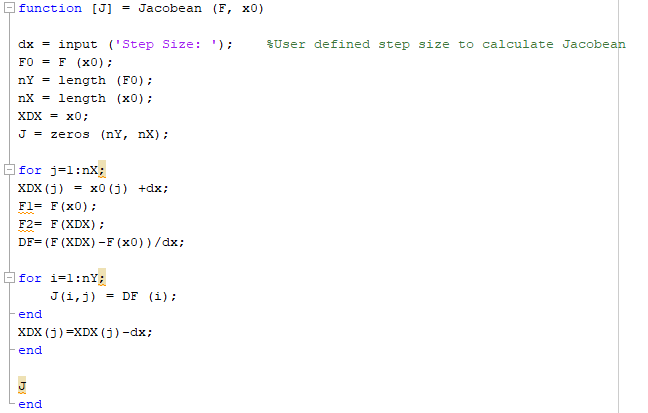
**Assignment 1**

**Question 3**

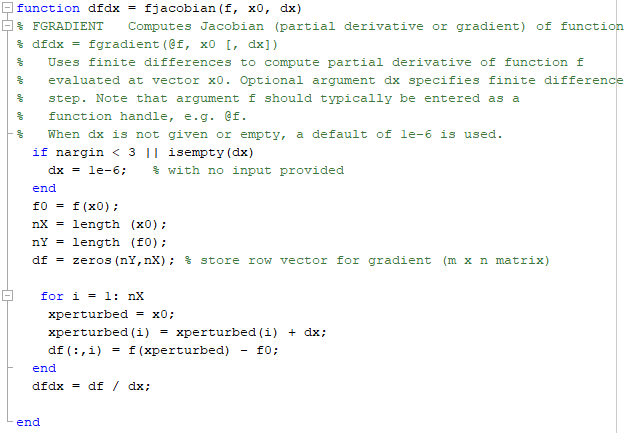
1. The developed function is developed using the following argument:

Therefore for a function of (dimension: m) depending of (dimension: n), the Jacobean is:

I developed the following matlab code to figure out how to modify the started code:

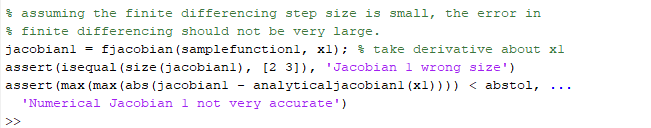


The developed function: fjacobian:



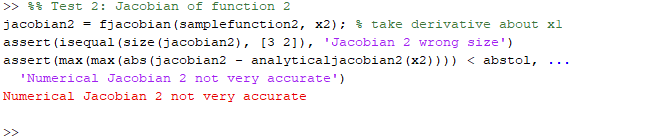
Test 1:

The developed function passed the first test:

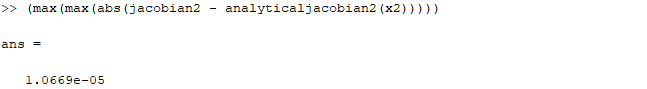


Test 2:

The developed function failed the second test (accuracy):



Developed routine accuracy:

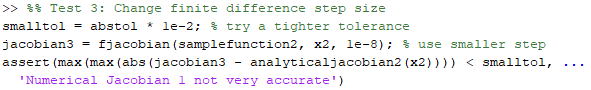


Required accuracy:

