)8 1ÕÅÓÔÉÏÎ - ÁÔÌÁÂ #ÏÄÅÓ

```
function [xr,T] = muller(f,xr,h,e)

fx = inline(f); %function inserted as a string converted

%fundamental parameters
x2 = xr;
x1 = xr+h;
x0 = xr-h;

k = 0;
devam = 1;

T=[0 0 0 0 0];  % Matrix that shows x0 - x1 - x2 - x3 - e respectively

while(devam)
    k= k+1;

    h0 = x1-x0;
    h1 = x2-x1;
    d0 = (fx(x1) - fx(x0))/h0;
    d1 = (fx(x2)-fx(x1))/h1;
    a = (d1-d0)/(h1+h0);
    b = (a*h1)+d1 ;
    c = fx(x2);

 kok= sqrt(b^2-4*a*c);
 if abs(b+kok) > abs(b-kok)
     bol = b+kok;
 else
     bol = b-kok;
 end
 xr=x2+(-2*c/bol);
 devam = abs(xr-x2)/xr > e;
 x0=x1;
 x1=x2;
 x2=xr;
 T(k,:)=[x0 x1 x2 fx(xr) e]

end
```

)8 1ÕÅÓÔÉÏÎ -ÁÔÌÁÂ 2ÅÓÕÌÔÓ

```
>> f='x^4+2*x^2-x-3';
muller(f,1,0.1,0.0001)

T =

    1.1000    1.0000    1.1244    0.0028    0.0001


T =

    1.1000    1.0000    1.1244    0.0028    0.0001
    1.0000    1.1244    1.1241   -0.0000    0.0001


T =

    1.1000    1.0000    1.1244    0.0028    0.0001
    1.0000    1.1244    1.1241   -0.0000    0.0001
    1.1244    1.1241    1.1241   -0.0000    0.0001


ans =

    1.1241

>>
```

))8 1ÕÅÓÔÉÏÎ -ÁÔÌÁÂ #ÏÄÅÓ

```matlab
function x = gausseidel(A,B,es)

%Inputs:

% A = Coefficent Matrix
% B = Right-Hand Side Vector
% es = stop criterion (default = 0.00001%)

%Outputs:
% x = solution vector

[m,n] = size(A);
if m ~= n, error('Matrix A must be square'); end
C = A;

for i=1:n
    C(i,i)=0;
    x(i)=0;
end

x= x';

for i=1:n
    C(i,1:n)=C(i,1:n)/A(i,i);
end

for i=1:n
    d(i)=B(i)/A(i,i);
end

iter = 0;
while(1)
    xold = x;
    fprintf('\n %d. Iteration! \n', iter);
    x
    for i=1:n
        x(i)=d(i)-C(i,:)*x;

        if x(i) ~= 0
            ea(i)=abs((x(i)-xold(i))/x(i))*100;
        end
    end
    ea
    iter = iter +1;
    if norm(ea,Inf)<=es break, end
end
for i= 1:length(x)
    fprintf('\nx%d = %f\n', i, x(i));
end
```

