APPM3021 Numerical Methods • Lab 3
Tyson Cross 1239448

Exercises.

---

exercise1.m :

```
% APPM3021 Lab 3, Exercise 1

clc
clear global variable

equation = @(x) x^2 -x - 2;
I_0 = [1, 4];
tol = 0.00001;

tic;
it_root_bisec = bisectionSearch(equation, tol, I_0);
t_bisec = toc;
disp(['Solution converged in ', num2str(t_bisec*1000), ' milli-seconds'])
```

When exercise1.m is run in the workspace, the following output is displayed to the command window:

```
Root = 2 found by bisection method within tolerance: 1e-05 in 19 iterations
Solution converged in 11.1548 milli-seconds
```

---

exercise2.m :

```
% APPM3021 Lab 3, Exercise 2

clc
clear global variable

equation = @(x) x^2 -x - 2;
I_0 = [1, 4];
tol = 0.00001;

tic;
it_root_falsi = regulaFalsiSearch(equation, tol, I_0);
t_falsi = toc;
disp(['Solution converged in ', num2str(t_falsi*1000), ' milli-seconds'])
```

When exercise2.m is run in the workspace, the following output is displayed to the command window:

```
Root = 2 found by Regula-Falsi method within tolerance: 1e-05 in 14 iterations
Solution converged in 8.1286 milli-seconds
```

---

exercise3.m :

```
% APPM3021 Lab 3, Exercise 3

clc
clear global variable

syms x;
f = @(x) x^2 - x - 2;
x_0 = 1;
tol = 0.00001;

tic;
fprime = matlabFunction( diff(f(x)) );                          % include in timing
it_root_newton = NewtonMethodScaler(f, fprime, x_0, tol);
t_newton = toc;
disp(['Solution converged in ', num2str(t_newton*1000), ' milli-seconds (including calculation of f'')'])
```

When exercise3.m is run in the workspace, the following output is displayed to the command window:

```
Root = 2 found by Newton method within tolerance: 1e-05 in 7 iterations
Solution converged in 89.4009 milli-seconds (including calculation of f')
```

