

```
% <---- This is the main Matlab script file called main1.m ---->
```

```
% call Newton Raphson
```

```
X0 = [ 0.1 0.2 ]' ; % Specify initial guess
```

```
tolerance = 1e-8 ; % Specify tolerance
```

```
max_iter = 100 ; % Specify maximum number of iterations
```

```
Xk = newt_raph( 'fun1' , 'grad1' , X0, tolerance, max_iter ) ;
```

```
FofX = fun1( Xk ) % Check the solution
```

```
norm(FofX)
```

```
% <----->
```

```
% function newt_raph.m
```

```
% This function implements Newton-Raphson algorithm for multi-variate case
```

```
% The gradienty information necessary at each step is evaluated numerically
```

```
% using another matlab function called 'gradient'.
```

```
% Input arguments to the function are
```

```
% fun_name : String containing name of the MATLAB function
```

```
% which returns function vector F(x) given x
```

```
% grad_fun: String containing name of the MATLAB function
```

```
% which returns Jacobian of the function vector F(x) given x
```

```
% x0 : Initial guess vector
```

```
% tolerance : Tolerance for convergence
```

```
% maxiter : Maximum number of iterations
```

```
% Output arguments of the function are
```

```
% xstar : Solution of F(x) = 0
```

```
% -----
```

```
function [ xstar ] = newt_raph( fun_name,grad_fun, x0, tolerance, maxiter )
```

```
conv_criterion = 1 ;
```

```
iter_count = 1 ;
```

```
while( ( conv_criterion > tolerance ) & ( iter_count <= maxiter ))
```

```
    fx0 = feval( fun_name, x0 );
    gradFx0 = feval( grad_fun, x0 );
    xnew = x0 - inv(gradFx0) * fx0 ;
    conv_criterion = norm(xnew-x0, 2) / norm(xnew, 2) ;
    x0 = xnew ;
    iter_count = iter_count + 1 ;
    [ iter_count xnew' ]
    fprintf('\n \t Press any key to continue...\n'), pause
```

```
end
```

```
xstar = xnew ;
```

```
% <----->
% function grad1.m
% This function computes Jacobian matrix for given functions
% Input arguments to the function are
% fun_name : String containing name of the MATLAB function,
%            which returns function vector F(x) given x
% x0       : vector at which gradient should be computed
% Output arguments of the function are
% gradF    : (n x n) gradient matrix
% -----
```

```
function gradF = grad1( Xk )
```

```
gradF(1,1) = 2*Xk(1) ;
gradF(1,2) = 2*Xk(2) ;
gradF(2,1) = Xk(2) ;
gradF(2,2) = Xk(1) ;
```

```
% <----- function fun1.m ----->  
% Given guess for X, this function returns  
% function vector evaluated at the guess values  
% -----
```

```
function FofX = fun1( Xk )
```

```
FofX(1) = Xk(1)^2 + Xk(2)^2 - 4 ;
```

```
FofX(2) = Xk(1) * Xk(2) - 1 ;
```

```
FofX= FofX' ;           % make FofX into a column vector
```