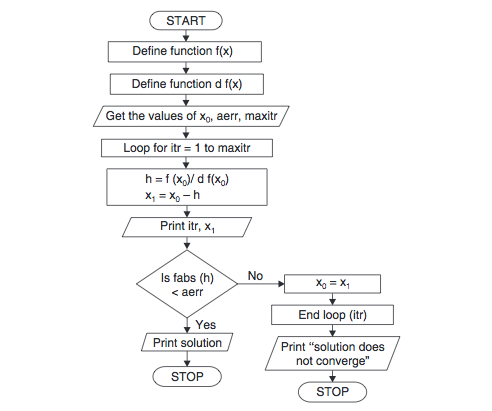
1.Newton Raphson Method Algorithm and Flowchart

Newton Raphson method, also called the Newton’s method, is the fastest and simplest approach of all methods to find the real root of a nonlinear function. It is an open bracket approach, requiring only one initial guess. This method is quite often used to improve the results obtained from other iterative approaches.

## **Newton Raphson Method Algorithm:**

1. Start
2. Read x, e, n, d  
   \*x is the initial guess  
   e is the absolute error i.e the desired degree of accuracy  
   n is for operating loop  
   d is for checking slope\*
3. Do for i =1 to n in step of 2
4. f = f(x)
5. f1 = f'(x)
6. If ( [f1] < d), then display too small slope and goto 11.  
   \*[ ] is used as modulus sign\*
7. x1 = x – f/f1
8. If ( [(x1 – x)/x1] < e ), the display the root as x1 and goto 11.  
   \*[ ] is used as modulus sign\*
9. x = x1 and end loop
10. Display method does not converge due to oscillation.
11. Stop

## **Newton Raphson Method Flowchart:**



Newton-Raphson Method in MATLAB

% Program Code of Newton-Raphson Method in MATLAB

a=input('Enter the function in the form of variable x:','s');

x(1)=input('Enter Initial Guess:');

error=input('Enter allowed Error:');

f=inline(a)

dif=diff(sym(a));

d=inline(dif);

for i=1:100

x(i+1)=x(i)-((f(x(i))/d(x(i))));

err(i)=abs((x(i+1)-x(i))/x(i));

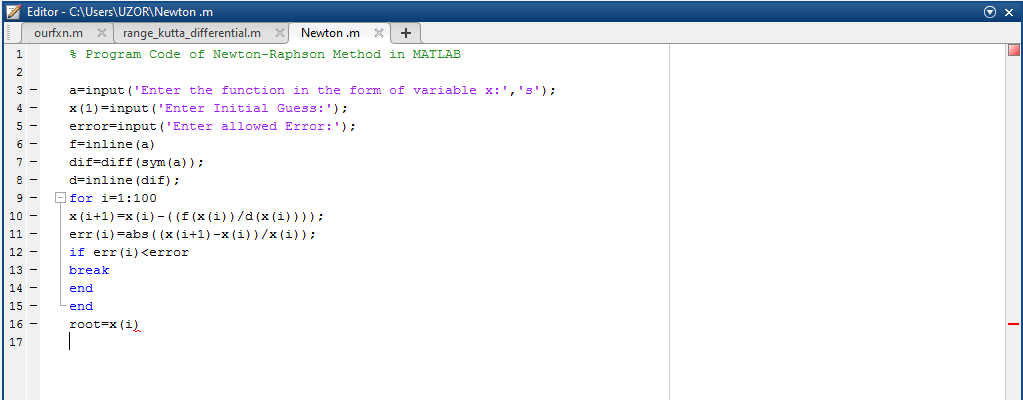
if err(i)<error

break

end

end

root=x



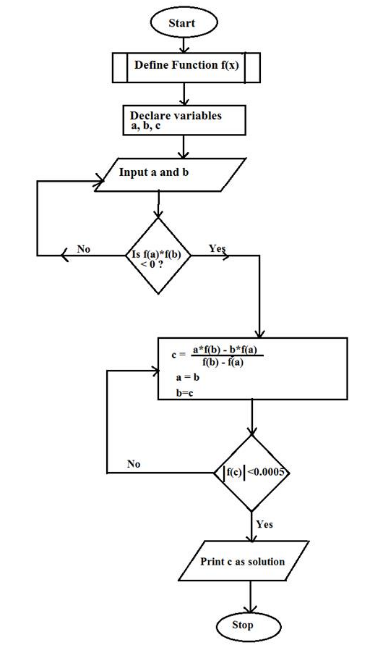
# 2. Secant Method Algorithm and Flowchart