---

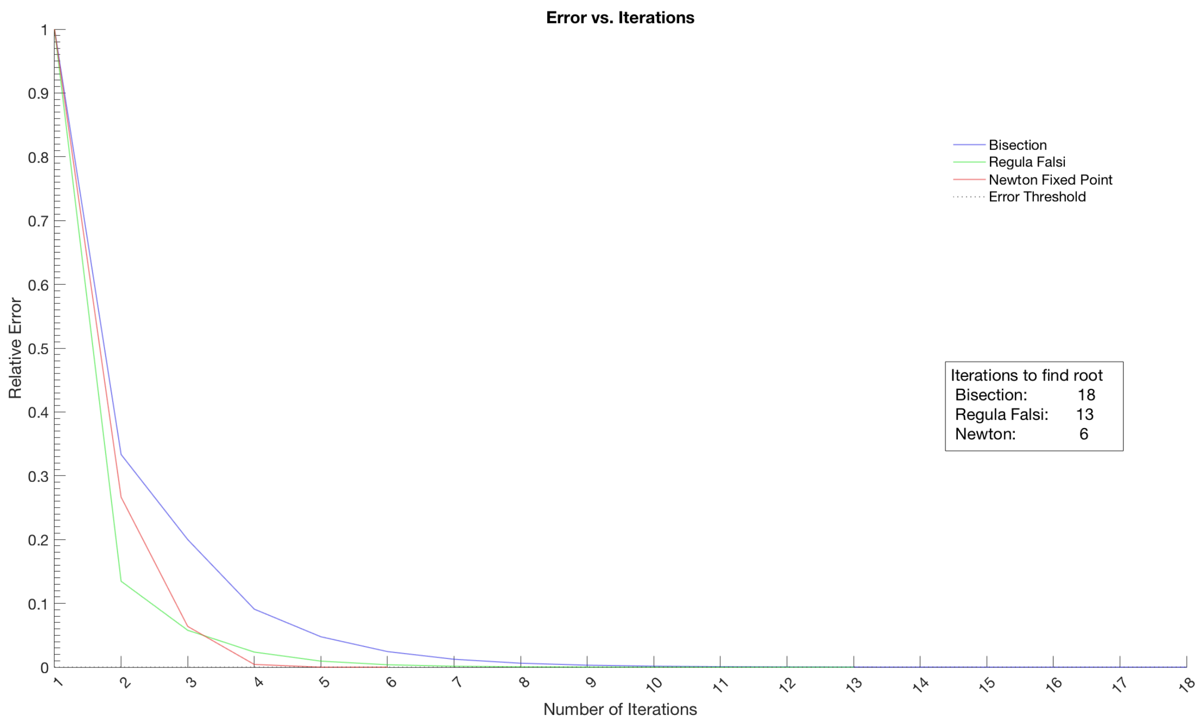
exercise4.m :

```
% APPM3021 Lab 3, Exercise 4

clc
clear global variable

syms f x;
```

```
f = @(x) 2*x^3 - -x^2 - exp(x) - 2.2;

warning('off');

x_0 = 1;
I_0 = [1, 2];
tol = 0.00001;

% measurements and timing
tic;
root_bisec = bisectionSearch(f, tol, I_0, true);
t_bisec = toc; tic;
root_falsi = regulaFalsiSearch(f, tol, I_0, true);
t_falsi = toc; tic;
fprime = matlabFunction( diff(f(x)) );                      % included in timing
root_newton = NewtonMethodScaler(f, fprime, x_0, tol, true);
t_newton = toc;

% iterations
iter_bisec = length(root_bisec);
iter_falsi = length(root_falsi);
iter_newton = length(root_newton);

% relative error
error_bisec(1) = I_0(2)-I_0(1);
error_falsi(1) = I_0(2)-I_0(1);
error_newton(1) = x_0;

for index=2:iter_bisec
    difference = abs(root_bisec(:,index) - root_bisec(:,index-1));
    error_bisec(index) = max(difference)/max(abs(root_bisec(:,index)));
end
for index=2:iter_falsi
    difference = abs(root_falsi(:,index) - root_falsi(:,index-1));
    error_falsi(index) = max(difference)/max(abs(root_falsi(:,index)));
end
for index=2:iter_newton
    difference = abs(root_newton(:,index) - root_newton(:,index-1));
    error_newton(index) = max(difference)/max(abs(root_newton(:,index)));
end

% time
disp(' ')
disp(['Bisection root found in ', num2str(t_bisec*1000), ' milli-seconds'])
disp(['Regula Falsi root found in ', num2str(t_falsi*1000), ' milli-seconds'])
disp(['Newton fixed-point root found in ', num2str(t_newton*1000), ' milli-seconds (including calculation of
 f'')'])


%% Main plot
% Display setting and output setup
fig1 =  figure('Position',...                               % draw figure
    [1 scr(4)*3/5 scr(3)*3.5/5 scr(4)*3/5]);
set(fig1,'numbertitle','off',...                            % Give figure useful title
    'name','Comparison of iterative root-finding methods',...
    'Color','white');
set(fig1, 'MenuBar', 'none');                               % Make figure clean