))8 1ÕÅÓÔÉÏÎ -ÁÔÌÁÂ 2ÅÓÕÌÔÓ

```
>> gausseidel(A,B,0.001)

 0. Iteration!

x =

     0
     0
     0
     0


ea =

   100   100   100   100


 1. Iteration!

x =

   -0.5000
   -0.1250
   -0.1250
    0.5000


ea =

   20.0000  100.0000   44.4444   18.9189


 2. Iteration!

x =

   -0.6250
         0
   -0.2250
    0.6167


ea =

   12.0235  100.0000   13.5654    7.3262
 3. Iteration!

x =

   -0.7104
    0.0255
   -0.2603
    0.6654


ea =

    3.7131   28.5714    4.8218    2.4812


 4. Iteration!

x =

   -0.7378
    0.0357
   -0.2735
    0.6823


ea =

    1.3480    8.8200    1.6934    0.8836


 5. Iteration!

x =

   -0.7479
    0.0392
   -0.2782
    0.6884


ea =

    0.4741    3.0520    0.6011    0.3127


 6. Iteration!

x =

   -0.7515
    0.0404
   -0.2799
    0.6906


ea =

    0.1686    1.0684    0.2133    0.1111
 7. Iteration!

x =


   -0.7527
    0.0409
   -0.2805
    0.6914


ea =

    0.0598    0.3781    0.0758    0.0394


 8. Iteration!

x =

   -0.7532
    0.0410
   -0.2807
    0.6916


ea =

    0.0213    0.1341    0.0269    0.0140


 9. Iteration!

x =

   -0.7533
    0.0411
   -0.2808
    0.6917


ea =

    0.0076    0.0476    0.0096    0.0050


 10. Iteration!

x =

   -0.7534
    0.0411
   -0.2808
    0.6918


ea =

    0.0027    0.0169    0.0034    0.0018
 11. Iteration!

x =

   -0.7534
    0.0411
   -0.2808
    0.6918


ea =

    0.0010    0.0060    0.0012    0.0006


 12. Iteration!

x =

   -0.7534
    0.0411
   -0.2808
    0.6918


ea =

    0.0003    0.0021    0.0004    0.0002


 13. Iteration!

x =

   -0.7534
    0.0411
   -0.2808
    0.6918


ea =

   1.0e-03 *

    0.1203    0.7581    0.1523    0.0793


x1 = -0.753424

x2 = 0.041096

x3 = -0.280822

x4 = 0.691781

ans =

   -0.7534
    0.0411
   -0.2808
    0.6918

>>
```

))8Â - ÁÔÌÁÂ #ÏÄÅÓ

```
function x = sorkod(A,B,es,w)

%Inputs:

% A = Coefficent Matrix
% B = Right-Hand Side Vector
% es = stop criterion (default = 0.00001%)  Detailed explanation goes here
% w = relaxation constant (Must be between 0 and 2)

%Outputs:
% x = solution vector

[m,n] = size(A);
if m ~= n, error('Matrix A must be square'); end
C = A;

for i=1:n
    C(i,i)=0;
    x(i)=0;
end

x= x';

for i=1:n
    C(i,1:n)=C(i,1:n)/A(i,i);
end

for i=1:n
    d(i)=B(i)/A(i,i);
end

iter = 0;
while(1)
    xold = x;
    fprintf('\n %d. Iteration! \n', iter);
    x
    for i=1:n
        x(i)=d(i)-C(i,:)*x;
        x(i)=(1-w)*xold(i)+w*(d(i)-C(i,:)*x);
        if x(i) ~= 0
            ea(i)=abs((x(i)-xold(i))/x(i))*100;
        end
    end
    ea
    iter = iter +1;
    if norm(ea,Inf)<=es break, end
end
for i= 1:length(x)
    fprintf('\nx%d = %f\n', i, x(i));
end
```

))8Â - ÁôÌÁÂ 2ÅóÕÌôó

```
>> sorkod(A,B,0.001,1.1)

 0. Iteration!

x =

     0
     0
     0
     0


ea =

   100   100   100   100


 1. Iteration!

x =

   -0.5500
   -0.1237
   -0.1482
    0.5773


ea =

   16.7285  434.3918   40.5558   12.0111


 2. Iteration!

x =

   -0.6605
    0.0370
   -0.2494
    0.6561


ea =

   11.1206    1.3659    9.2094    4.6342


 3. Iteration!

x =

   -0.7431
    0.0375
   -0.2746
    0.6880


ea =

    1.0124    9.3086    1.8967    0.4280
 4. Iteration!

x =

   -0.7507
    0.0414
   -0.2800
    0.6910


ea =

    0.3415    0.7907    0.2697    0.1165


 5. Iteration!

x =

   -0.7533
    0.0410
   -0.2807
    0.6918


ea =

    0.0111    0.1675    0.0373    0.0020


 6. Iteration!

x =

   -0.7534
    0.0411
   -0.2808
    0.6918


ea =

    0.0057    0.0489    0.0023    0.0014



 7. Iteration!

x =

   -0.7534
    0.0411
   -0.2808
    0.6918


ea =

    0.0007    0.0033    0.0000    0.0004
 8. Iteration!

x =

   -0.7534
    0.0411
   -0.2808
    0.6918


ea =

    0.0000    0.0018    0.0001    0.0000


 9. Iteration!

x =

   -0.7534
    0.0411
   -0.2808
    0.6918


ea =

   1.0e-03 *

    0.0443    0.1338    0.0239    0.0174


x1 = -0.753425

x2 = 0.041096

x3 = -0.280822

x4 = 0.691781

ans =

   -0.7534
    0.0411
   -0.2808
    0.6918
```

