PROJET

Numerical Computation

Instructor:

Dr. Umair Umar

Prepared by:

NAME	***REG. NO***
Muhammad Abuzar	**SP20-BSM-024**
Esha Mumtaz	**SP20-BSM-012**

Date:

June 21 ,2022

Table of Contents

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 Abuzar alongside Esha Mumtaz worked upon the project. I Esha have done my FSC from Punjab groups of colleges. For now, I am enrolled in the Undergraduate program of Mathematics at CUI Islamabad. While Muhammad Abuzar completed FSC from FG Quaid e Azam inter college and completed my matriculation from Siddeeq public school . NOW I am enrolled in the Undergraduate program of Mathematics at CUI Islamabad.

Some methods in numerical analysis:

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

BISECTION METHOD:

The bisection method is used to find the roots of a polynomial equation. It separates the interval and subdivides the interval in which the root of the equation lies. The principle behind this method is the intermediate theorem for continuous functions. It works by narrowing the gap between the positive and negative intervals until it closes in on the correct answer. This method narrows the gap by taking the average of the positive and negative intervals. It is a simple method and it is relatively slow. The bisection method is also known as interval halving method, root-finding method, binary search method or dichotomy method.
Let us consider a continuous function "f" which is defined on the closed interval [a, b], is given with f(a) and f(b) of different signs. Then by intermediate theorem, there exists a point x belong to (a, b) for which f(x) = 0.

Algorithm

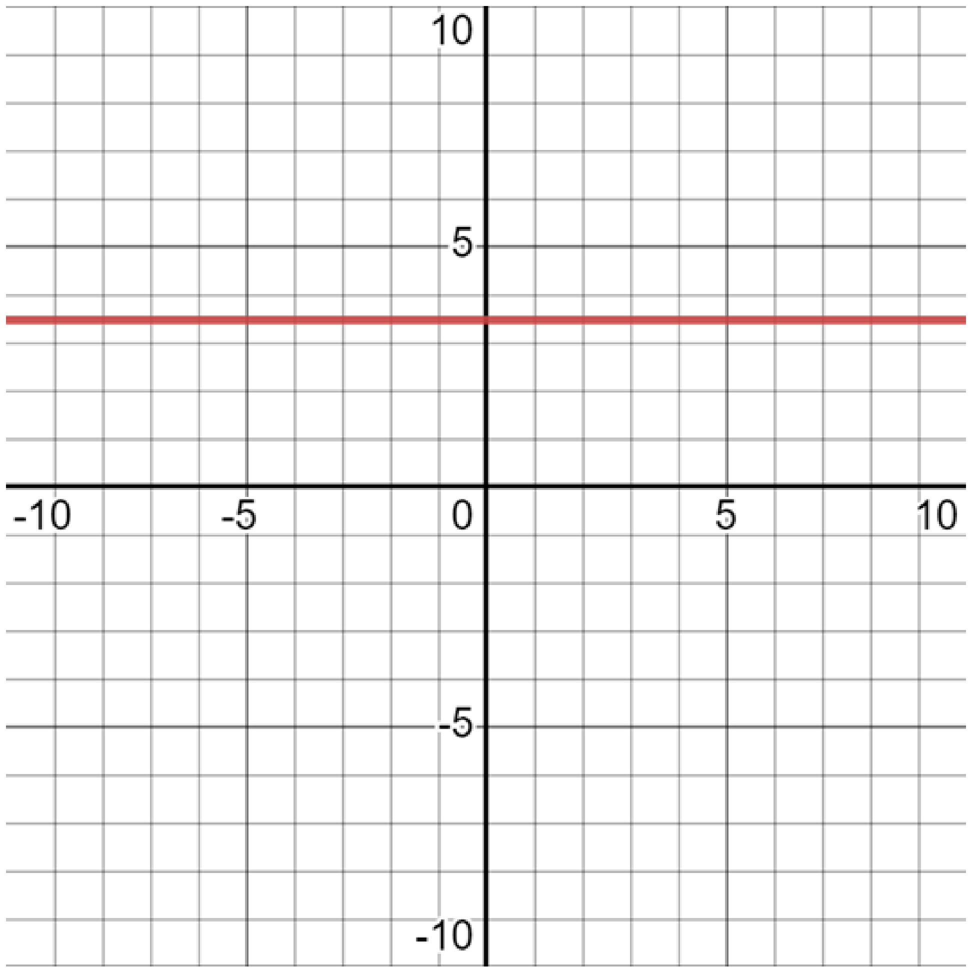
Bisection method Steps (Rule)	
Step-1:	Find points a and b such that $a<b$ and $f(a)\cdot f(b)<0$.
Step-2:	Take the interval $[a,b]$ and find next value $x0=a+b2$

Step-3:	If $f(x0)=0$ then $x0$ is an exact root,
	else if $f(a)\cdot f(x0)<0$ then $b=x0$,
	else if $f(x0)\cdot f(b)<0$ then $a=x0$.

Step-4:	Repeat steps 2 & 3 until $f(xi)=0$ or $\lvert f(xi)\rvert\le$Accuracy

EXAMPLE:

$f(x)=\sqrt{12}$



Find a root of an equation $f(x)=\sqrt{12}$ using Bisection method

Solution:
Here $\sqrt{12}=0$

Let $f(x)=\sqrt{12}$

$x=\sqrt{12}$

$\therefore x2=12$

$\therefore x2-12=0$

Here

x	0	1	2	3	4
$f(x)$	-12	-11	-8	-3	4

<u>$1st$ iteration :</u>

<u>Here</u> $f(3)=-3<0$ and $f(4)=4>0$

$\therefore$ Now, Root lies between 3 and 4

$x0=3+42=3.5$

$f(x0)=f(3.5)=3.52-12=0.25>0$

Approximate root of the equation $x2-12=0$ using Bisection method is 3.4641 (After 12 iterations)

n	a	$f(a)$	b	$f(b)$	$c=a+b2$	$f(c)$	**Update**
1	3	-3	4	4	3.5	0.25	$b=c$
2	3	-3	3.5	0.25	3.25	-1.4375	$a=c$
3	3.25	-1.4375	3.5	0.25	3.375	-0.6094	$a=c$
4	3.375	-0.6094	3.5	0.25	3.4375	-0.1836	$a=c$
5	3.4375	-0.1836	3.5	0.25	3.4688	0.0322	$b=c$
6	3.4375	-0.1836	3.4688	0.0322	3.4531	-0.0759	$a=c$
7	3.4531	-0.0759	3.4688	0.0322	3.4609	-0.0219	$a=c$
8	3.4609	-0.0219	3.4688	0.0322	3.4648	0.0051	$b=c$
9	3.4609	-0.0219	3.4648	0.0051	3.4629	-0.0084	$a=c$
10	3.4629	-0.0084	3.4648	0.0051	3.4639	-0.0016	$a=c$
11	3.4639	-0.0016	3.4648	0.0051	3.4644	0.0018	$b=c$
12	3.4639	-0.0016	3.4644	0.0018	3.4641	0.0001	$b=c$

FALSE METHOD:

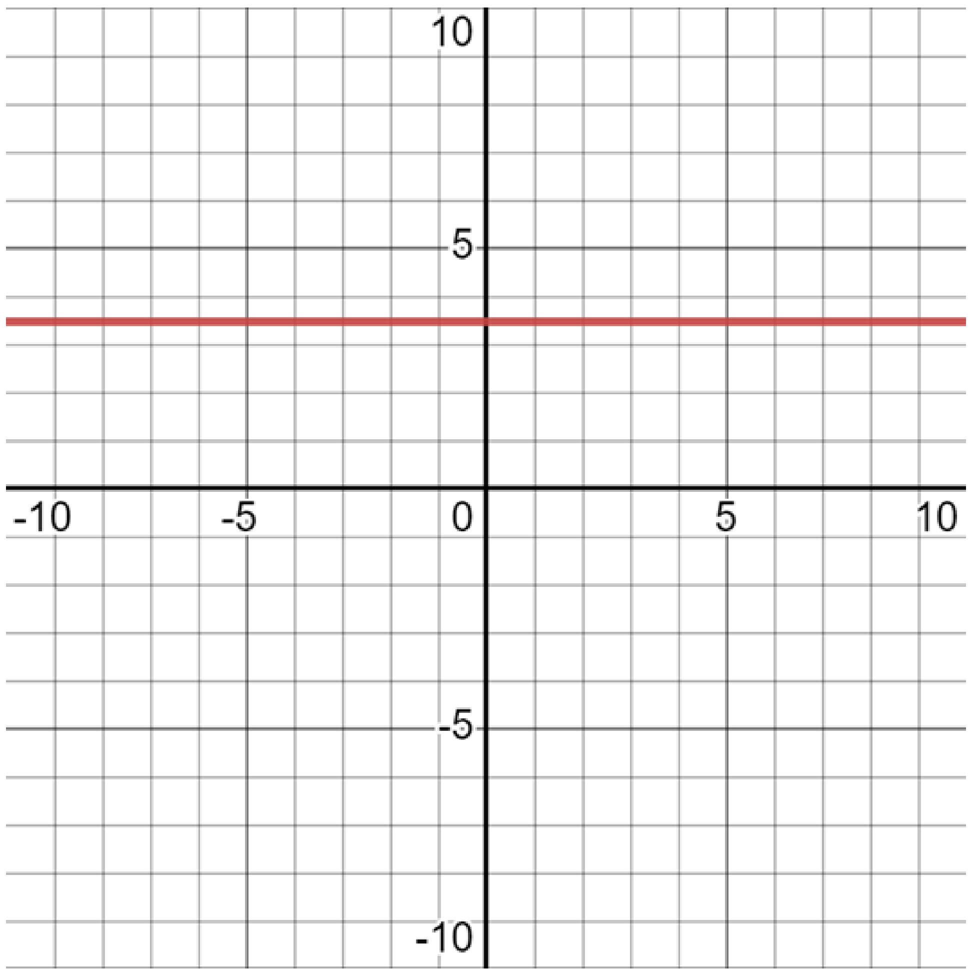
In mathematics, an ancient method of solving an equation in one variable is the **false position method** (method of false position) or **regula falsi method**. In simple words, the method is described as the trial and error approach of using “false” or “test” values for the variable and then altering the test value according to the result.