set(fig1, 'ToolBar', 'none');
fontName='Helvetica';
set(0,'defaultAxesFontName', fontName);                     % Make fonts pretty
set(0,'defaultTextFontName', fontName);
set(groot,'FixedWidthFontName', 'ElroNet Monospace')

% Plot
p1 = plot(error_bisec,...
        'Color',[0.18 0.18 0.9 .6],...
        'LineStyle','-',...
        'LineWidth',1);
hold on
p2 = plot(error_falsi,...
        'Color',[0.18 0.9 0.18 .6],...
        'LineStyle','-',...
        'LineWidth',1);
hold on
p3 = plot(error_newton,...
        'Color',[0.9 0.18 0.18 .6],...
        'LineStyle','-',...
        'LineWidth',1);
hold on
p4 = refline(0,tol);
set(p4,'Color',[0.18 0.18 0.18 .6],...
        'LineStyle',':',...
        'LineWidth',1);
hold on

% Title
title('Error vs. Iterations',...
    'FontSize',14,...
    'FontName',fontName);

% Annotations
    info_pos =   [0.74 0.3 0.5 0.2];
    str_info = {'Iterations to find root',...
            [' Bisection:            ', num2str(iter_bisec)],...
            [' Regula Falsi:       ', num2str(iter_falsi)],...
```

```
            [' Newton:                ', num2str(iter_newton)]};
    info = annotation('textbox',info_pos,...
        'String', str_info,...
        'FitBoxToText','on',...
        'LineStyle','-',...
        'FontName',fontName,...
        'FontSize',15);

% Axes and labels
ax1 = gca;
ylabel('Relative Error',...
    'FontName',fontName,...
    'FontSize',14);%,...
xlabel('Number of Iterations',...
    'FontName',fontName,...
    'FontSize',14);
max_x = max(iter_bisec(1,1),iter_falsi(1,1));
xlim(ax1,[1 max_x]);
box(ax1,'off');
set(ax1,'FontSize',14,...
    'XTick',[0:1:max_x],...
    'XTickLabelRotation',45,...
    'YMinorTick','on');hold on

% Legend
legend1 = legend({'Bisection','Regula Falsi',...
     'Newton Fixed Point', 'Error Threshold'},...
     'Position',[0.7    0.7    0.2    0.09],...
     'Box','off',...
     'FontName',fontName,...
     'FontSize',13);
hold off
% epswrite('images/relative_error.eps');
```

