***BISECTION METHOD:***

**Bisection Method background:**

The bisection method is one of the simplest and most reliable of iterative methods for the solution of nonlinear equations. This method, also known as binary chopping or half-interval method, relies on the fact that if f(x) is real and continuous in the interval a < x < b, and f(a) and f(b) are of opposite signs, that is,

F(a) f(b) < 0

Then there is at least one real root in the interval between a and b. (There may be more than one root in the interval).

Let x = a and x2 = b. Let us also define another point x0 r, to be the midpoint between a and b. That is,

x0 =

Now, there exists the following three conditions:

1. f (x0) = 0, we have a root at x0
2. 2. If f(x0) f(x1)<0, there is a root between x0 and x1 .
3. 3. If f(X0) f(x2 )< 0, there is a root between x0, and x2

It follows that by testing the sign of the function at midpoint, we can deduce which part of the interval contains the root. This is illustrated in shows that, since f(x0) and f(x2) are of opposite sign, a root between x0, and x2 .We can further divide this subinterval into two ha to locate a new subinterval containing the root. This process can be repeated until the interval containing the root is as small as we desire.

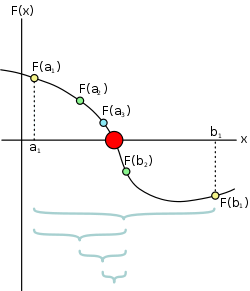


Fig : illustration of bisection method

**Bisection Method Algorithm:**

1. Decide initial values for x1, and x2, and stopping criterion, E.
2. Compute f1 = f(x1) and f2= f(x2).
3. If f1 \* f2 >0, x1  and x2  do not bracket any root and go to step 7;

Otherwise continue.

4. Compute x0 = (x1 + X2)/2 and compute f0= f(x0)

5. If f1 x f2<0 then

set x2 = x0

else

set x1 = x0

set f1 = f0

6. If absolute value of (x2 – x1)/x2 is less than error E, then

root = (x1 + x2)/2

write the value of root

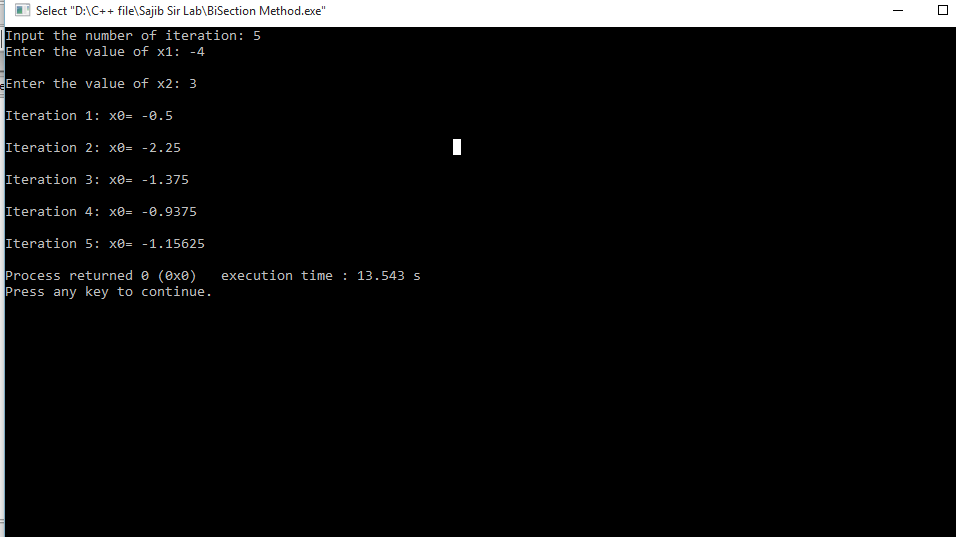
go to step 7

else

go to step 4

7. Stop

**Bisection Method Output:**

****

***FALSE POSITION METHOD:***

**False Position Method Background:**

In bisection method, the interval between x1, and x2, is divided into two equal halves, irrespective of location of the root. It may be possible that the root is closer to one end. Note that the root is closer to x1. Let us join the points x1, and x2, by a straight line. The point of intersection of this line with the x0 axis (x0) gives an improved estimate of the root and is called the false position of the root. This point then replaces one of the initial guesses that has a function value of the same sign as f(x0). The process is repeated with the new values of x1, and x2. Since this method uses the false position of the root repeatedly, it is called the false position method. It is also called the linear interpolation method (because approximate root is determined by linear interpolation).