Algorithm:

False Position method (regula falsi method) Steps (Rule)	
Step-1:	Find points $x0$ and $x1$ such that $x0<x1$ and $f(x0)\cdot f(x1)<0$.
Step-2:	Take the interval $[x0,x1]$ and find next value using Formula-1 : $x2=x0-f(x0)\cdot x1-x0f(x1)-f(x0)$ or Formula-2 : $x2=x0\cdot f(x1)-x1\cdot f(x0)f(x1)-f(x0)$ or Formula-3 : $x2=x1-f(x1)\cdot x1-x0f(x1)-f(x0)$ (Using any of the formula, you will get same x2 value)
Step-3:	If $f(x2)=0$ then $x2$ is an exact root, else if $f(x0)\cdot f(x2)<0$ then $x1=x2$, else if $f(x2)\cdot f(x1)<0$ then $x0=x2$.
Step-4:	Repeat steps 2 & 3 until $f(xi)=0$ or $\lvert f(xi)\rvert\leq \text{Accuracy}$

EXAMPLE:

***f*(*x*)=√12**



Find a root of an equation *f*(*x*)=√12 using False Position method (regula falsi method)

Solution:
Here √12=0

Let *f*(*x*)=√12

x=√12

∴*x*2=12

∴*x*2-12=0

Here

x	0	1	2	3	4
$f(x)$	-12	-11	-8	-3	4

1st iteration :

Here $f(3)=-3<0$ and $f(4)=4>0$

$\therefore$ Now, Root lies between $x0=3$ and $x1=4$

$x2=x0-f(x0)\cdot x1-x0f(x1)-f(x0)$

$x2=3-(-3)\cdot 4-34-(-3)$

$x2=3.4286$

$f(x2)=f(3.4286)=3.42862-12=-0.2449<0$

Approximate root of the equation $x2-12=0$ using False Position method is 3.4641 (After 4 iterations)

n	$x0$	$f(x0)$	$x1$	$f(x1)$	$x2$	$f(x2)$	Update
1	3	-3	4	4	3.4286	-0.2449	$x0=x2$
2	3.4286	-0.2449	4	4	3.4615	-0.0178	$x0=x2$
3	3.4615	-0.0178	4	4	3.4639	-0.0013	$x0=x2$
4	3.4639	-0.0013	4	4	3.4641	0	$x0=x2$

DIFFERENCE:
The difference between bisection method and false-position method is that in bisection method, both limits of the interval have to change. This is not the case for false position method, where one limit may stay fixed throughout the computation while the other guess converges on the root.

SECANT METHOD:

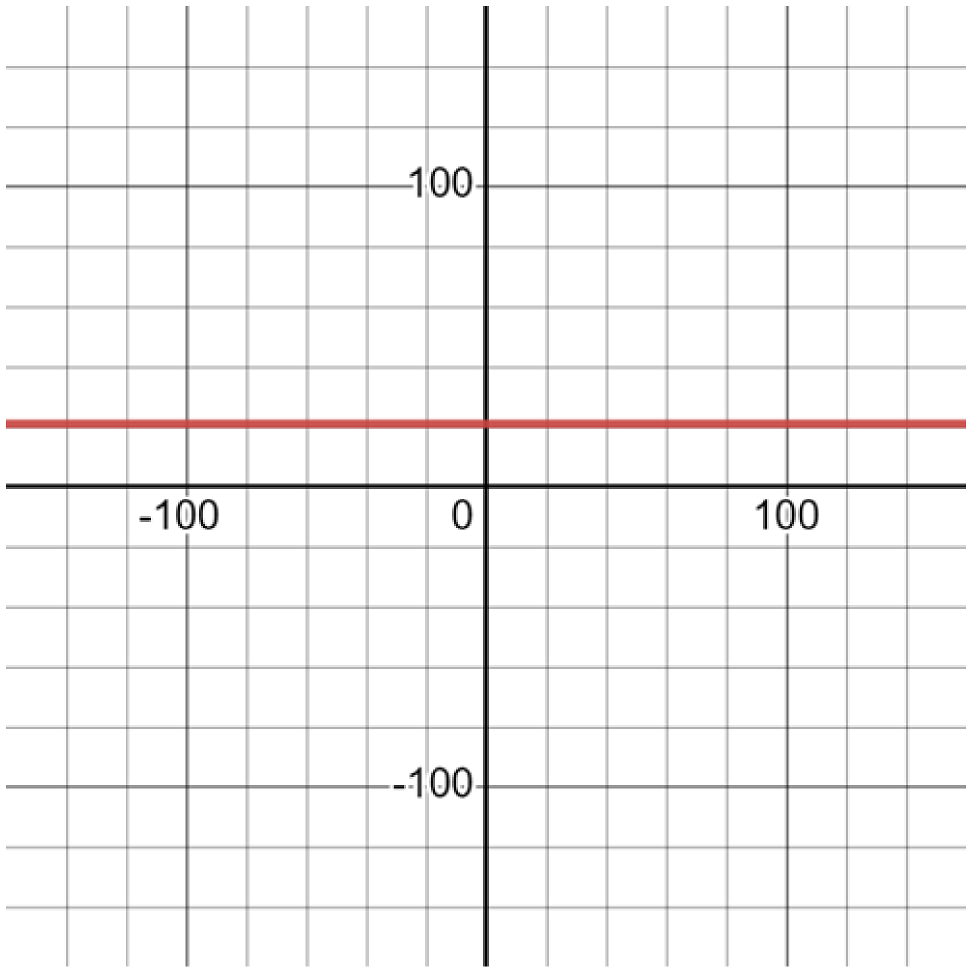
The **secant method** is a root-finding procedure in numerical analysis that uses a series of roots of secant lines to better approximate a root of a function f.

Algorithm:

Secant method Steps (Rule)

Step-1:	Find points $x0$ and $x1$ such that $x0<x1$ and $f(x0)\cdot f(x1)<0$.
Step-2:	find next value using Formula-1 : $x2=x0-f(x0)\cdot x1-x0f(x1)-f(x0)$ or Formula-2 : $x2=x0\cdot f(x1)-x1\cdot f(x0)f(x1)-f(x0)$ or Formula-3 : $x2=x1-f(x1)\cdot x1-x0f(x1)-f(x0)$ (Using any of the formula, you will get same x2 value)
Step-3:	If $f(x2)=0$ then $x2$ is an exact root, else $x0=x1$ and $x1=x2$
Step-4:	Repeat steps 2 & 3 until $f(xi)=0$ or $\|f(xi)\|\leq$Accuracy