Figure 1. Comparison of iterative root-finding methods



Observances:

The Newton Method initially converges towards a root the fastest, but gets progressively

---

exercise5.m :

```
% APPM3021 Lab 3, Exercise 5

clc
clear global variable

% system of equations
syms f g x y;
f(x,y) = x^2 + y^2 - 2.12;
g(x,y) = y^2 - x^2*y - 0.04;

F = [f;g];
J = jacobian(F, [x, y]);
```

```
X_0 = [ 1 1 ];
tol = 0.00001;

tic;
it_root_newton_sys = NewtonMethodSystem(F, J, X_0, tol);
t_newton_sys = toc;
disp(['Solution converged in ', num2str(t_newton_sys*1000), ' milli-seconds (including calculation of f'')'])
```

Questions.

Question 1 (a) (i)

```
% APPM3021 Lab 3, Question 1 (a) (I)

clc
clear all

syms x;
f = @(x) exp(x) + 2^(-x) + 2*cos(x) - 6
x_0 = 2;
I_0 = [1, 2];
tol = 0.00001;

% measurements
tic;
root_bisec = bisectionSearch(f, tol, I_0,true);
t_bisec = toc; tic;
root_falsi = regulaFalsiSearch(f, tol, I_0,true);
t_falsi = toc; tic;
fprime = matlabFunction( diff(f(x)) );                      % included in Newton timing
root_newton = NewtonMethodScaler(f, fprime, x_0, tol,true);
t_newton = toc;

% iterations
iter_bisec = length(root_bisec);
iter_falsi = length(root_falsi);
iter_newton = length(root_newton);

% relative error
error_bisec(1) = I_0(2)-I_0(1);
error_falsi(1) = I_0(2)-I_0(1);
error_newton(1) = x_0;

for index=2:iter_bisec
    difference = abs(root_bisec(:,index) - root_bisec(:,index-1));
    error_bisec(index) = max(difference)/max(abs(root_bisec(:,index)));
end
for index=2:iter_falsi
    difference = abs(root_falsi(:,index) - root_falsi(:,index-1));
    error_falsi(index) = max(difference)/max(abs(root_falsi(:,index)));
end
for index=2:iter_newton
    difference = abs(root_newton(:,index) - root_newton(:,index-1));
    error_newton(index) = max(difference)/max(abs(root_newton(:,index)));
end

% time
disp(' ')
disp(['Bisection root converged in ', num2str(t_bisec*1000), ' milli-seconds'])
disp(['Regula Falsi root converged in ', num2str(t_falsi*1000), ' milli-seconds'])
disp(['Newton fixed-point root converged in ', num2str(t_newton*1000), ' milli-seconds (including calculation of
 f'')'])

%% Main plot
% Display setting and output setup
fig1 =  figure('Position',...                               % draw figure
    [1 scr(4)*3/5 scr(3)*3.5/5 scr(4)*3/5]);
set(fig1,'numbertitle','off',...                            % Give figure useful title
    'name','Comparison of iterative root-finding methods',...
    'Color','white');
set(fig1, 'MenuBar', 'none');                               % Make figure clean
set(fig1, 'ToolBar', 'none');
% fontName='CMU Serif';
fontName='Helvetica';
set(0,'defaultAxesFontName', fontName);                     % Make fonts pretty
set(0,'defaultTextFontName', fontName);
set(groot,'FixedWidthFontName', 'ElroNet Monospace')

%% Plot
p1 = plot(error_bisec,...
        'Color',[0.18 0.18 0.9 .6],...
        'LineStyle','-',...
        'LineWidth',1);
```

```
hold on
p2 = plot(error_falsi,...
        'Color',[0.18 0.9 0.18 .6],...
        'LineStyle','-',...
        'LineWidth',1);
hold on
p3 = plot(error_newton,...
        'Color',[0.9 0.18 0.18 .6],...
        'LineStyle','-',...
        'LineWidth',1);
hold on
p4 = refline(0,tol);
set(p4,'Color',[0.18 0.18 0.18 .6],...
        'LineStyle',':',...
        'LineWidth',1);
hold on

% Title
title('Error vs. Iterations',...
    'FontSize',14,...
    'FontName',fontName);

% Annotations
    info_pos =   [0.74 0.3 0.5 0.2];
    str_info = {'Iterations to find root',...
            [' Bisection:           ', num2str(iter_bisec)],...
            [' Regula Falsi:      ', num2str(iter_falsi)],...
            [' Newton:              ', num2str(iter_newton)]};
    info = annotation('textbox',info_pos,...
        'String', str_info,...
        'FitBoxToText','on',...
        'LineStyle','-',...
        'FontName',fontName,...
        'FontSize',15);

% Axes and labels
ax1 = gca;
% hold(ax1,'on');
ylabel('Relative Error',...
    'FontName',fontName,...
    'FontSize',14);%,...
xlabel('Number of Iterations',...
    'FontName',fontName,...
    'FontSize',14);
max_x = max(iter_bisec(1,1),iter_falsi(1,1))+1;
% ax1.XLim = [1 max_x];
box(ax1,'off');
set(ax1,'FontSize',14,...
    'XLim',[1 max_x],...
    'XTick',[0:1:max_x],...
    'XTickLabelRotation',45,...
    'YMinorTick','on');hold on

% Legend
legend1 = legend({'Bisection','Regula Falsi','Newton Fixed Point', 'Error Threshold'},...
     'Position',[0.7    0.7    0.2    0.09],...
     'Box','off',...
     'FontName',fontName,...
     'FontSize',13);
hold off
% epswrite('images/relative_error.eps');
```

When question1a_I.m is run in the workspace, the following output is displayed to the command window:

```
Undefined function 'scr' for input arguments of type 'double'.
```

Question 1 (a) (ii)

```
% APPM3021 Lab 3, Question 1 (a) (II)

clc
clear global variable

syms f x;
f = @(x) 1 - 2/(x^2 - 2*x + 2)
x_0 = 0;
I_0 = [-1, 1];
tol = 0.0001;

% measurements
root_bisec = bisectionSearch(f, tol, I_0);
root_falsi = regulaFalsiSearch(f, tol, I_0);
fprime = matlabFunction( diff(f(x)) );
root_newton = NewtonMethodScaler(f, fprime, x_0, tol);
```