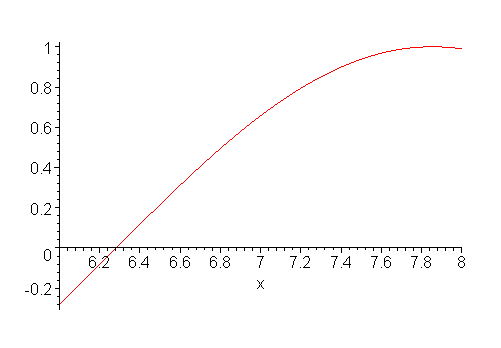


Fig: illustration of false position method

A graphical depiction of the false position method is show the picture .we know that equation of the line joining the points ( x1, f(x1) )and ( x2, f(x2) is given by

Since the line intersects the x-axis at x0 when x=x0 ,y=0,we have

Or,

Then we have,

This equation is known as the false position formula. Note that is obtained by applying a correction to .

**False Position Method Algorithm:**

Let \*

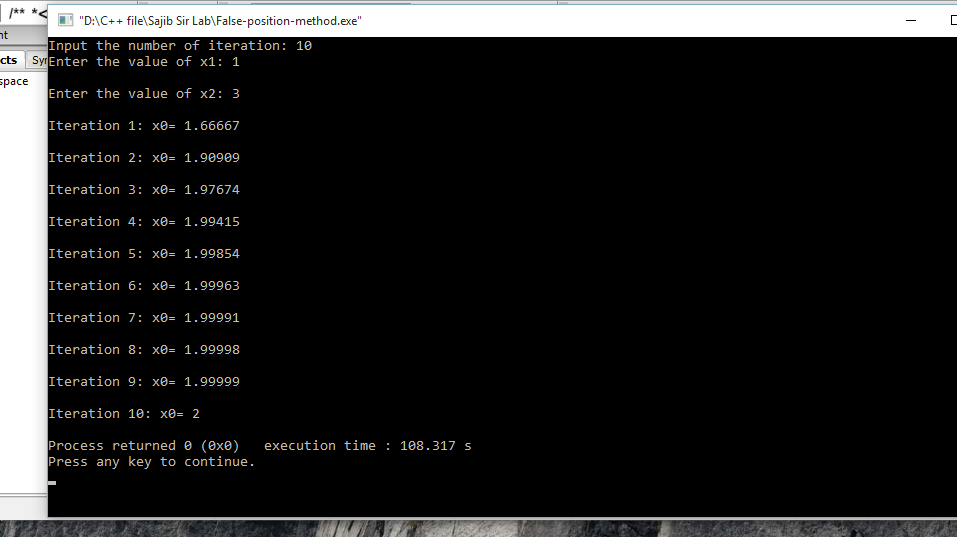
If <0

Set x2 =x0

Otherwise

Set x1=x0

**False position Method Output:**



***SECANT METHOD:***

**Secant Method Background:**

Secant method, like the false position and bisection methods, uses two initial estimates but does not require that they must bracket the root. For example, the secant method can use the points x1 and x2 as starting values, although they do not bracket the root. Slope of the secant line passing through x1, and x2 is given by The secant method does not require that the root remain bracketed, like the [bisection method](https://en.wikipedia.org/wiki/Bisection_method) does, and hence it does not always converge. The [false position method](https://en.wikipedia.org/wiki/False_position_method) (or *regula falsi*) uses the same formula as the secant method. However, it does not apply the formula on {\displaystyle x\_{n-1}}and {\displaystyle x\_{n-2}} like the secant method, but on {\displaystyle x\_{n-1}} and on the last iterate {\displaystyle x\_{k}}such that {\displaystyle f(x\_{k})}and {\displaystyle f(x\_{n-1})} have a different sign. This means that the [false position method](https://en.wikipedia.org/wiki/False_position_method) always converges.