Secant method is considered to be the most effective approach to find the root of a non-linear function. It is a generalized from the [Newton-Raphson method](https://www.codewithc.com/newton-raphson-method-algorithm-flowchart/) and does not require obtaining the derivatives of the function. So, this method is generally used as an alternative to Newton Raphson method.

**Secant Method Algorithm:**

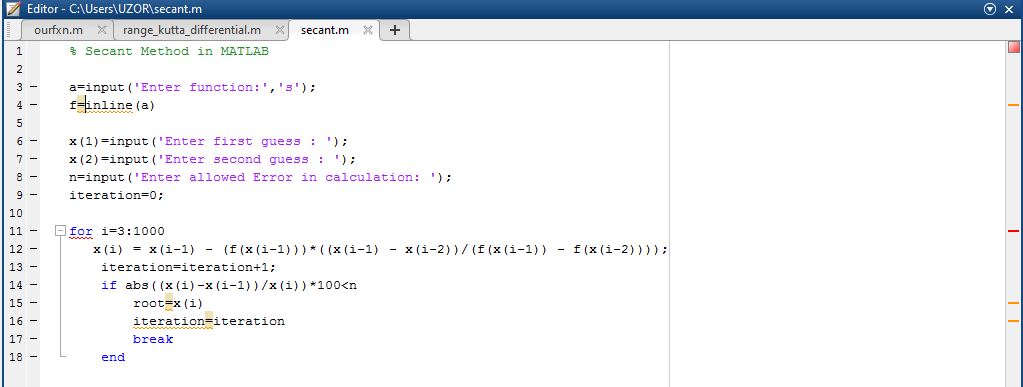
1. Start
2. Get values of x0, x1 and e  
   \*Here x0 and x1 are the two initial guesses  
   e is the stopping criteria, absolute error or the desired degree of accuracy\*
3. Compute f(x0) and f(x1)
4. Compute x2 = [x0\*f(x1) – x1\*f(x0)] / [f(x1) – f(x0)]
5. Test for accuracy of x2  
   If [ (x2 – 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 Flowchart:**



## **Secant Method in MATLAB:**

|  |  |
| --- | --- |
| 2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | % Secant Method in MATLAB    a=input('Enter function:','s');  f=inline(a)    x(1)=input('Enter first point of guess interval: ');  x(2)=input('Enter second point of guess interval: ');  n=input('Enter allowed Error in calculation: ');  iteration=0;    for i=3:1000     x(i) = x(i-1) - (f(x(i-1)))\*((x(i-1) - x(i-2))/(f(x(i-1)) - f(x(i-2))));      iteration=iteration+1;      if abs((x(i)-x(i-1))/x(i))\*100<n          root=x(i)          iteration=iteration          break      end  end |



3.Crammers Rule

Crammers rule algorithm.

1. Start

a11 a12 a13 x b1

1. Represent our linear equation in this form Ax=b = a21 a22 a23 y = b2

a31 a32 a33 z b3

1. Compute x =Ab
2. Find the Determinate of A and Ab
3. Compute x = Determinate of Ab/Determinate of A
4. Print valves of x
5. End

Crammers Rule flowcharts

Start

Represent our linear equation in this form Ax=b

x =Ab

Find the determinate of A and Ab

x = Determinate of Ab/Determinate of A

Print solution of x

END

ITERATIVE PROCESSES

1.Jacobi methods of literation

Jacobi algorithm

1. START
2. Express x1, x2 and x3 as a subject of the formula from our linear equation.
3. Initialize the values of x1, x2 and x3 = 0 and get new x1 x2 and x3 values
4. If new value of x1 x2 and x3 = True solution go to 5, Else go to 4
5. Print values of x1 x2 and x3
6. END