$\boldsymbol{f(x)=3\surd 48}$



Find a root of an equation $f(x)=cube48\surd$ using Secant method

Solution:
Here $4813=0$

Let $f(x)=4813$

x=$cube48\surd$

$\therefore x3=48$

$\therefore x3\text{-}48=0$

Here

x	0	1	2	3	4
$f(x)$	-48	-47	-40	-21	16

<u>$1st$ iteration :</u>

<u>x</u>$0=3$ and $x1=4$

f$(x0)=f(3)=\text{-}21$ and $f(x1)=f(4)=16$

$\therefore x2=x0\text{-}f(x0)\cdot x1\text{-}x0f(x1)\text{-}f(x0)$

x2$=3\text{-}(\text{-}21)\cdot 4\text{-}316\text{-}(\text{-}21)$

x2$=3.5676$

$\therefore f(x2)=f(3.5676)=3.56763\text{-}48=\text{-}2.5936$

Approximate root of the equation $x3\text{-}48=0$ using Secant method is 3.6342 (After 4 iterations)

n	$x0$	$f(x0)$	$x1$	$f(x1)$	$x2$	$f(x2)$	**Update**
1	3	-21	4	16	3.5676	-2.5936	$x0=x1$ x1$=x2$
2	4	16	3.5676	-2.5936	3.6279	-0.2513	$x0=x1$ x1$=x2$
3	3.5676	-2.5936	3.6279	-0.2513	3.6344	0.0047	$x0=x1$ x1$=x2$

4	3.6279	-0.2513	3.6344	0.0047	3.6342	0	$x0=x1$ $x1=x2$

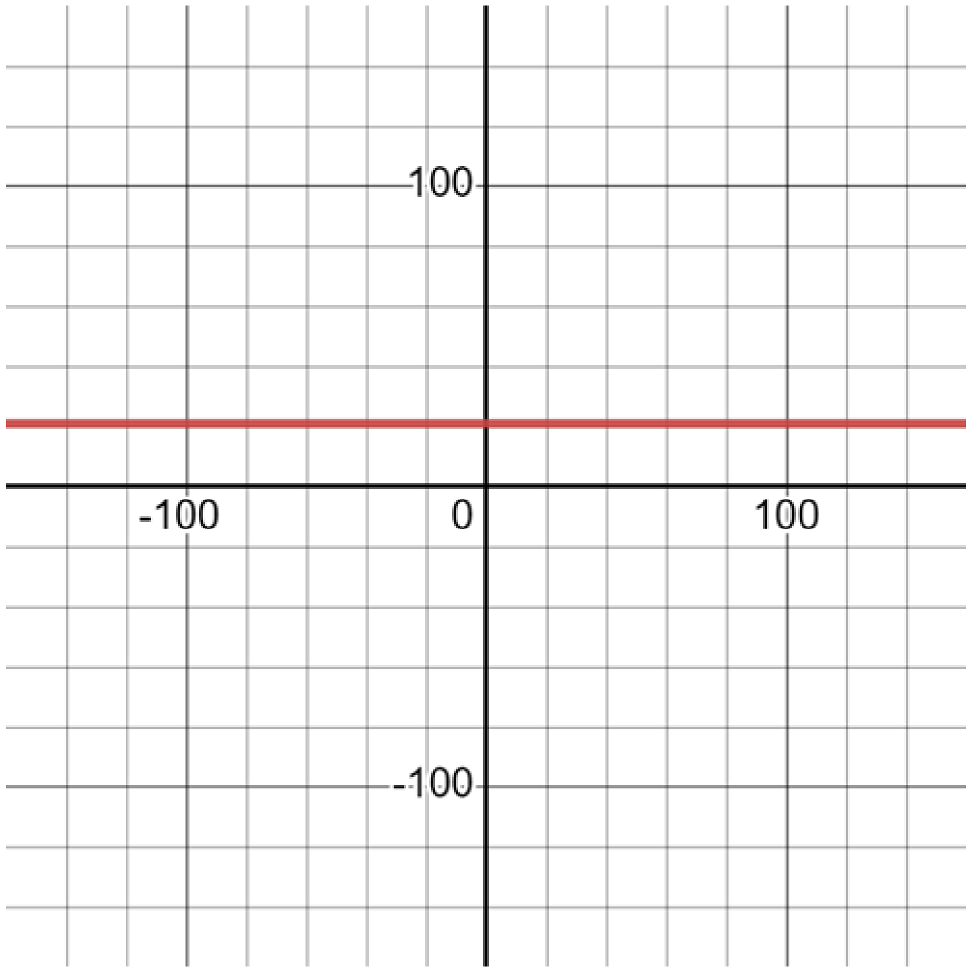
Newton Raphson method:

In numerical analysis, Newton's method, also known as the Newton–Raphson method, named after Isaac Newton and Joseph Raphson, is a root-finding algorithm which produces successively better approximations to .the roots (or zeroes) of a real-valued function

Algorithm

Newton Raphson method Steps (Rule)	
Step-1:	Find points a and b such that $a<b$ and $f(a)\cdot f(b)<0$.
Step-2:	Take the interval $[a,b]$ and find next value $x0=a+b2$
Step-3:	Find $f(x0)$ and $f'(x0)$ $x1=x0-f(x0)f'(x0)$
Step-4:	If $f(x1)=0$ then $x1$ is an exact root, else $x0=x1$
Step-5:	Repeat steps 3 and 4 until $f(xi)=0$ or $\|f(xi)\|\leq \text{Accuracy}$

***f*(*x*)=3√48**



Find a root of an equation *f*(*x*)=*cube*48√ using Newton Raphson method

Solution:
Here 4813=0

Let *f*(*x*)=4813

x=*cube*48√

∴*x*3=48

∴*x*3-48=0

d*dx*(*x*3-48)=3*x*2
∴*f*′(*x*)=3*x*2

Here

x	0	1	2	3	4
***f*(*x*)**	-48	-47	-40	-21	16

Here $f(3)=-21<0 \text{and} f(4)=16>0$

∴ Root lies between 3 and 4

$x0=3+42=3.5$

$x0=3.5$

1*st* iteration :

$f(x0)=f(3.5)=3.53-48=-5.125$

$f'(x0)=f'(3.5)=3 \cdot 3.52=36.75$

$x1=x0-f(x0)f'(x0)$

$x1=3.5--5.12536.75$

$x1=3.6395$

Approximate root of the equation $x3-48=0$ using Newton Raphson method is 3.6342 (After 3 iterations)

n	***x*0**	***f*(*x*0)**	***f'*(*x*0)**	***x*1**	**Update**
1	3.5	-5.125	36.75	3.6395	$x0=x1$
2	3.6395	0.2069	39.7369	3.6342	$x0=x1$
3	3.6342	0.0003	39.6233	3.6342	$x0=x1$

FIXED POINT ITERATION METHOD:

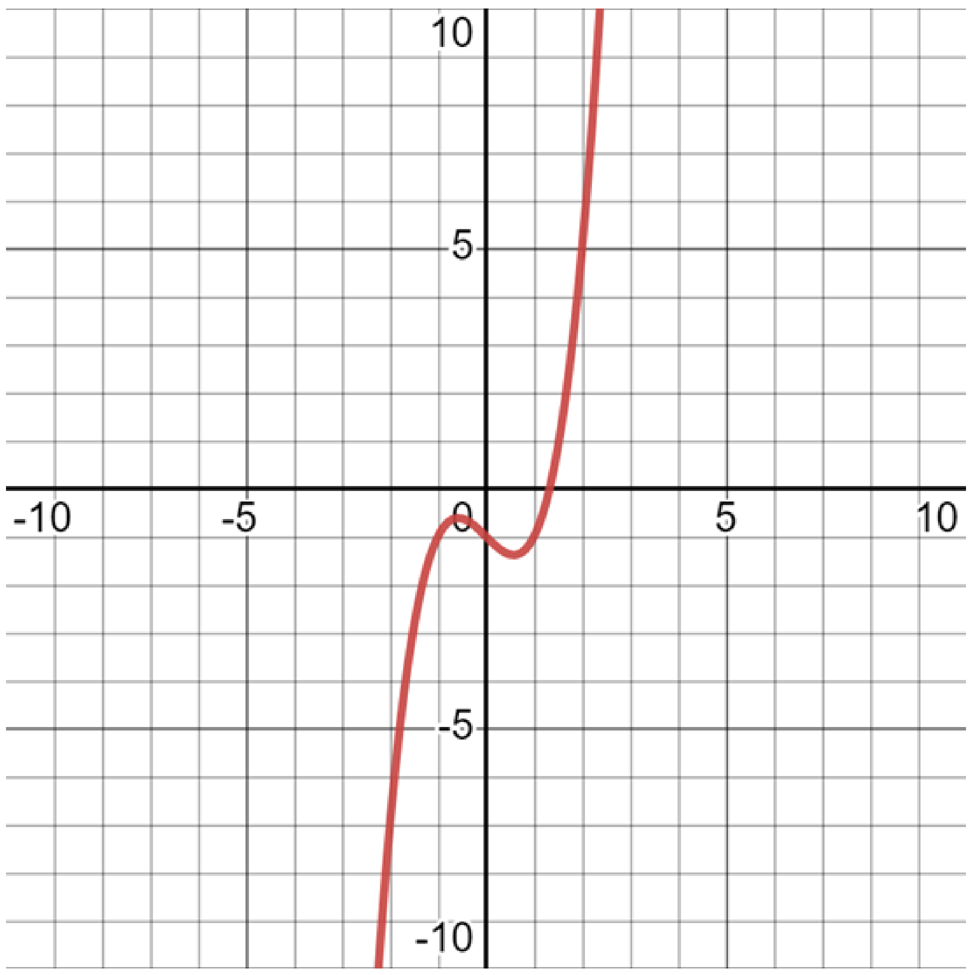
The fixed point iteration method uses the concept of a fixed point in a repeated manner to compute the solution of the given equation. A fixed point is a point in the domain of a function g such that g(x) = x. In the fixed point iteration method, the given function is algebraically converted in the form of g(x) = x.