The recurrence formula of the secant method can be derived from the formula for [Newton's method](https://en.wikipedia.org/wiki/Newton%27s_method)

{\displaystyle x\_{n}=x\_{n-1}-{\frac {f(x\_{n-1})}{f'(x\_{n-1})}}}by using the [finite-difference](https://en.wikipedia.org/wiki/Finite-difference) approximation

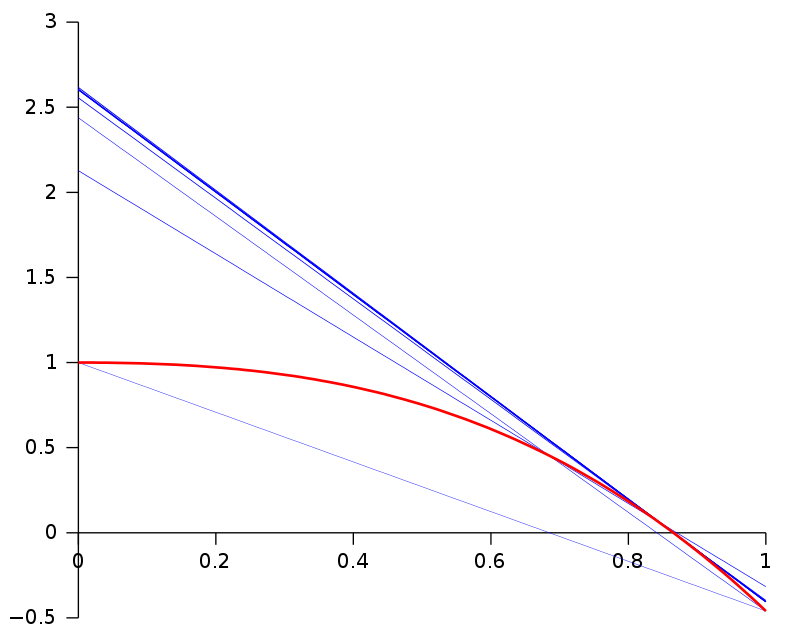
{\displaystyle f'(x\_{n-1})\approx {\frac {f(x\_{n-1})-f(x\_{n-2})}{x\_{n-1}-x\_{n-2}}}.}

The secant method can be interpreted as a method in which the derivative is replaced by an approximation and is thus a [quasi-Newton method](https://en.wikipedia.org/wiki/Quasi-Newton_method).

If we compare Newton's method with the secant method, we see that Newton's method converges faster (order 2 against *φ* ≈ 1.6). However, Newton's method requires the evaluation of both {\displaystyle f} and its derivative {\displaystyle f'} at every step, while the secant method only requires the evaluation of {\displaystyle f}. Therefore, the secant method may occasionally be faster in practice. For instance, if we assume that evaluating {\displaystyle f} takes as much time as evaluating its derivative and we neglect all other costs, we can do two steps of the secant method (decreasing the logarithm of the error by a factor *φ*2 ≈ 2.6) for the same cost as one step of Newton's method (decreasing the logarithm of the error by a factor 2), so the secant method is faster. If, however, we consider parallel processing for the evaluation of the derivative, Newton's method proves its worth, being faster in time, though still spending more steps.