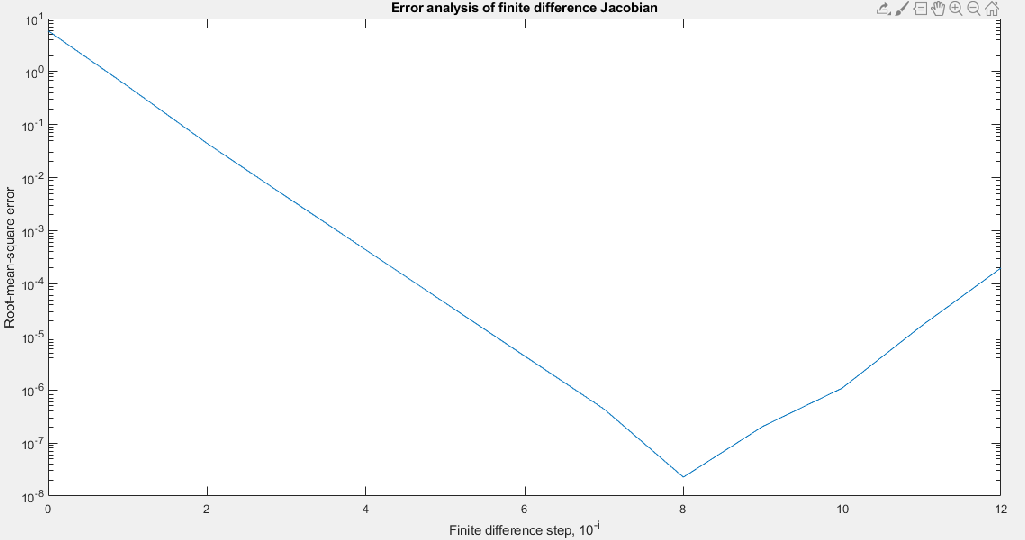


Test 3:

Developed routine passed the test 3:



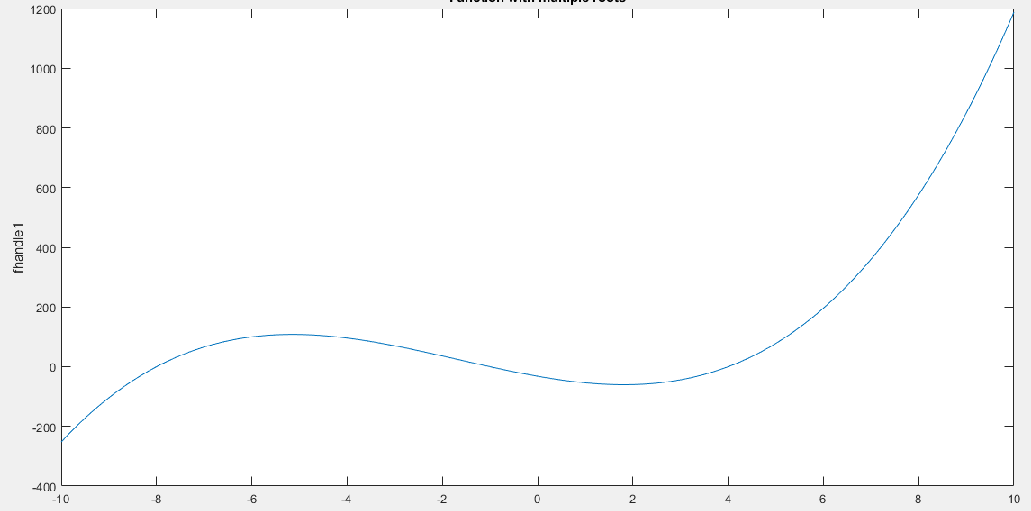
Test 4:

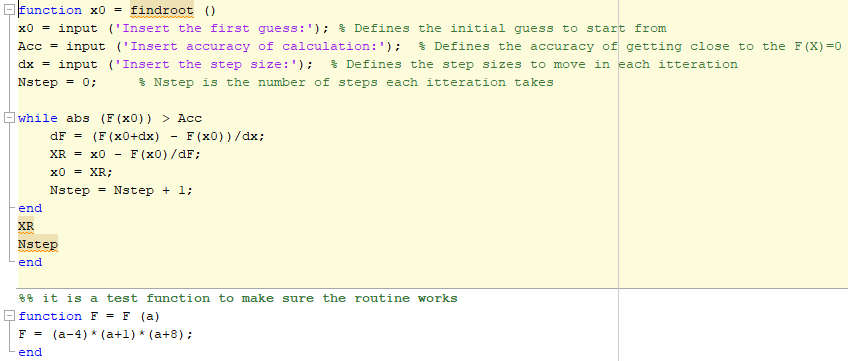


Based on the test 4 the error decreases by shortening the increment size of dx to a certain point. I believe it is due to the truncation of the higher orders of the Taylor series (representing a differentiation) getting more accurate. However, after a certain point the error increases which I thing is due to rounding error of the computing tool.

**Question 4:**

Running the testfindroot.m (the graph representing the test function):



I developed the following routine performing simple Newton method to modify the started file:

The resulted modified function (findroot.m):

function [xstar, cnvrg] = findroot(f, x0)

% FINDROOT Finds the root of a vector function, with vector-valued x0.