Algorithm:

Fixed Point Iteration method Steps (Rule)	
Step-1:	First write the equation $x=\phi(x)$
Step-2:	Find points a and b such that $a<b$ and $f(a)\cdot f(b)<0$.
Step-3:	If $f(a)$ is more closer to 0 then $f(b)$ then $x0=a$ else $x0=b$
Step-4:	$x1=\phi(x0)$ $x2=\phi(x1)$ $x3=\phi(x2)$ ... Repeat until $\lvert f(xi)-f(xi-1)\rvert\approx 0$

Find a root of an equation $f(x)=x3-x-1$ using Fixed Point Iteration method

Solution:



Method-1

Let $f(x)=x3-x-1$

$x3-x-1=0$

$\therefore x3=x+1$

$\therefore x=3\sqrt{x+1}$

$\therefore \phi(x)=3\sqrt{x+1}$

Here

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

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

∴ Root lies between 1 and 2

x0=1+22=1.5

x1=$\phi(x0)=\phi(1.5)$=1.35721

x2=$\phi(x1)=\phi(1.35721)$=1.33086

x3=$\phi(x2)=\phi(1.33086)$=1.32588

x4=$\phi(x3)=\phi(1.32588)$=1.32494

x5=$\phi(x4)=\phi(1.32494)$=1.32476

Approximate root of the equation $x3-x-1=0$ using Iteration method is 1.32476

n	*x*0	*x*1=$\phi(x0)$	Update	Difference x1-*x*0
2	1.5	1.35721	$x0=x1$	0.14279
3	1.35721	1.33086	$x0=x1$	0.02635
4	1.33086	1.32588	$x0=x1$	0.00498
5	1.32588	1.32494	$x0=x1$	0.00094
6	1.32494	1.32476	$x0=x1$	0.00018

SYSTEM OF LINEAR EQUATIONS:

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

JACOBI METHOD:

Jacobi method is one the iterative methods for approximating the solution of a system of n linear equations in n variables. The Jacobi iterative method is considered as an iterative algorithm which is used for determining the solutions for the system of linear equations in numerical linear algebra, which is diagonally dominant. In this method, an approximate value is filled in for each diagonal element. Until it converges, the process is iterated. This algorithm was first called the Jacobi transformation process of matrix diagonalization. Jacobi Method is also known as the simultaneous displacement method.

Properties of Jacobian Method

Adding the applications of the Jacobian matrix in different areas, this method holds some important properties. The simplicity of this method is considered in both the aspects of good and bad. This method can be stated as good since it is the first iterative method and easy to understand. However, the method is also considered bad since it is not typically used in practice. Though there are cons, is still a good starting point for those who are willing to learn more useful but more complicated iterative methods.

Problem: Gauss Jacobi 7y+2x=11,3x-y=5

Solution:

7y+2x=11,3x-y=5

Total Equations are 2

$2x+7y=11$

$3x-y=5$

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.
$3x-y=5$

$2x+7y=11$

From the above equations

$x_{k+1}=13\left(5+y_k\right)$

$y_{k+1}=17\left(11-2x_k\right)$

Initial gauss $(x,y)=(0,0$

Solution steps are
1st Approximation

$x_1=13[5+(0)]=13[5]=1.6667$

$y_1=17[11-2(0)]=17[11]=1.5714$

Solution By Gauss Jacobi Method.
$x=1.9998\cong 2$

$y=0.9999\cong 1$

Iterations are tabulated as below

Iteration	x	y
1	1.6667	1.5714
2	2.1905	1.0952
3	2.0317	0.9456
4	1.9819	0.9909
5	1.997	1.0052
6	2.0017	1.0009
7	2.0003	0.9995
8	1.9998	0.9999

GUASS SIEDAL METHOD:

In numerical linear algebra, the Gauss–Seidel method, also known as the Liebmann method or the method of successive displacement, is an iterative method used to solve a system of linear equations. It is named after the German mathematicians Carl Friedrich Gauss and Philipp Ludwig von Seidel, and is similar to the Jacobi method. Though it can be applied to any matrix with non-zero elements on the diagonals, convergence is only guaranteed if the matrix is either strictly diagonally dominant,[1] or symmetric and positive definite. It was only mentioned in a private letter from Gauss to his student Gerling in 1823.[2] A publication was not delivered before 1874 by Seidel.[3]

EXAMPLE:

$$\begin{array}{rcrcrcr} 4x_1 & - & x_2 & - & x_3 & = & 3 \\ -2x_1 & + & 6x_2 & + & x_3 & = & 9 \\ -x_1 & + & x_2 & + & 7x_3 & = & -6 \end{array}.$$

At each step, given the current values $x_1^{(k)}$, $x_2^{(k)}$, $x_3^{(k)}$, we solve for $x_1^{(k+1)}$, $x_2^{(k+1)}$, $x_3^{(k+1)}$ in

$$\begin{array}{rcrcrcr} 4x_1^{(k+1)} & - & x_2^{(k)} & - & x_3^{(k)} & = & 3 \\ -2x_1^{(k+1)} & + & 6x_2^{(k+1)} & + & x_3^{(k)} & = & 9 \\ -x_1^{(k+1)} & + & x_2^{(k+1)} & + & 7x_3^{(k+1)} & = & -6 \end{array}.$$

To compare our results from the two methods, we again choose $\boldsymbol{x}^{(0)} = (0, 0, 0)$. We then find $\boldsymbol{x}^{(1)} = (x_1^{(1)}, x_2^{(1)}, x_3^{(1)})$ by solving

$$\begin{array}{rcrcrcr} 4x_1^{(1)} & - & 0 & - & 0 & = & 3 \\ -2x_1^{(1)} & + & 6x_2^{(1)} & + & 0 & = & 9 \\ -x_1^{(1)} & + & x_2^{(1)} & + & 7x_3^{(1)} & = & -6 \end{array}.$$

Let us be clear about how we solve this system. We first solve for $x_1^{(1)}$ in the first equation and find that

$$x_1^{(1)} = 3/4 = 0.750.$$

We then solve for $x_2^{(1)}$ in the second equation, using the new value of $x_1^{(1)} = 0.750$, and find that

$$x_2^{(1)} = [9 + 2(0.750)] / 6 = 1.750.$$

Finally, we solve for $x_3^{(1)}$ in the third equation, using the new values of $x_1^{(1)} = 0.750$ and $x_2^{(1)} = 1.750$, and find that

$$x_3^{(1)} = [-6 + 0.750 - 1.750] / 7 = -1.000.$$

The result of this first iteration of the Gauss-Seidel Method is

$$\boldsymbol{x}^{(1)} = (x_1^{(1)}, x_2^{(1)}, x_3^{(1)}) = (0.750, 1.750, -1.000).$$

We iterate this process to generate a sequence of increasingly better approximations $\boldsymbol{x}^{(0)}$, $\boldsymbol{x}^{(1)}$, $\boldsymbol{x}^{(2)}$, … and find results similar to those that we found for Example 1.

k	$\boldsymbol{x}^{(k)}$			$\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-1)}$			$\boldsymbol{e}^{(k)} = \boldsymbol{x} - \boldsymbol{x}^{(k)}$			$\|\|\boldsymbol{e}^{(k)}\|\|$
0	0.000	0.000	0.000	–	–	–	0.000	0.000	-1.000	2.449
1	0.750	1.750	-1.000	0.000	0.000	-1.000	0.250	0.250	0.000	0.354
2	0.938	1.979	-1.006	0.188	0.229	-0.006	0.063	0.021	0.006	0.066
3	0.993	1.999	-1.001	0.056	0.020	0.005	0.007	0.001	0.001	0.007
4	0.999	2.000	-1.000	0.006	0.001	0.001	0.001	0.000	0.000	0.001
5	1.000	2.000	-1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

SOR:

In numerical linear algebra, the method of successive over-relaxation (SOR) is a variant of the Gauss–Seidel method for solving a linear system of equations, resulting in faster convergence. A similar method can be used for any slowly converging iterative process.

