial. Simpson rule can be derived from the various way using Newton's divided difference polynomial, Lagrange polynomial and the method of coefficients. Simpson's 1/3 rule is defined by:

$$\int_b^a f(x)dx = \frac{h}{3}[(y_0 + y_n) + 4(y_1 + y_3 + y_5 \ldots . + y_{n-1}) + 2(y_2 + y_4 + y_6 \ldots . + y_{n-2})$$

This rule is known as Simpson's **One-third rule**.

Simpson's Rule Example

Example: Evaluate $\int_1^0 e^x\, dx$,by Simpson's ⅓ rule.

Solution:

Let us divide the range [0, 1] into six equal parts by taking h = 1/6.

$x_0 = 0\ then\ y_0 = e^0 = 1$

$x_1 = x_0 + h = e^{\frac{1}{6}} = 1.1813$

$x_2 = x_0 + 2h = \frac{2}{6} = \frac{1}{3}\ \ then\ y_2 = e^{\frac{1}{2}} = 1.3956$

$x_3 = x_0 + 3h = \frac{3}{6} = \frac{1}{2}\ then\ \ y_3 = e^{\frac{1}{2}} = 1.6487$

$x_4 = x_0 + 4h = \frac{4}{6} = \frac{2}{3} then\ y_4 = e^{\frac{2}{3}} = 1.94$

$x_5 = x_0 + 5h = \frac{5}{6}\ \ then\ y_5 = e^{\frac{5}{6}} = 2.3009$

$x_6 = x_0 + 6h = \frac{6}{6} = 1\ then\ y_6 = e^1 = 2.7182$

We know by Simpson's ⅓ rule.

$$\int_b^a f(x)dx = \frac{h}{3}[(y_0 + y_n) + 4(y_1 + y_3 + y_5 \ldots . + y_{n-1}) + 2(y_2 + y_4 + y_6 \ldots . + y_{n-2})$$

Therefore,

$\int_1^0 e^x\, dx = \frac{1}{18}$[(1 + 2.7182) + 4(1.1813 + 1.6487 + 2.3009) + 2(1.39561 + 1.9477)]

)]

= (1/18)[3.7182 + 20.5236 + 6.68662]

= 1.7182 (approx.)

References:

https://sites.google.com/site/knowyourrootsmaxima/introduction/bisectionmethod
https://sites.google.com/site/knowyourrootsmaxima/introduction/newtonmethod
https://www.youtube.com/watch?v=vfEq-WKyVbQ&t=30s
https://www.youtube.com/watch?v=x7m0m5A5EiQ http://article.sapub.org/10.5923.j.ajsp.20170702.01.html
http://compmath-journal.org/dnload/Robin-Kumar-and-Vipan-/CMJV06I06P0290.pdf
https://www.youtube.com/watch?v=8F-IY4oihR4
http://compmath-journal.org/dnload/Robin-Kumar-and-Vipan-/CMJV06I06P0290.pdf
https://www.math.usm.edu/lambers/mat772/fall10/lecture17.pdf
https://dmpeli.math.mcmaster.ca/Matlab/Math4Q3/NumMethods/Lecture2-3.html
https://ece.uwaterloo.ca/~dwharder/NumericalAnalysis/10RootFinding/bisection/examples.html
http://compmath-journal.org/dnload/Robin-Kumar-and-Vipan-/CMJV06I06P0290.pdf
https://files.eric.ed.gov/fulltext/EJ1231189.pdf