When question1a_II.m is run in the workspace, the following output is displayed to the command window:

```
f =

    @(x)1-2/(x^2-2*x+2)
```

```
Root = 0 found by bisection method within tolerance: 0.0001 in 2 iterations
Root = 0 found by Regula-Falsi method within tolerance: 0.0001 in 9 iterations
Root = 0 found by Newton method within tolerance: 0.0001 in 2 iterations
```

Question 1 (b)

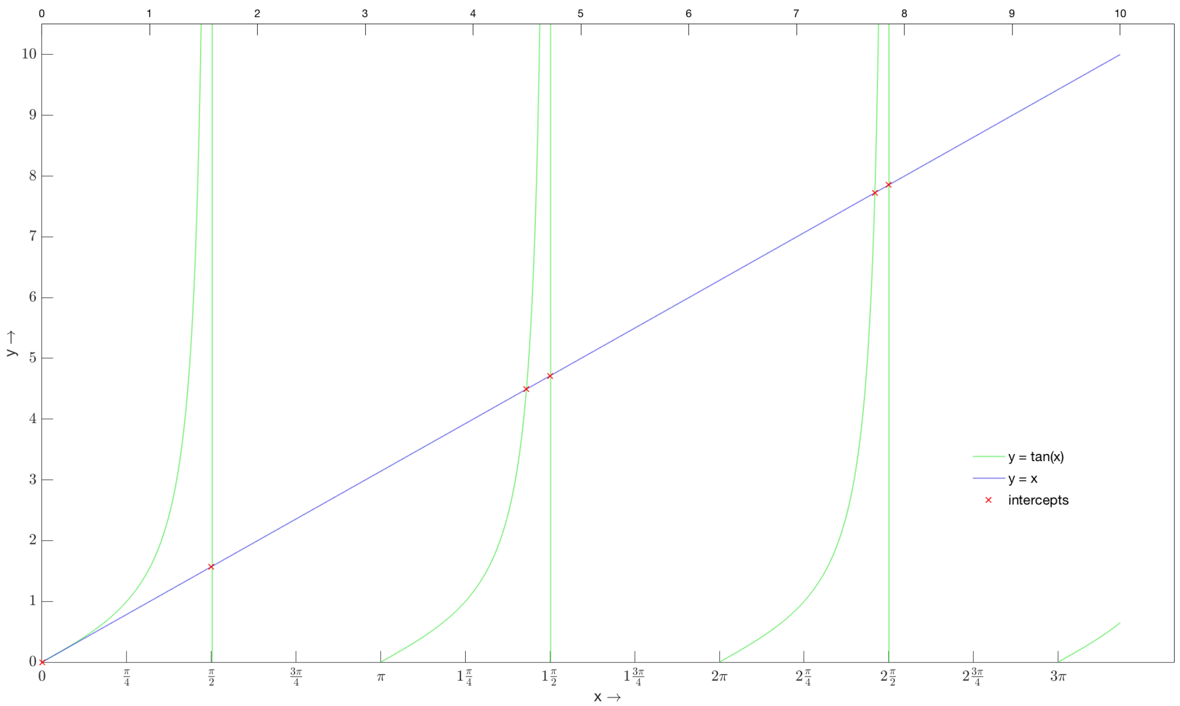
```
% APPM3021 Lab 3, Question 1 (b)

clc
clear all

syms x y;
f = @(x) tan(x) - x;
x_0 = [1.5, pi/2, 4.6, 3*pi/2, 4.71, 5*pi/2, 7.6, 7.85]; % guesses based on visual intercepts
tol = 0.0001;
root_newton = [];

%% calculations
fprime = matlabFunction( diff(f(x)) );
for i=1:length(x_0)
    root = NewtonMethodScaler(f, fprime, x_0(i), tol);
    if isempty(root) || isnan(root) || isinf(root)
    elseif (root > 10) || (root < 0)
    else
        root_newton(i) = root;
    end
end
root_newton= sort(unique(root_newton));

% iterations
iter_newton = length(root_newton);

%% Display setting and output setup
scr = get(groot,'ScreenSize');                              % screen resolution
fig1 =  figure('Position',...                               % draw figure
    [1 scr(4)*3/5 scr(3)*3.5/5 scr(4)*3/5]);
set(fig1,'numbertitle','off',...                            % Give figure useful title
    'Color','white');
fontName='Helvetica';
set(0,'defaultAxesFontName', fontName);                     % Make fonts pretty
set(0,'defaultTextFontName', fontName);
set(groot,'FixedWidthFontName', 'ElroNet Monospace')

% Draw plot to examine the function tan(x)-x=0
values = [0:0.01:10];
a=tan(values);
p2 = plot(values,a,...
        'Color',[0.18 0.9 0.18 .6],...
        'LineStyle','-',...
        'LineWidth',1);
hold on

r1 = refline(1,0);
set(r1,'Color',[0.18 0.18 0.9 .6],...
        'LineStyle','-',...
        'LineWidth',1);
hold on

for i=1:length(root_newton)
    root_dot(i) = plot(root_newton(:,i),root_newton(:,i),'rx');
hold on
end

% Axes and labels
ax1 = gca;;
ax1.XTick = [0:pi/4:3*pi];
ax1.XTickLabel = {'0','$\frac{\pi}{4}$','$\frac{\pi}{2}$','$\frac{3\pi}{4}$','$\pi$',...
                  '$1\frac{\pi}{4}$','$1\frac{\pi}{2}$','$1\frac{3\pi}{4}$','$2\pi$',...
                  '$2\frac{\pi}{4}$','$2\frac{\pi}{2}$','$2\frac{3\pi}{4}$','$3\pi$',...
                  '$3\frac{\pi}{4}$','$3\frac{\pi}{2}$','$3\frac{3\pi}{4}$','$4\pi$'};
box(ax1,'off');
set(ax1,'FontSize',14,...
    'YMinorTick','off',...
    'XMinorTick','off',...
    'TickLabelInterpreter','latex');
hold on
ylabel('y \rightarrow',...
    'FontName',fontName,...
    'FontSize',14);%,...
xlabel('x \rightarrow',...
    'FontName',fontName,...
    'FontSize',14);
% Legend
legend1 = legend({'y = tan(x)','y = x', 'intercepts'},...
     'Location','best',...
     'Position',[0.7    0.3    0.2    0.09],...
     'Box','off');
hold on

ax2 = axes('Position',get(ax1,'Position'),...
```

```
            'XAxisLocation','top',...
            'YAxisLocation','right',...
            'Color','none',...
            'XColor','k','YColor','k',...
             'Box','off');
offsetx = 0.5;
ax1.XLim = [0 10+offsetx];
ax1.YLim = [0 10+offsetx];
ax2.XLim=ax1.XLim;
ax2.YLim=ax1.YLim;
set(ax2,'YTick','')
```