EXAMPLE:

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

Solution:
We know that, for symmetric positive definite matrix the SOR method converges for values of the relaxation parameter w from the interval $0<w<$
The iterations of the SOR method
1. Total Equations are 3

$10x+2y-z=7$

$x+8y+3z=-4$

$-2x-y+10z=9$

2. From the above equations, First write down the equations for Gauss Seidel method

$x_{k+1}=110\left(7-2y_k+z_k\right)$

$y_{k+1}=18\left(-4-x_{k+1}-3z_k\right)$

$z_{k+1}=110\left(9+2x_{k+1}+y_{k+1}\right)$

3. 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 $(1-w)$

$x_{k+1}=(1-w)\cdot x_k+w\cdot 110\left(7-2y_k+z_k\right)$

$y_{k+1}=(1-w)\cdot y_k+w\cdot 18\left(-4-x_{k+1}-3z_k\right)$

$z_{k+1}=(1-w)\cdot z_k+w\cdot 110\left(9+2x_{k+1}+y_{k+1}\right)$

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

Solution steps are
1st Approximation

$x_1=(1-1.25)\cdot 0+1.25\cdot 110[7-2(0)+(0)]=(-0.25)\cdot 0+1.25\cdot 110[7]=0+0.875=0.875$

$y_1=(1-1.25)\cdot 0+1.25\cdot 18[-4-(0.875)-3(0)]=(-0.25)\cdot 0+1.25\cdot 18[-4.875]=0\pm 0.7617=-0.7617$

$z_1=(1-1.25)\cdot 0+1.25\cdot 110[9+2(0.875)+(-0.7617)]=(-0.25)\cdot 0+1.25\cdot 110[9.9883]=0+1.2485=1.2485$

iterations are tabulated as below

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:

It is an estimation if values with in two known values in a sequence if value.

NEWTON FARWARD DIFFERENCE:

Newton's forward interpolation is a polynomial interpolation that is based on Newton's forward operator's initial value and degrees. The amount of data points is one less than the degree of polynomial fitted. This form of interpolation is only utilised when the data points are evenly spaced.

Formula

Newton's Forward Difference formula
$p=x\text{-}x0h$
$y(x)=y0+p\Delta y0+p(p\text{-}1)2!\cdot\Delta 2y0+p(p\text{-}1)(p\text{-}2)3!\cdot\Delta 3y0+p(p\text{-}1)(p\text{-}2)(p\text{-}3)4!\cdot\Delta 4y0+...$

Find Solution of an equation x^3-x+1 using Newton's Forward Difference formula
x1 = 2 and x2 = 4
x = 2.25
Step value (h) = 0.5
Finding f(2)

Solution:
Equation is $f(x)=x3\text{-}x+1$.

The value of table for x and y

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

Newton's forward difference interpolation method to find solution

Newton's forward difference table is

x	**y**	**Δy**	**$\Delta 2y$**	**$\Delta 3y$**	**$\Delta 4y$**
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=2.25

$h=x1-x0=2.5-2=0.5$

$p=x-x0h=2.25-20.5=0.5$

Newton's forward difference interpolation formula is
$y(x)=y0+p\Delta y0+p(p-1)2!\cdot\Delta 2y0+p(p-1)(p-2)3!\cdot\Delta 3y0+p(p-1)(p-2)(p-3)4!\cdot\Delta 4y0$

$y(2.25)=7+0.5\times 7.125+0.5(0.5-1)2\times 3.75+0.5(0.5-1)(0.5-2)6\times 0.75+0.5(0.5-1)(0.5-2)(0.5-3)24\times 0$

$y(2.25)=7+3.5625-0.4688+0.0469+0$

$y(2.25)=10.1406$

NEWTON BACKWARD DIFFERENCE:

In order to reduce the number of numerical computations required to compute a large number of interpolated values using the existing interpolation formula.

Formula

Newton's Backward Difference formula
$p=x-xnh$ $y(x)=yn+p\nabla yn+p(p+1)2!\cdot\nabla 2yn+p(p+1)(p+2)3!\cdot\nabla 3yn+p(p+1)(p+2)(p+3)4!\cdot\nabla 4yn+\dots$

2. Find Solution using Newton's Backward Difference formula

x	f(x)
0	1
1	0
2	1
3	10

x = 4

Solution:
The value of table for x and y

x	0	1	2	3
y	1	0	1	10

Newton's backward difference interpolation method to find solution

Newton's backward difference table is

x	y	∇y	$\nabla 2y$	$\nabla 3y$
0	1			
		-1		
1	0		2	
		1		**6**
2	1		**8**	
		9		
3	**10**			

The value of x at you want to find the $f(x)$:x=4
$h=x1-x0=1-0=1$

p=$x-xnh$=4-31=1

Newton's backward difference interpolation formula is
y(x)=$yn+p\nabla yn+p(p+1)2!\cdot\nabla 2yn+p(p+1)(p+2)3!\cdot\nabla 3yn$

y(4)=10+1×9+1(1+1)2×8+1(1+1)(1+2)6×6

y(4)=10+9+8+6

y(4)=33

NEWTONS DIVIDED DIFFERENCE INTERPOLATION METHOD:

(x - xk-1) f [x0, x1, . . ., xk]. 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. using Newton's divided difference formula.

Formula

Newton's Divided Difference Interpolation formula
$y(x)=y0+(x-x0)f[x0,x1]+(x-x0)(x-x1)f[x0,x1,x2]+...$

Examples
1. Find Solution using Newton's Divided Difference Interpolation formula

x	f(x)
300	2.4771
304	2.4829
305	2.4843
307	2.4871

x = 301

Solution:
The value of table for x and y

x	300	304	305	307
y	2.4771	2.4829	2.4843	2.4871

Numerical divided differences method to find solution

Newton's divided difference table is

x	**y**	**1*st* order**	**2*nd* order**
300	2.4771		
		0.00145	
304	2.4829		0
		0.0014	
305	2.4843		0
		0.0014	
307	2.4871		

The value of x at you want to find the $f(x):x=301$

Newton's divided difference interpolation formula is

$f(x)=y0+(x-x0)f[x0,x1]+(x-x0)(x-x1)f[x0,x1,x2]$

$y(301)=2.4771+(301-300)\times0.00145+(301-300)(301-304)\times0$

$y(301)=2.4771+(1)\times0.00145+(1)(-3)\times0$

$y(301)=2.4771+0.00145+0$

y(301)=2.47858

Spline interpolation:

Spline interpolation similar to the Polynomial interpolation x' uses low-degree polynomials in each of the intervals and chooses the polynomial pieces such that they fit smoothly together. The resulting function is called a spline.

Spline interpolation divided into 3 types

- **Linear spline**
- **Quardatic spline**
- **Cubic spline**

LINEAR SPLINE:

The linear spline represents a set of line segments between the two adjacent data points (Vk,Ik) and (Vk+1,Ik+1). The equations for each line segment can be immediately found in a simple form: Ik(V) = Ik + (Ik+1 - Ik) (V - Vk) / (Vk+1 - Vk), where V = [Vk,Vk+1] and k = 0,1,...,(n-1).

For a set of data points usually termed **knots**: $(x_i,\ y_i,\ i = 1, 2, \cdots, n)$, the linear spline is

$$s_i(x) = a_i x + b_i,\ \ for\, x \in \lfloor x_i, x_{i+1} \rfloor, \quad i = 1, 2, \cdots, n-1$$

The C^0 conditions, $s_i(x_i) = y_i$ and $s_i(x_{i+1}) = y_{i+1}$ yield $2(n-2)$ equations at interior points. These along with one condition at each end point give a total of $2n-2$ equations to match the $2n-2$ unknowns: $a_i, b_i, i = 1, 2, \cdots, n-1$. By applying these, we get

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

which results in straight lines joining neighboring points.

Clearly, $s_i(x)$ is the Lagrange interpolation formula for the data set consisting of the following two points: (x_i, y_i) and (x_{i+1}, y_{i+1}). Hence it is the solution approximation for linear finite elements in one dimension.

