Homework 2

Name: YOUR NAME HERE

Due Date:

Table of Contents


```
clear, clc
```

Problem #1 - Trajectory of a toy missile

a) Plot the equation as a function of θ_0

```
y0 = 1.3;
v0 = 70;
g = 3.721;
x = 290;
h = 3.4;

thetaDeg = [5,85];

a0 = @() y0;
a1 = @(a) tand(a);
a2 = @(a) -(g/2)./((v0.^2).*cosd(a).^2);

f = @(a) a2(a).*x.^2 + a1(a).*x + a0() - h;

figure()
fplot(f, thetaDeg)
title("HW2.1.a")
xlabel("\theta [Degrees]")
ylabel("f(\theta)")
grid on
hold on
```

b) How many solutions exist?

2

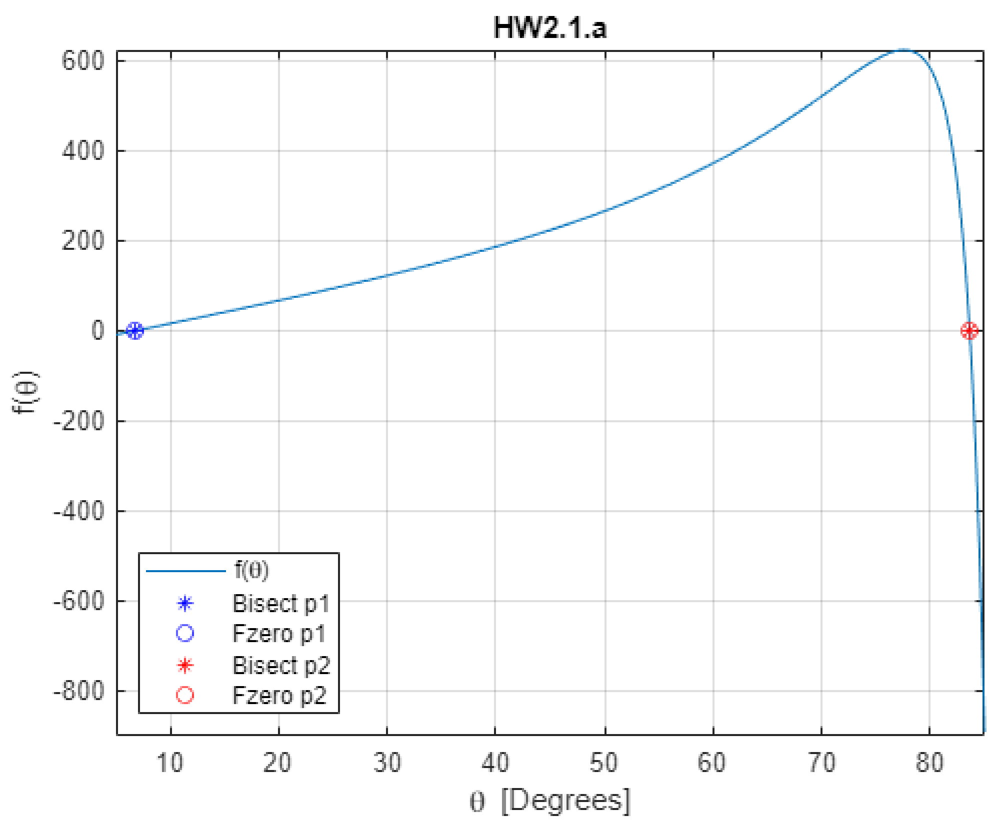
c) Table of angles that will allow the missile to hit the target

```
guess_bisect_1 = bisect(f, 0, 10);
guess_bisect_2 = bisect(f, 80, 90);
guess_fzero_1 = fzero(f, 5);
guess_fzero_2 = fzero(f, 80);
fprintf("Zero #    Bisect     Fzero\n" + ...
        "------ --------- ---------\n" + ...
        "   1   %9.6f %9.6f\n" + ...
        "   2   %9.6f %9.6f\n", guess_bisect_1, guess_fzero_1, guess_bisect_2, guess_fzero_2)
```

```
Zero #    Bisect     Fzero
------ --------- ---------
   1    6.781173  6.781172
   2   83.633652 83.633721
```