When question1b.m is run in the workspace, the following output is displayed to the command window:

```
Root = 0 found by Newton method within tolerance: 0.0001 in 51 iterations
Root = 1.5708 found by Newton method within tolerance: 0.0001 in 2 iterations
Root = 4.4934 found by Newton method within tolerance: 0.0001 in 6 iterations
Root = 4.7124 found by Newton method within tolerance: 0.0001 in 2 iterations
Root = 4.4934 found by Newton method within tolerance: 0.0001 in 12 iterations
Root = 7.854 found by Newton method within tolerance: 0.0001 in 2 iterations
Root = 7.7252 found by Newton method within tolerance: 0.0001 in 12 iterations
Root = 7.7252 found by Newton method within tolerance: 0.0001 in 10 iterations
```

Figure 2. Figure



Question 2 (a)

```
% APPM3021 Lab 3, Question 2 (a)

clc
clear all

% system of equations
syms f g h u v w;
f(u,v,w) = u^2 +4*u*v - 2;
g(u,v,w) = u*v - u^2 + v^2;
h(u,v,w) = u^2 + w;
F = [f;g;h];
J = jacobian(F, [u, v, w]);

X_0 = [ 1 1 1 ];
tol = 0.0000001;

tic;
it_root_newton_sys = NewtonMethodSystem(F, J, X_0, tol);
t_newton_sys = toc;
disp(['Solution converged in ', num2str(t_newton_sys*1000), ' milli-seconds (including calculation of f'')'])
```