Flowchart

Start

Express x1 x2 x3 as subject formulas in our linear equation

Initialize x1 x2 and x3 = 0

And get new values of x1 x2 and x3

If new values of x1 x2 x3 =true solution

NO

Make new values of x1 x2and x3 = initial values

YES

Print solution of x

JACOBI METHODS IN MATLAB :

clc

A = [15 2 1; 2 20 -3; 3 -6 25];

B = [18; 19; 22;];

x1 = 0; x2 = 0; x3 = 0;

n = 0;

while(n < 5)

x1n = (B(1) - (A(1,2)\*x2 + A(1,3)\*x3))/A(1,1);

x2n = (B(2) - (A(2,1)\*x1 + A(2,3)\*x3))/A(2,2);

x3n = (B(3) - (A(3,1)\*x1 + A(3,2)\*x2))/A(3,3);

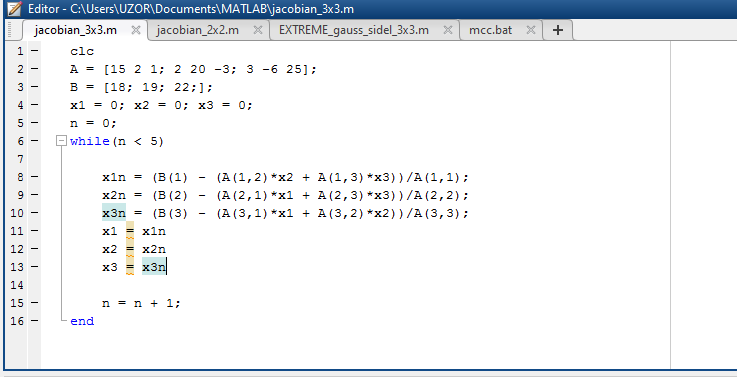
x1 = x1n

x2 = x2n

x3 = x3n

n = n + 1;

end



2. Gauss-seidel method.

Algorithm

1. START
2. Express x1, x2 and x3 as a subject of the formula from our linear equation.
3. Initialize x1 x2 x3 values =0
4. Using initial x2, x3 values get new x1 value.
5. Using initial values for x3 and new for x1 get new x2 value.
6. Using new values for x1 and x2 get new x3 value
7. If new value of x1 x2 and x3 = true solution go to 9.
8. Else make new x1 x2 and x3 = initial values and go to 4.
9. Print final values of x1 x2 and x3.
10. END

Flowchart

Start

Express x1 x2 x3 as subject formulas in our linear equation

Initialize x1 x2 and x3 = 0

Using initial x2,x3 get new x1

Using initial x1,x3 get new x2

Using initial x1,x2 get new x3

If new values of x1 x2 x3 =true solution

NO

Make new values of x1 x2and x3 = initial values

YES

Print solution of x

MATLAB CODE

clc

no\_of\_iterations = input('please input the number of iterations and press enter: ');

a = [5 -1 2; 3 8 -2; 1 1 4];

disp(a)

b = [12; -25; 8];

disp(b)

n = 3;

in = input('are the above matrixes correct? yes: press enter to continue. NO: press enter and rectify the matrix: ');

clc

x1 = 0; x2 = 0; x3 = 0;

X = [x1; x2; x3];

for k = 0:no\_of\_iterations

% k is the present iteration value from 0 to your input value

k

% solve for X(1)

i = 1;

disp('for X(1)')

%define its upper and lower

% limit

first\_term = 1/a(i,i)

second\_term = b(i)