d) Plot the solutions from part c into the plot from part a

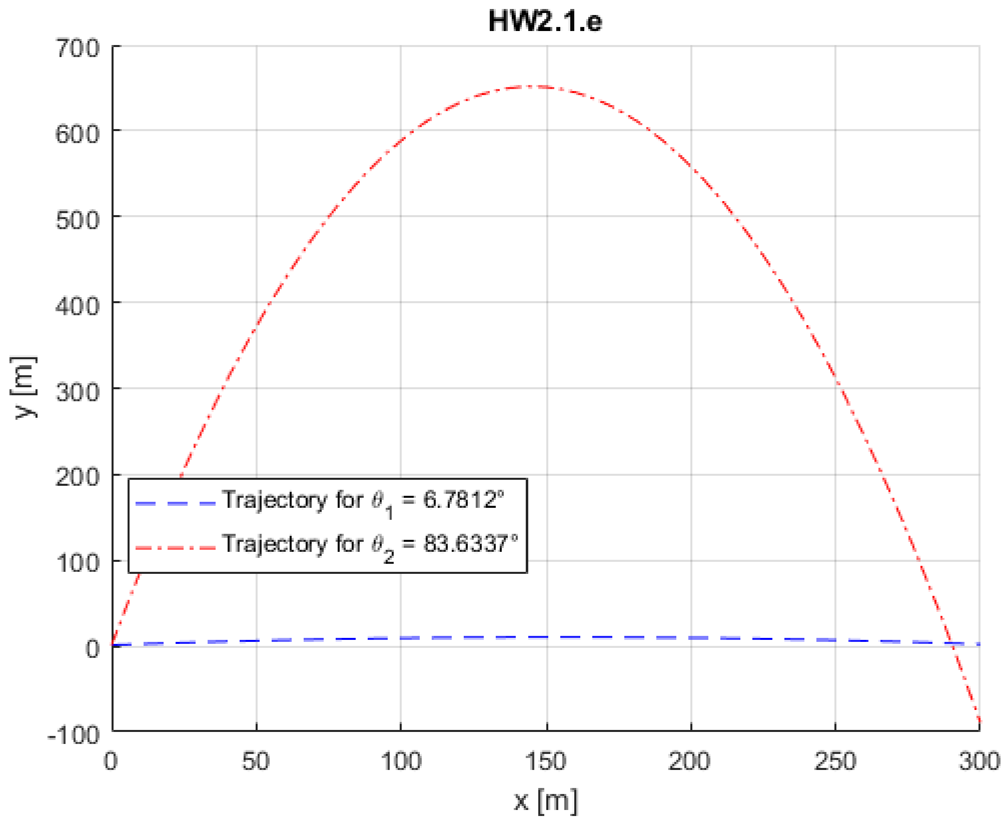
```
plot(guess_bisect_1, f(guess_bisect_1), "b*", guess_fzero_1, f(guess_fzero_1), "bo", ...
     guess_bisect_2, f(guess_bisect_2), "r*", guess_fzero_2, f(guess_fzero_2), "ro")
legend("f(\theta)", "Bisect p1", "Fzero p1", "Bisect p2", "Fzero p2","location", "southwest")
hold off
```



e) Plot the trajectory of the toy missile

```
ax1 = guess_fzero_1;
fx1 = @(x) a2(ax1).*x.^2 + a1(ax1).*x + a0();
ax2 = guess_fzero_2;
fx2 = @(x) a2(ax2).*x.^2 + a1(ax2).*x + a0();
```

```
figure()
hold on
fplot(fx1, [0,300], "b--")
fplot(fx2, [0,300], "r-.")
grid on
title("HW2.1.e")
xlabel("x [m]")
ylabel("y [m]")
str1 = ['Trajectory for \theta_1 = ' num2str(ax1) '°'];
str2 = ['Trajectory for \theta_2 = ' num2str(ax2) '°'];
legend(str1, str2, "location", "best")
```



Problem #2 - Bisect vs. False position

```
maxerr = .0002;
f = @(x) .074*exp(-.3*x).*sin(.1*x) + 8;
```

a) Plot the function

```
figure
hold on
fplot(f, [-35, -2], "b-.")
title("MAE284 HW2.2")
xlabel("X Position")
ylabel("Y Position")
```

b) Compute the root. Display the results in a table. Explain the difference

```
[xrb, ib] = bisect(f, -39,-28, maxerr);
[xrfp, ifp] = falsePos(f, -39, -28, maxerr);
fprintf("Using -39 and -28 as Bounds\n\n" + ...
        "        Method          Root Iteration\n" + ...
        "-------------- ------------- ---------\n" + ...
        "        Bisect %13.10f %9d\n" + ...
        "False Position %13.10f %9d\n", xrb, ib, xrfp, ifp)
```

```
Using -39 and -28 as Bounds

        Method          Root Iteration
-------------- ------------- ---------
        Bisect -31.3263435364        18
False Position -31.3258411426        85
```