When question2a.m is run in the workspace, the following output is displayed to the command window:

```
roots =
    0.7590
    0.4691
   -0.5760
```

```
Roots found by Newton (system) method within tolerance: 1e-07 in 5 iterations
Solution converged in 353.3214 milli-seconds (including calculation of f')
```

Question 2 (b)

```
% APPM3021 Lab 3, Question 2 (b)

clc
clear all

% system of equations
syms f g x y;
f(x,y) = x^3 + y^3 -3 ;
g(x,y) = x^2 - y^2 -2;
F = [f;g];
J = jacobian(F, [x, y]);

X_0 = [ 1 1 ; -1 -1 ];
tol = 0.000001;

for i=1:length(X_0)
    root = NewtonMethodSystem(F, J, X_0(i,:), tol);
    if isempty(root(:)) || isnan(sum(root)) || isinf(sum(root(:)))
    else
        root_newton(:,i) = [root(1),root(2)]';
    end
end
```

When question2b.m is run in the workspace, the following output is displayed to the command window:

```
roots =
    1.4392
    0.2670

Roots found by Newton (system) method within tolerance: 1e-06 in 7 iterations
roots =
    1.4468
   -0.3053

Roots found by Newton (system) method within tolerance: 1e-06 in 13 iterations
```

Functions and Code

bisectionSearch.m :

```
function [ root ] = bisectionSearch( f, tol, I_0, keep_iterations )
% bisectionSearch returns the root of an equation f, within tolerance tol
% with initial bracket I_0 = [a_initial, b_initial]
% The method is based on continual bisection of the interval
% containing the root (by evaluating the sign of of the input
% equation over the two halves of the current interval)

if nargin<4
    keep_iterations = false;
end

% initial values
a = I_0(1);
b = I_0(2);
c = [ b-a ];
root_found = false;

if f(a)*f(b) > 0                                                % must have opposite signs
    error(['No root can be found within the interval [',...
        num2str(a),',', num2str(b),'] with the equation ',func2str(f)])
end

sign_places = abs(log10(tol))+1;
max_iterations = ceil(log2(2*(b-a)/tol))-1;

for i = 2:max_iterations+1

    if f(b)==f(a)
        error(['Interval is zero between [',...
        num2str(a),',', num2str(b),']'])
    end

    c(i) = (a+b)/2;

    % stopping criteria
    if f(c(i)) == 0                                             % root found!
        root_found = true;
    elseif c(i)==0
        error('Division by zero (cannot test stopping criteria)')
    end

    if (abs(c(i)-c(i-1)) / abs(c(i)) < tol) || root_found
        if keep_iterations
            root = round(c,sign_places);
        else
            root = round(c(end),sign_places);
        end
        disp(['Root = ',num2str(root(end)),...
            ' found by bisection method within tolerance: ',...
            num2str(tol), ' in ', num2str(i), ' iterations'])
        return
    end

    % prepare for next loop
    if f(a)*f(c(i)) < 0
        b = c(i);
    else                                                        % f(a)*f(c(i)) > 0, i.e. same signs
        a = c(i);
    end
end
disp('Operation failed')
root = [];
end
```