% the third term is a summation, so we must def

upper\_limit = i - 1;

lower\_limit = 1;

if lower\_limit > upper\_limit

third\_term = 0

else

sum = 0; % initial value of sum is zero

for j = lower\_limit:upper\_limit

third\_term = a(i,j)\*X(j);

sum = sum + third\_term;

end

third\_term = sum

end

%fourth term is also a summation

%so like before we define an upper limit and a lower limit

upper\_limit = 3;

lower\_limit = i + 1;

if lower\_limit > upper\_limit

fourth\_term = 0

else

sum = 0;

for j = lower\_limit:upper\_limit

fourth\_term = a(i,j)\*X(j);

sum = sum + fourth\_term;

end

fourth\_term = sum

end

X(1) = (first\_term)\*(second\_term - third\_term - fourth\_term);

% end of X(1)

% solve for X(2)

i = 2;

disp('for X(2)')

first\_term = 1/a(i,i)

second\_term = b(i)

% the third term is a summation, so we must define its upper and lower

% limit

upper\_limit = i - 1;

lower\_limit = 1;

if lower\_limit > upper\_limit

third\_term = 0

else

sum = 0; % initial value of sum is zero

for j = lower\_limit:upper\_limit

third\_term = a(i,j)\*X(j);

sum = sum + third\_term;

end

third\_term = sum

end

%fourth term is also a summation

%so like before we define an upper limit and a lower limit

upper\_limit = 3;

lower\_limit = i + 1;

if lower\_limit > upper\_limit

fourth\_term = 0;

else

sum = 0;

for j = lower\_limit:upper\_limit

fourth\_term = a(i,j)\*X(j);

sum = sum + fourth\_term;

end

fourth\_term = sum

end

X(2) = (first\_term)\*(second\_term - third\_term - fourth\_term);

% end of X(2)

% solve for X(3)

i = 3;

disp('for X(3)')

first\_term = 1/a(i,i)

second\_term = b(i)

% the third term is a summation, so we must define its upper and lower

% limit

upper\_limit = i - 1;

lower\_limit = 1;

if lower\_limit > upper\_limit

third\_term = 0;

else

sum = 0; % initial value of sum is zero

for j = lower\_limit:upper\_limit

third\_term = a(i,j)\*X(j);

sum = sum + third\_term;

end

third\_term = sum

end

%fourth term is also a summation

%so like before we define an upper limit and a lower limit

upper\_limit = 3;

lower\_limit = i + 1;

if lower\_limit > upper\_limit

fourth\_term = 0

else

sum = 0;

for j = lower\_limit:upper\_limit

fourth\_term = a(i,j)\*X(j);

sum = sum + fourth\_term;

end

fourth\_term = sum

end

X(3) = (first\_term)\*(second\_term - third\_term - fourth\_term);

% end of X(3)

X

end

NEWTON COTES FORMULA AND METHODS

Trapezoidal rule

1. START
2. *Read b*

*F(x)dx =k*

*a*

1. compute h= b-a/n where b and a are limits of intervals and n = number of approximation
2. Read Xi = a + ih, I = index number from 0,1,2,3… nth
3. k = h/2{F(x0) +2F(x1)+ 2 F(x2)……. 2 F(xn-1) + F(xn)}
4. Print solution for k
5. END

FLOWCHART

Start

F(x)dx =k

n = number of approximation and

I = index number from 0,1,2,3… nth

h= b-a/n

Xi = a + ih,

k = h/2{F(x0) +2F(x1)+ 2 F(x2)……. 2 F(xn-1) + F(xn)}

Print solution of k

END

SIMPSON RULE

Algorithm.

1. START
2. *Read b*

*F(x)dx =k*

*a*

1. compute h= b-a/n where b and a are limits of intervals and n = number of approximation
2. Read Xi = a + ih, I = index number from 0,1,2,3… nth
3. If n = even number go to 5 Else use trapezoidal rule methods.
4. k = h/2{F(x0) +2F(x1)+ 2 F(x2)……. 2 F(xn-1) + F(xn)}
5. Print solution for k
6. END

Start

F(x)dx =k

n = number of approximation and

I = index number from 0,1,2,3… nth

h= b-a/n

Xi = a + ih,

Is n= even number

NO

YES

k = h/2{F(x0) +2F(x1)+ 2 F(x2)……. 2 F(xn-1) + F(xn)}

USE Trapezoidal rule

Print solution of k

END