Explain the difference in the number of iterations between the two methods

The bisect Function is a more effecient function for this plot, as the limitations on bisect that require the use of False Position are not present for this function.

c) Compute the root. Display the results in a table. Explain the difference

```
[xrb2, ib2] = bisect(f, -26,-6, maxerr);
[xrfp2, ifp2] = falsePos(f, -26, -6, maxerr);
fprintf("Using -26 and -6 as Bounds\n\n" + ...
        "        Method          Root Iteration\n" + ...
        "-------------- ------------- ---------\n" + ...
        "        Bisect %13.10f %9d\n" + ...
        "False Position %13.10f %9d\n", xrb2, ib2, xrfp2, ifp2)
```

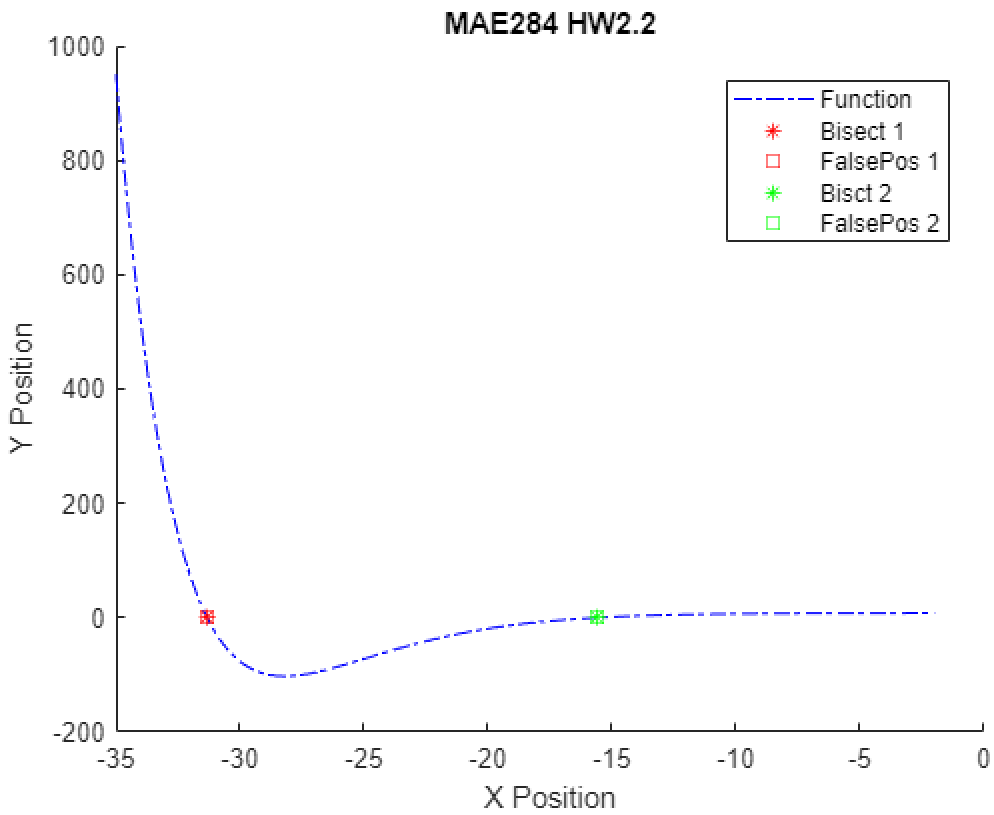
```
Using -26 and -6 as Bounds

        Method          Root Iteration
-------------- ------------- ---------
        Bisect -15.6106147766        20
False Position -15.6105358852        37
```

Explain the difference in the number of iterations between part b and part c

The section for part c did not contain a maximum with both sides being larger than the value of the root, and instead both sides approached the root from opposite signs, allowing the methods to converge faster

d) Plot all solutions from part b and c in the figure from part a

```
plot(xrb, f(xrb), "r*", xrfp, f(xrfp), "rs", xrb2, f(xrb2), "g*", xrfp2, f(xrfp2), "gs")
legend("Function", "Bisect 1", "FalsePos 1", "Bisct 2", "FalsePos 2", "Location", "best")
hold off
```