Find the linear splines for the following data set

i	1	2	3	4	5
x	0	5	7	8	10
y	0	2	-1	-2	20

Linear spline fit

$$s_1(x) = 0\frac{x-5}{0-5} + 2\frac{x-0}{5-0} = 2.5x, \quad x \in [0, 5]$$

$$s_2(x) = 2\frac{x-7}{5-7} - 1\frac{x-5}{7-5} = -1.5x + 9.5, \quad x \in [5, 7]$$

$$s_3(x) = -1\frac{x-8}{7-8} - 2\frac{x-7}{8-7} = -x + 6, \quad x \in [7, 8]$$

$$s_4(x) = -2\frac{x-10}{8-10} + 20\frac{x-8}{10-8} = 11x - 90, \quad x \in [8, 10]$$

The "*linear_spline.m*" function is built according to the following algorithm:

(1) Input x where the interpolation value y is needed;
Check to see if x is out of range; if yes, abort the program; if no, proceed;
(2) Find the data interval in which x resides;
(3) Calculate y according to Equation

CUBIC SPLINE:

Cubic spline interpolation is a way of finding a curve that connects data points with a degree of three or less. Splines are polynomial that are smooth and continuous across a given plot and also continuous first and second derivatives where they join.

Formula

Cubic spline formula is $f(x)=(xi-x)36hMi-1+(x-xi-1)36hMi+(xi-x)h(yi-1-h26Mi-1)+(x-xi-1)h(yi-h26Mi)\rightarrow(1)$
$Mi-1+4Mi+Mi+1=6h2(yi-1-2yi+yi+1)\rightarrow(2)$

Calculate Cubic Splines

X	1	2	3	4
Y	1	5	11	8

y(1.5), y'(2)

Solution:

x	1	2	3	4
y	1	5	11	8

Cubic spline formula is
$f(x)=(xi-x)36hMi-1+(x-xi-1)36hMi+(xi-x)h(yi-1-h26Mi-1)+(x-xi-1)h(yi-h26Mi)\rightarrow(1)$

We have, $Mi-1+4Mi+Mi+1=6h2(yi-1-2yi+yi+1)\rightarrow(2)$

Here $h=1,n=3$

$M0=0,M3=0$

Substitute $i=1$ in equation (2)

$M0+4M1+M2=6h2(y0-2y1+y2)$

$\Rightarrow 0+4M1+M2=61\cdot(1-2\cdot5+11)$

$\Rightarrow 4M1+M2=12$

Substitute $i=2$ in equation (2)
$M1+4M2+M3=6h2(y1-2y2+y3)$

$\Rightarrow M1+4M2+0=61\cdot(5-2\cdot11+8)$

$\Rightarrow M1+4M2=-54$

Solving these 2 equations using elimination method
Substitute $i=1$ in equation (1), we get cubic spline in $1st$ interval $[x0,x1]=[1,2]$

$f1(x)=(x1-x)36hM0+(x-x0)36hM1+(x1-x)h(y0-h26M0)+(x-x0)h(y1-h26M1)$

$f1(x)=(2-x)36\cdot0+(x-1)36\cdot6.8+(2-x)1(1-16\cdot0)+(x-1)1(5-16\cdot6.8)$

$f1(x)=1.1333x3-3.4x2+6.2667x-3$, for $1\leq x\leq 2$

Substitute $i=2$ in equation (1), we get cubic spline in $2nd$ interval $[x1,x2]=[2,3]$

$f2(x)=(x2-x)36hM1+(x-x1)36hM2+(x2-x)h(y1-h26M1)+(x-x1)h(y2-h26M2)$

$f2(x)=(3-x)36\cdot6.8+(x-2)36\cdot-15.2+(3-x)1(5-16\cdot6.8)+(x-2)1(11-16\cdot-15.2)$

$f2(x)=-3.6667x3+25.4x2-51.3333x+35.4$, for $2\leq x\leq 3$

Substitute $i=3$ in equation (1), we get cubic spline in $3rd$ interval $[x2,x3]=[3,4]$

$f3(x)=(x3-x)36hM2+(x-x2)36hM3+(x3-x)h(y2-h26M2)+(x-x2)h(y3-h26M3)$

$f3(x)=(4-x)36\cdot-15.2+(x-3)36\cdot0+(4-x)1(11-16\cdot-15.2)+(x-3)1(8-16\cdot0)$

$f3(x)=2.5333x3-30.4x2+116.0667x-132$, for $3\leq x\leq 4$

For $y(1.5)$, $1.5\in[1,2]$, so substitute $x=1.5$ in $f1(x)$, we get

$f1(1.5)=2.575$

For $y'(2)$, $2\in[1,2]$, so find $f'1(x)$

$f'1(x)=3.4x2-6.8x+6.2667$

Now substitute $x=2$ in $f'1(x)$, we get

$f'1(2)=6.2667$

QUADRATIC SPLINE:

A Quadratic Spline is the creation of a set of polynomial functions that are quadratic, or, easier to understand, follow the format $f(x)=ax^2+bx+c$, where a, b and c are the values obtained while doing the Splines to create the desired functions.

A quadratic spline has a quadratic function for each data interval

$$s_i(x) = a_i x^2 + b_i x + c_i, \quad for\ x \in [x_i, x_{i+1}], \quad i = 1, 2, \cdots, n-1$$

which is constrained to satisfy the C^0 and C^1 conditions.

For the C^0 condition, we get

$$s_i(x_i) = y_i, \quad s_i(x_{i+1}) = y_{i+1}, \quad i = 1,2,\ldots n-1$$

For the C^1 condition, we get

$$s_i'(x_{i+1}) = s_{i+1}'(x_{i+1}), \quad i = 1, 2, \cdots, n-2$$

From the above three requirements, there are *3n-4* constraint conditions. But the splines require a total of *3n-3* conditions, so we are one condition short and the problem is not completely defined. Usually, $s'_1(x_1) = 0$ is used for the additional condition. This results in the so-called **natural** quadratic spline. Certainly other conditions can be used, such as $s'_1(x_1) = s'_{n-1}(x_n)$.

To find the formula of the splines, first denote $d_i = s'_i(x_i)$, then since $s'(x)$ is a linear spline for the data set (x_i, d_i, i = 1,2, …, n), we have

$$s_i'(x) = d_{i+1}\frac{x - x_i}{x_{i+1} - x_i} + d_i\frac{x_{i+1} - x}{x_{i+1} - x_i} = \frac{(d_{i+1} - d_i)x - d_{i+1}x_i + d_i x_{i+1}}{x_{i+1} - x_i}$$
$$= \frac{d_{i+1} - d_i}{x_{i+1} - x_i}(x - x_i) + d_i$$

which is of course is the linear equation for the slope.

Integrating over *x*, we get

$$s_i(x) = \frac{d_{i+1} - d_i}{2(x_{i+1} - x_i)}(x - x_i)^2 + d_i(x - x_i) + y_i$$

after solving for the constant of integration using the C^0 boundary condition: $s_i(x_i) = y_i$. So the quadratic spline is defined once we have the d_i's.

Letting $x = x_{i+1}$, we get

$$y_{i+1} = \frac{d_{i+1} - d_i}{2(x_{i+1} - x_i)}(x_{i+1} - x_i)^2 + d_i(x_{i+1} - x_i) + y_i$$

which leads to

$$d_{i+1} = 2\frac{y_{i+1} - y_i}{x_{i+1} - x_i} - d_i, \quad i = 1, 2, \cdots, n-1$$

which is a recursive scheme. If d_1 is known, then d_i can be derived from the above equation.

Various conditions can be imposed to obtain d_1:

- For the natural spline, $d_1 = 0$.

- If $d_1 = d_2$, then d_1 can be calculated by

$$d_1 = d_2 = 2\frac{y_2 - y_1}{x_2 - x_1} - d_1 \Rightarrow d_1 = \frac{y_2 - y_1}{x_2 - x_1}$$

- If $d_1 = d_n$, then d_1 can be calculated by

$$d_1 = d_n = 2\sum_{i=2}^{n-1}\left((-1)^{n-i+1}\frac{y_{i+1} - y_i}{x_{i+1} - x_i}\right) - (-1)^n d_1 \Rightarrow \begin{cases} d_1 = \sum_{i=2}^{n-1}\left((-1)^{n-i+1}\frac{y_{i+1} - y_i}{x_{i+1} - x_i}\right), & n \text{ is even} \\ \text{no solution}, & n \text{ is odd} \end{cases}$$

This condition can not be applied when n is an odd number.

The "*quadratic_spline.m*" function is built according to the following algorithm

(1) Input *x* where the interpolation value *y* is needed;
Check to see if *x* is out of range; if yes, abort the program; if no, proceed;
(2) Find the data interval where *x* belongs;
(3) Calculate d_1; Calculate d_i (Equation 4.2.5);
(4) Calculate *y* according to Equation .

Example 4.2.1. Given the same data points in Example 4.1.1, derive the quadratic splines.