In numerical analysis, the **secant method** is a root-

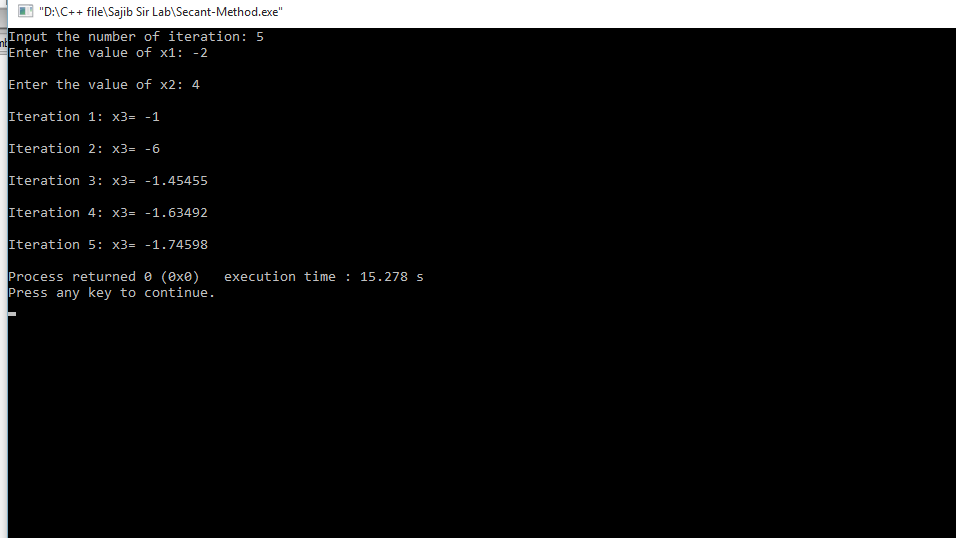
finding **algorithm** that uses a succession of roots of **secant** line to better approximate a root of a function f. The **secant method** can be thought of as a finite-difference approximation of Newton's **method**.



**Secant Method Algorithm:**

1. Start
2. Get values of x0, x1 and e  
   \*Here x0 and x1 are the two initial guesses  
   e is the stopping criteria, absolute error or the desired degree of accuracy\*
3. Compute f(x0) and f(x1)
4. Compute x2 = [x0\*f(x1) – x1\*f(x0)] / [f(x1) – f(x0)]
5. Test for accuracy of x2  
   If [ (x2 – x1)/x2 ] > e, \*Here [ ] is used as modulus sign\*  
   then assign x0 = x1 and x1 = x2  
   goto step 4  
   Else,  
   goto step 6
6. Display the required root as x2.
7. Stop

**Secant Method Output:**



***JACOBI ITERATION METHOD:***

**Jacobi Iteration Background:**

The Gauss–Seidel method is an [iterative technique](https://en.wikipedia.org/wiki/Iterative_method) for solving a square system of *n* linear equations with unknown **x**:

{\displaystyle A\mathbf {x} =\mathbf {b} }.

It is defined by the iteration {\displaystyle L\_{\*}\mathbf {x} ^{(k+1)}=\mathbf {b} -U\mathbf {x} ^{(k)},} where {\displaystyle \mathbf {x} ^{(k)}} is the *k*th approximation or iteration of {\displaystyle \mathbf {x} ,\,\mathbf {x} ^{(k+1)}}is the next or *k* + 1 iteration of {\displaystyle \mathbf {x} }and the matrix *A* is decomposed into a [lower triangular](https://en.wikipedia.org/wiki/Triangular_matrix) component {\displaystyle L\_{\*}}, and a [strictly upper triangular](https://en.wikipedia.org/wiki/Triangular_matrix#Strictly_triangular_matrix) component *U*: {\displaystyle A=L\_{\*}+U}

In more detail, write out *A*, **x** and **b** in their components:

{\displaystyle A={\begin{bmatrix}a\_{11}&a\_{12}&\cdots &a\_{1n}\\a\_{21}&a\_{22}&\cdots &a\_{2n}\\\vdots &\vdots &\ddots &\vdots \\a\_{n1}&a\_{n2}&\cdots &a\_{nn}\end{bmatrix}},\qquad \mathbf {x} ={\begin{bmatrix}x\_{1}\\x\_{2}\\\vdots \\x\_{n}\end{bmatrix}},\qquad \mathbf {b} ={\begin{bmatrix}b\_{1}\\b\_{2}\\\vdots \\b\_{n}\end{bmatrix}}.}

Then the decomposition of *A* into its lower triangular component and its strictly upper triangular component is given by:

{\displaystyle A=L\_{\*}+U\qquad {\text{where}}\qquad L\_{\*}={\begin{bmatrix}a\_{11}&0&\cdots &0\\a\_{21}&a\_{22}&\cdots &0\\\vdots &\vdots &\ddots &\vdots \\a\_{n1}&a\_{n2}&\cdots &a\_{nn}\end{bmatrix}},\quad U={\begin{bmatrix}0&a\_{12}&\cdots &a\_{1n}\\0&0&\cdots &a\_{2n}\\\vdots &\vdots &\ddots &\vdots \\0&0&\cdots &0\end{bmatrix}}.}

The system of linear equations may be rewritten as:

{\displaystyle L\_{\*}\mathbf {x} =\mathbf {b} -U\mathbf {x} }

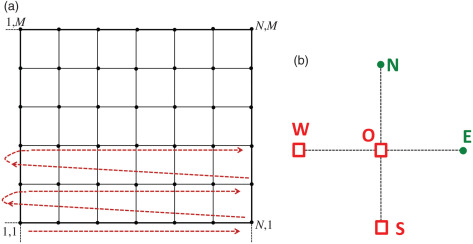
The Gauss–Seidel method now solves the left hand side of this expression for **x**, using previous value for **x** on the right hand side. Analytically, this may be written as:

{\displaystyle \mathbf {x} ^{(k+1)}=L\_{\*}^{-1}(\mathbf {b} -U\mathbf {x} ^{(k)}).}

However, by taking advantage of the triangular form of {\displaystyle L\_{\*}}, the elements of **x**(*k*+1) can be computed sequentially using [forward substitution](https://en.wikipedia.org/wiki/Forward_substitution):

{\displaystyle x\_{i}^{(k+1)}={\frac {1}{a\_{ii}}}\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 i=1,2,\dots ,n.}

The procedure is generally continued until the changes made by an iteration are below some tolerance, such as a sufficiently small [residual](https://en.wikipedia.org/wiki/Residual_(numerical_analysis)).