Problem #3 - Newton-Raphson

a) Display the vector as a column vector

```
format long
x = [.9];
g = @(x) x.^3 + 2*x +34;
dg = @(x) 3*x.^2 + 2;

for i = 1:9
    x(i+1) = x(i)- (g(x(i))/dg(x(i)));
end

disp(x')
```

```
   0.900000000000000
  -7.345823927765236
  -5.044915901549569
  -3.711359168885046
  -3.144825871980325
  -3.037728262027065
  -3.034124086846070
  -3.034120091462843
  -3.034120091457937
  -3.034120091457937
```

b) Calculate the exact real root using the roots function.

```
p = [1 0 2 34];
```

```
r = roots(p)
```

```
r = 3×1 complex
   1.517060045728969 + 2.984026398516197i
   1.517060045728969 - 2.984026398516197i
  -3.034120091457936 + 0.000000000000000i
```

```
r_real = r(r==real(r));
fprintf("The exact root of the function is %.10f", r_real)
```

```
The exact root of the function is -3.0341200915
```

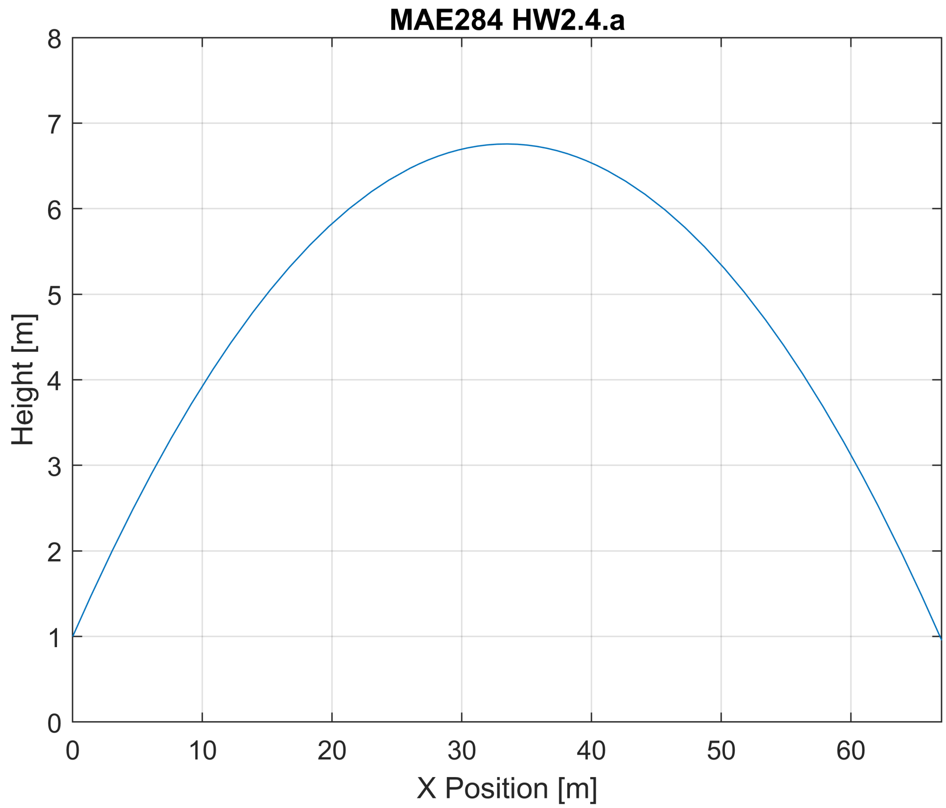
c) Why does the Newton-Raphson method initially overshoot the exact value and then increase (or decrease) to converge to the exact value?

Because the Netwon method uses the slope of the problem at the point, therefore if the slope not similar to the slope at the root, then the initial estimate will be far off, and it will slowly work its way closer and closer to the actual root.

Problem #4 - Baseball trajectory

a) Plot the function

```
theta = 19;     %deg
v0 = 73;        %mph
d = 67;         %m
g = 9.81;       %m/s²
y0 = 1;         %m
v0 = v0 * 5280/3600 * 1/3.28;

f = @(x) x*(tan((pi/180)*theta)) - (g/(2*v0^2*cos((pi/180)*theta)^2))*x.^2 + y0;
fplot(f, [0,67])
ylim([0,8])
grid on
title("MAE284 HW2.4.a")
xlabel("X Position [m]")
ylabel("Height [m]")
```



b) Will the catcher catch the ball?

```
if f(d) > 2.6
    canCatch = "CANNOT" ;
else
    canCatch = "CAN" ;
end
fprintf("The elevation at x = %2dm is %6.4fm. The catcher %s Catch the Ball", d, f(d), canCatch
```

```
The elevation at x = 67m is 0.9553m. The catcher CAN Catch the Ball
```

c) Compute the maximum height reached

```
g = @(x) -f(x);
```

i) Golden section search

```
gminx = goldmin(g, 8, 53, .1);
fprintf("The Maximum height using goldmin is %.8f", gminx)
```

```
The Maximum height using goldmin is 27.72922574
```