)))8 1ÕÅÓÔÉÏÎ -ÁÔÌÁÂ #ÏÄÅÓ

```
function x0= newton(f,x0,es)
%Performs Newton's for the function defined in f starting with x0 and...
%...running maximum of 100 times.

[y,dy]=f(x0);         % x0 values inserted at f function, which gives y and
                      % dy as outputs.
n=length(x0);

i=1;

while i<100
  s= -(dy)\y;         % x1=x0-(J^-1)*F   Newton-Raphson's Method
  err = norm(s, Inf);         % x1-x0 = -(J^-1)*F = error
  x1=x0+s;
  x0=x1;              % assigned for iteration
  [y,dy]=f(x0);       % new x0 inserted to f function

  fprintf('%d. Iteration!\n',i);

  x0

  y

  err

 i=i+1;

 if err <= es break,

 end
end


function [y,dy]=question(x)
% The function and its jacobian matrix

n=length(x);
y=zeros(size(x)); %setting up initial values for both y and dy
dy=zeros(n,n);

y(1)= 3*x(1)-cos(x(2)*x(3))-(1/2);
y(2)= x(1)^2-(81*(x(2)+0.1)^2)+sin(x(3))+(1.06);
y(3)= exp((-x(1)*x(2)))+20*x(3)+(10*pi-3)/3;

% creating jacobian matrix...

dy(1,1)= 3; %dy(1)/dx(1)
dy(1,2)= x(3)*sin(x(3)*x(2)); %dy(1)/dx(2)
dy(1,3)= x(2)*sin(x(3)*x(2)); %dy(1)/dx(3)

dy(2,1)= 2*x(1); %dy(2)/dx(1)
dy(2,2)= -2*81*(x(2)+0.1); %dy(2)/dx(2)
dy(2,3)= cos(x(3)); %dy(2)/dx(3)

dy(3,1)= -x(2)*exp(-x(1)*x(2)); %dy(3)/dx(1)
dy(3,2)= -x(1)*exp(-x(1)*x(2)); %dy(3)/dx(2)
dy(3,3)= 20; %dy(3)/dx(3)
```

)))8 1ÕÅÓÔÉÏÎ -ÁÔÌÁÂ 2ÅÓÕÌÔÓ

```
>> x0=[0.1;0.1;-0.1];
>> newton(@question,x0,0.001);
1. Iteration!

x0 =

    0.4999
    0.0195
   -0.5215


y =

   -0.0003
   -0.3444
    0.0319


err =

    0.4215

2. Iteration!

x0 =

    0.5000
    0.0016
   -0.5236


y =

    0.0000
   -0.0259
    0.0000


err =

    0.0179

3. Iteration!

x0 =

    0.5000
    0.0000
   -0.5236


y =

   1.0e-03 *
    0.0003
   -0.2012
    0.0003


err =

    0.0016
```

```
4. Iteration!

x0 =

    0.5000
    0.0000
   -0.5236


y =

   1.0e-07 *

    0.0002
   -0.1254
    0.0002


err =

   1.2444e-05

>>
```