**Jacobi Iteration Algorithm:**

**1.Obtain n,1i and bi values.**

**2.2.Set x0i**

**3.Set key=0**

**4.For i=1,2,…n**

**(1)Set sum =bi**

**(2)For j=1,2….n**

**Set sum =sum-ai xoi**

**Repeat j**

**(3)Set xi=sum/ai**

**(4)if key=0 then**

**Set key=1**

**Repeat i**

**5.If key =1 then**

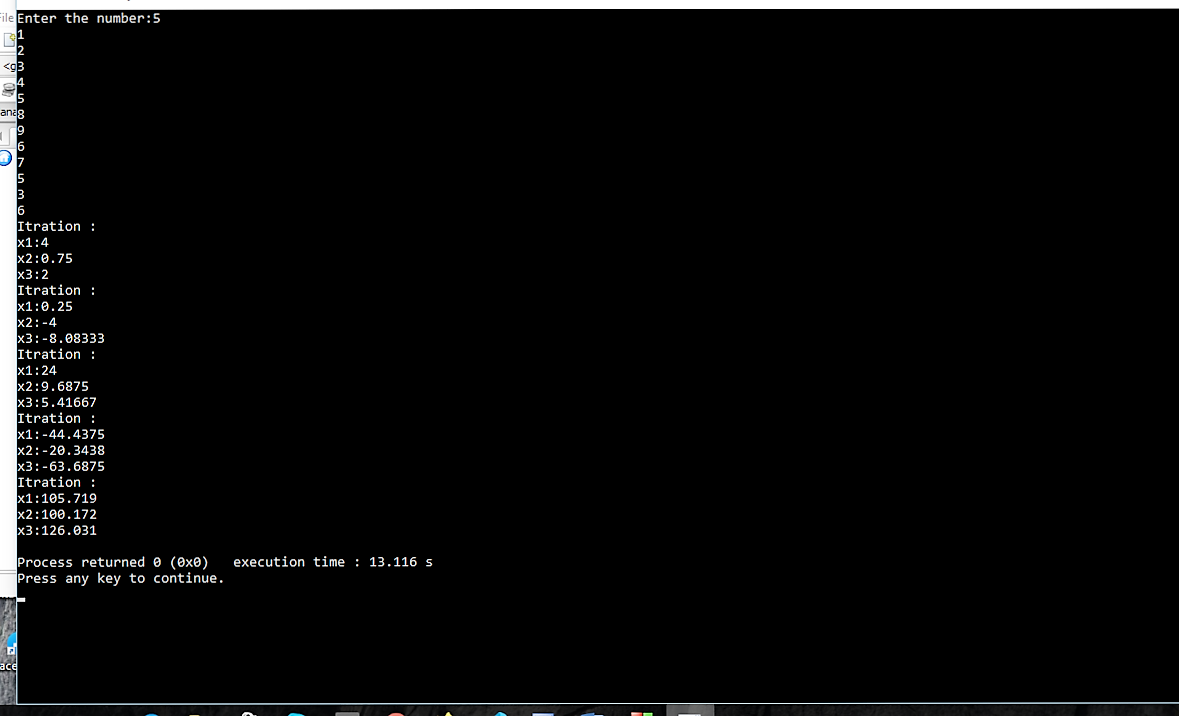
**Set xoi =xi**

**Go to step 3**

**6.Write result**

{\displaystyle k=k+1}

**Jacobi Iteration Output:**

****

***GAUSS-SEIDEL METHOD:***

**Gauss–Seidel Background:**

**Gauss–Seidel Algorithm:**

1.Obtain n ,aij and bi values

2.Set xi=b/ai for i=1 to n

3.Set key =0

4.For i=1 to n

(1)Set sum =bi

(2) For j =1 to n

Set sum =sum-aij-xi

Repeat j

(3)Set dummy =sum/ai

(4)if kew=0 then

(5)Set xi dummy

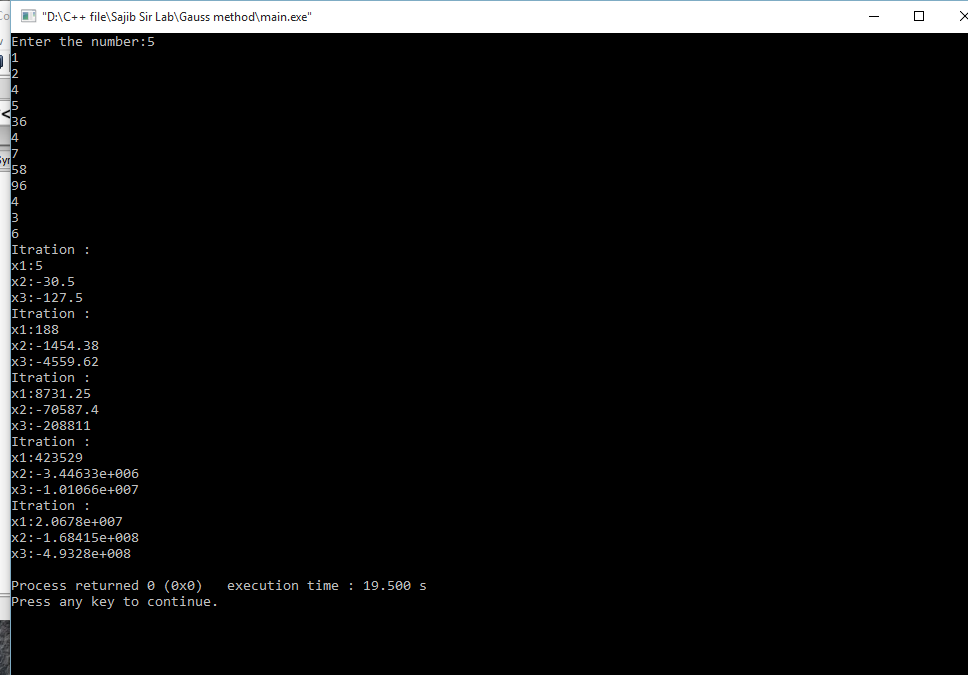
Repeat i

5.If kew =1 then

Go to step 3

6.Write results

**Gauss–Seidel Output:**

****