regulaFalsiSearch.m :

```
function [ root ] = regulaFalsiSearch(f, tol, I_0, keep_iterations)
% regulaFalsiSearch returns the root of an equation f, within tolerance tol
% with initial bracket I_0 = [a_initial, b_initial]
% regular falsi does a linear interpolation between two points,
% finding the x-intercept and using this intercept for the new interval

if nargin<4
```

```
    keep_iterations = false;
end


% initial values
a = I_0(1);
b = I_0(2);
c = [ b-a ];
root_found = false;

if f(a)*f(b) > 0                                                  % must have opposite signs
    error(['No root can be found within the interval [',...
        num2str(a),',', num2str(b),'] with the equation ',func2str(f)])
end

sign_places = abs(log10(tol))+1;
i=2;

while true
    if f(b)==f(a)
        error(['Interval is zero between [',...
        num2str(a),',', num2str(b),']'])
    end

    c(i) = ( a*f(b) - b*f(a) ) / ( f(b) - f(a) );

    % stopping criteria
    if round(f(c(i)),sign_places+1) == 0                                                    % root found!
        root_found = true;
    elseif c(i)==0
        error('Division by zero (cannot test stopping criteria)')
    end

    if abs(c(i)-c(i-1)) / abs(c(i)) < tol || root_found
        if keep_iterations
            root = round(c,sign_places);
        else
            root = round(c(end),sign_places);
        end
        disp(['Root = ',num2str(root(end)),...
            ' found by Regula-Falsi method within tolerance: ',...
            num2str(tol), ' in ', num2str(i), ' iterations'])
        return
    end

    % prepare for next loop
    if f(c(i)) > 0
        b = c(i);
    else                                                          % f(c(i)) > 0
        a = c(i);
    end
    i=i+1;
end
end
```

NewtonMethodScaler.m :

```
function [ root ] = NewtonMethodScaler(f, fprime, x_0, tol, keep_iterations)
% NewtonMethodScaler returns the root of an equation f, within tolerance tol
% using initial guess x_0, with the iterative approximation taken as the
% intersection of f and derivative f'

if nargin<5
    keep_iterations = false;
end

% initial values
x = x_0;
root_found = false;

sign_places = abs(log10(tol))+1;
i=2;
iteration_limit = 10^sign_places;

while true

%     if fprime(x(i-1))==0
%         error('Division by zero, fprime = 0')
%     else
        x(i) =  x(i-1) - ( f(x(i-1) ) / fprime(x(i-1)) );
%     end

    % stopping criteria
    if f(x(i)) == 0                                               % root found!
        root_found = true;
    elseif abs(x(i))==0
        error('Division by zero (cannot test stopping criteria)')
    end

    if (abs( x(i) - x(i-1) ) / abs( x(i) ) < tol) || root_found
        if keep_iterations
            root = round(x,sign_places);
        else
```

```
            root = round(x(end),sign_places);
        end
        disp(['Root = ',num2str(root(end)),...
            ' found by Newton method within tolerance: ',...
            num2str(tol), ' in ', num2str(i), ' iterations'])
        return
    end

    if i>iteration_limit
        disp(['Unable to find root (within ', num2str(iteration_limit),...
            ' iterations)'])
        root = [];
        return
    else
        i=i+1;
    end
end
end
```

NewtonMethodSystem.m :

```
function [ roots ] = NewtonMethodSystem(F, J, X_0, tol, keep_iterations)
% NewtonMethodSystem find the roots to a system of two (or more) equations
% using Newton's fixed-point iterative method

if nargin<5
    keep_iterations = false;
end

sign_places = abs(log10(tol));

% M = inv(J)*F;
M = J\F;                                             % inverse jacobian * system of equations (symbolic)
X(:,1) = X_0;                                        % initial guess
variables = num2cell(X_0);                           % seperate out individual inputs to function
% u = X_0(1);                                        % (individual variable)
% v = X_0(2);
i = 1;                                               % iterations

while true
%  X(:,i+1) = X(:,i) - M(u,v);                      % Newton Method
    X(:,i+1) = X(:,i) - M(variables{:});

   if (norm(double(M(variables{:})),inf) < tol)      % check error tolerance
       if keep_iterations
        roots = round(X,sign_places);                % assign output
       else
           roots = round(X(1:end,end),sign_places); % assign output
       end
       disp('roots = ')
       disp(roots(1:end,end))
       disp(['Roots found by Newton (system) method within tolerance: ',...
             num2str(tol), ' in ', num2str(i), ' iterations'])
       return
   else

   variables = num2cell(X(:,i+1));                    % update individual variables

   i = i + 1;                                         % update iterations
   end
end

end
```