%

% xstar = findroot(f, x0) performs a Newton search and returns xstar,

% the root of the function f, starting with initial guess x0. The

% function will typically be expressed with a function handle, e.g.

% @f.

% parameter structure for root-finding

parms.dxtol = 1e-6; % default tolerance for min change in x

parms.dftol = 1e-6; % default tolerance for min change in f

parms.maxiter = 1000; % allow max of 1000 iterations before quitting

parms.finitediffdx = []; % finite differencing step size

% We will take steps to improve on x, while monitoring

% the change dx, and stopping when it becomes small

dx = Inf; % initial dx is large

iter = 0; % count the iterations we go through

x = x0(:); % start at this initial guess (force to be column)

fprevious = f(x0); % use to compare changes in function value

df = Inf; % change in f is initialized as large

% Loop through the refinements, checking to make sure that x and f

% change by some minimal amount, and that we haven't exceeded

% the maximum number of iterations

while max(abs(dx)) >= parms.dxtol && max (abs (df)) >= parms.dftol% MODIFY this line to also test whether

% df (change in f(x)) exceeds its tolerance, and whether

% dx (change in x) exceeds its tolerance

% Here is the main Newton step

J = fjacobian (f, x0) % FILL in code here to calculate the gradient or jacobian : f-prime = J = (f(x+dx)-f(x))/dx

dx = -J\f(x) % FILL in this line to solve for the (vector) dx (change in x): x(New) = x(OLD) - f(x)/f-prime

x = x + dx; % Apply the calculated step to form the next x value

% Update information about changes

iter = iter + 1;

df = f(x) - fprevious; % change in f

fprevious = f(x);

end

xstar = x; % use the latest, best guess

cnvrg = true;

if iter > parms.maxiter % we probably didn't find a good solution

warning('Maximum iterations exceeded in findroot');

cnvrg = false;

end

if size(x0,2) > 1 % x0 was given to us as a row vector

xstar = xstar'; % so return xstar in the same shape

end

end % findroot

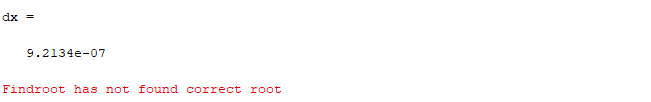
Test case 1 result (last iteration):



Test case 2:

Wrong initial guess causes the routine not to find the answer:



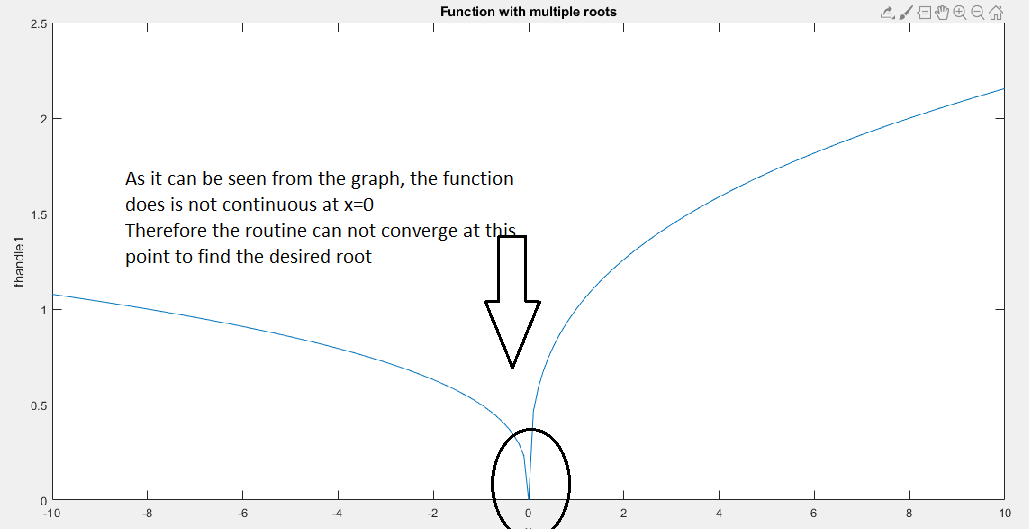


Test case 3:

Due to the nature of the function in which the root is positioned where the limits (right and left limits are not the same, therefore, the function is not continuous in that point, the routine does not converge:



The operation was disrupted by Ctrl+C



Test case 4:

The routine is able to handle the vector function presented. Derived results are as follows:



**Question 5:**