Using the natural quadratic splines, where $d_1 = 0$, then we have

$$d_2 = 2\frac{2-0}{5-0} - 0 = 0.8, \quad d_3 = 2\frac{-1-2}{7-5} - 0.8 = -3.8$$

$$d_4 = 2\frac{-2+1}{8-7} + 3.8 = 1.8, \quad d_5 = 2\frac{20+2}{10-8} - 1.8 = 20.2$$

$$s_1(x) = \frac{0.8-0}{2(5-0)}(x-0)^2 + 0(x-0) + 0 = 0.08x^2, \quad x \in [0, 5]$$

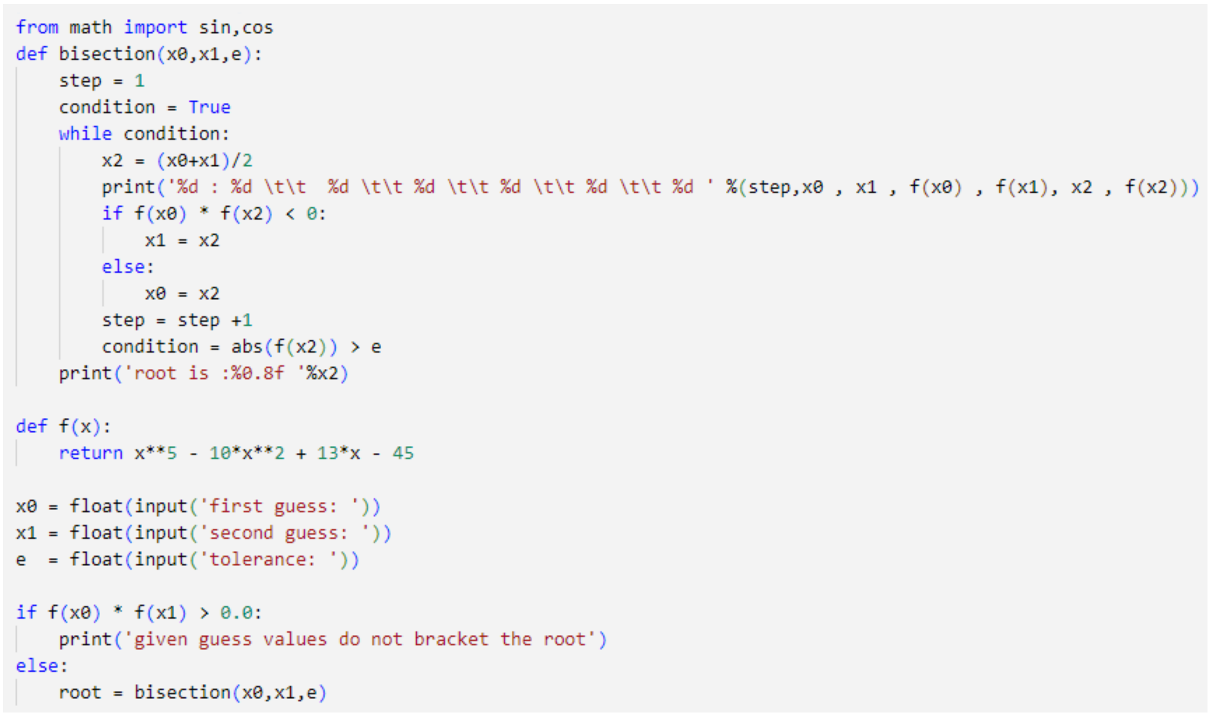
$$s_2(x) = \frac{-3.8-0.8}{2(7-5)}(x-5)^2 + 0.8(x-5) + 2 = -1.15x^2 + 12.3x - 289.5, \quad x \in [5, 7]$$

$$s_3(x) = \frac{1.8+3.8}{2(8-7)}(x-7)^2 - 3.8(x-7) - 1 = 2.8x^2 - 9.4x + 162.8, \quad x \in [7, 8]$$

$$s_4(x) = \frac{20.2-1.8}{2(10-8)}(x-8)^2 + 1.8(x-8) - 2 = 4.6x^2 - 71.8x + 273.6, \quad x \in [8, 10]$$

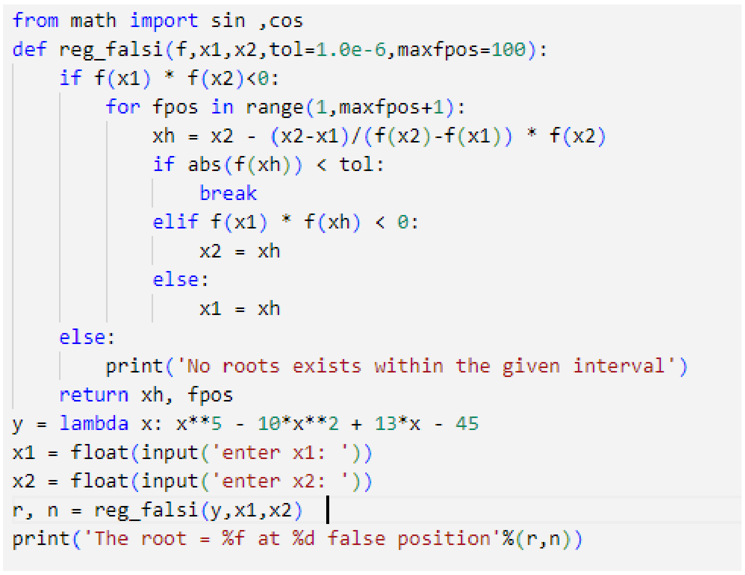
The results and the splines due to $d_1 = d_2$, are plotted in Figure 4.2.1. Near $x = 0$, the natural spline has a flatter curve due to zero slope at x = 0. The other spline has a linear slope in the first interval.

BISECTION METHOD



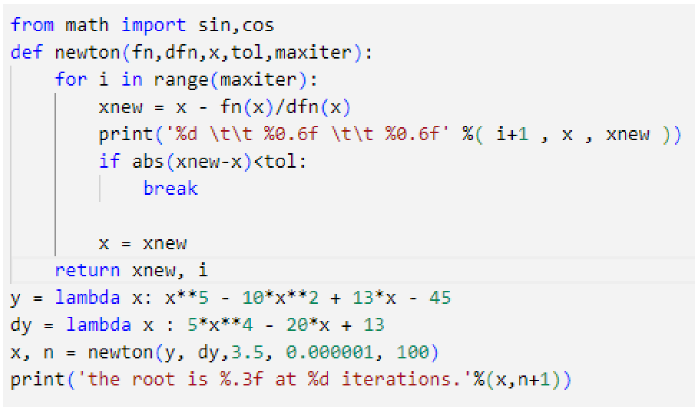
```
from math import sin,cos
def bisection(x0,x1,e):
    step = 1
    condition = True
    while condition:
        x2 = (x0+x1)/2
        print('%d : %d \t\t  %d \t\t %d \t\t %d \t\t %d \t\t %d ' %(step,x0 , x1 , f(x0) , f(x1), x2 , f(x2)))
        if f(x0) * f(x2) < 0:
            x1 = x2
        else: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

REGULA FALSE METHOD



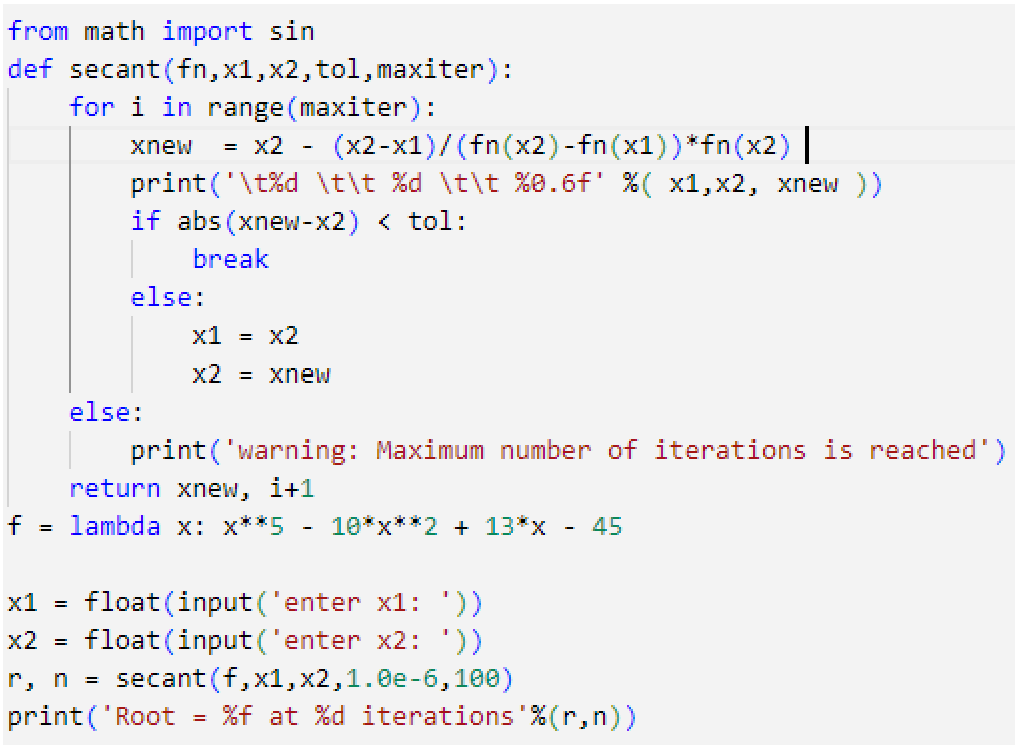
```python
from math import sin ,cos
def reg_falsi(f,x1,x2,tol=1.0e-6,maxfpos=100):
    if f(x1) * f(x2)<0:
        for fpos in range(1,maxfpos+1):
            xh = x2 - (x2-x1)/(f(x2)-f(x1)) * f(x2)
            if abs(f(xh)) < tol:
                break
            elif f(x1) * f(xh) < 0:
                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))
```

Newton Raphson Method



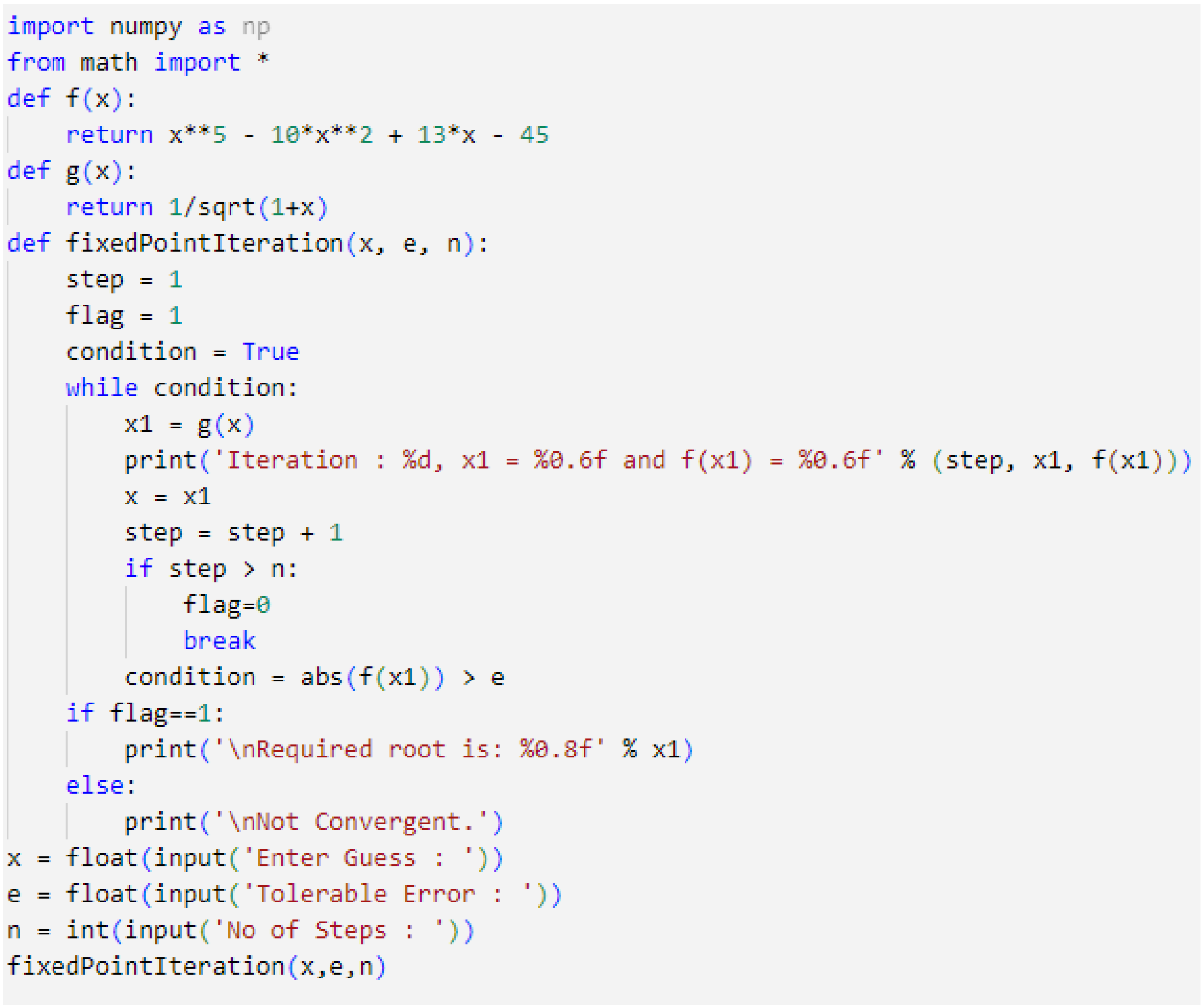
```python
from math import sin,cos
def newton(fn,dfn,x,tol,maxiter):
    for i in range(maxiter):
        xnew = x - fn(x)/dfn(x)
        print('%d \t\t %0.6f \t\t %0.6f' %( i+1 , x , xnew ))
        if abs(xnew-x)<tol:
            break

        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))
```

Secant Method



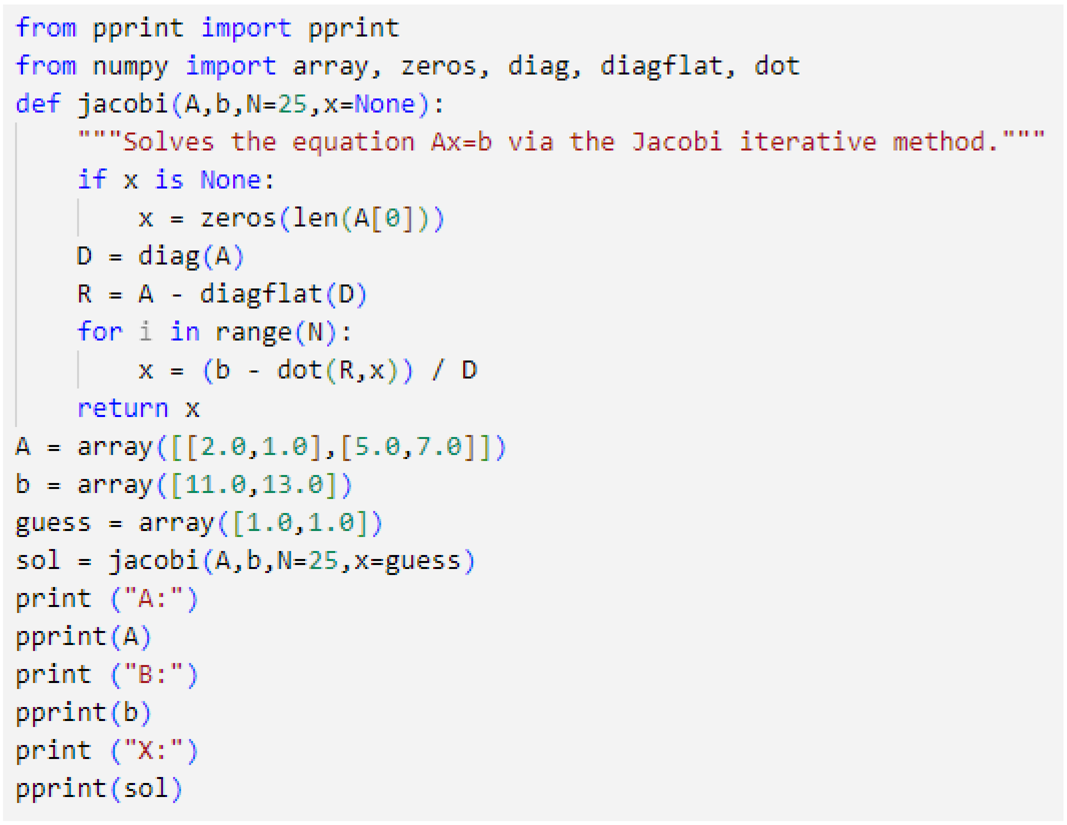
```python
from math import sin
def secant(fn,x1,x2,tol,maxiter):
    for i in range(maxiter):
        xnew  = x2 - (x2-x1)/(fn(x2)-fn(x1))*fn(x2)
        print('\t%d \t\t %d \t\t %0.6f' %( x1,x2, xnew ))
        if abs(xnew-x2) < tol:
            break
        else:
            x1 = x2
            x2 = 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))
```

Fixed Point Method



```python
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)
```

Jacobi Method



```python
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)
```

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