ii) Parabolic interpolation

```
paraminx = paramin(g, 4, 42, 18, 3);
fprintf("The Maximum height using paramin is %.8f", paraminx)
```

```
The Maximum height using paramin is 33.43519435
```

iii) MATLAB's fminbnd

```
options = optimset('Display','iter');
fminx = fminbnd(g, 0, 67, options);
```

```
 Func-count     x          f(x)         Procedure
    1        25.5917     -6.43955        initial
    2        41.4083       -6.429        golden
    3        15.8166     -5.15794        golden
    4        33.4352     -6.75633        parabolic
    5        33.4352     -6.75633        parabolic
    6        33.4352     -6.75633        parabolic

Optimization terminated:
 the current x satisfies the termination criteria using OPTIONS.TolX of 1.000000e-04
```

```
fprintf("The Maximum height using fminbnd is %.8f", fminx)
```

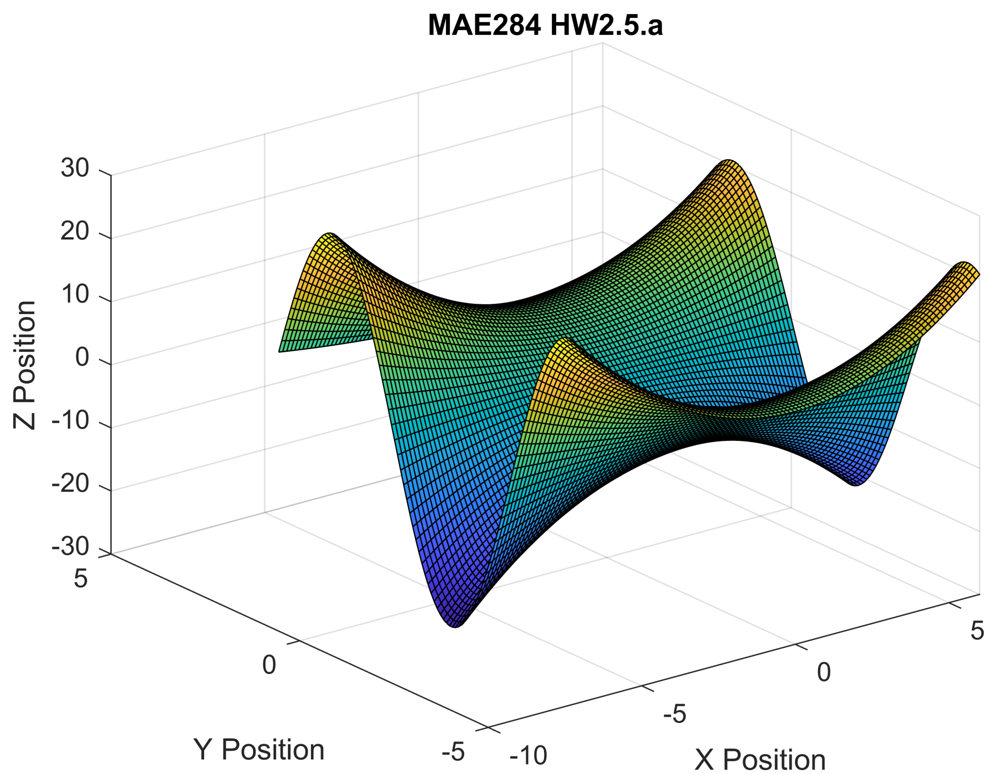
```
The Maximum height using fminbnd is 33.43519435
```

Problem #5 - 2D Optimization

```
x = linspace(-7,6);
y = linspace(-5,3);

c = @(x,y) (7.3 - .02*x - .38*y + .4*x.^2 - .08*y.^2 - .004*x.*y).*sin(y);

[X,Y] = meshgrid(x,y);
Z = c(X,Y);
```

a) 3D plot of function

```
surf(X,Y,Z)
ylabel("Y Position")
xlabel("X Position")
zlabel("Z Position")
title("MAE284 HW2.5.a")
```



b) 1st initial guess, find maximum, display in fprintf

```
G = @(P) -c(P(1), P(2));
options = optimset('MaxIter', 10^6, 'MaxFunEvals', 10^6);
[Q, fmin] = fminsearch(G,[2,-2], options);
```

```
Exiting: Maximum number of function evaluations has been exceeded
         - increase MaxFunEvals option.
         Current function value: -Inf
```

c) plot maximum on plot from part a)

```

```

d) 2nd initial guess, find max and add to plot from part a)

```

```

e) What changes between the two different initial guesses and why?

ANSWER HERE

Helper functions

Bisect function

```
function[xr, i, fx, ea] = bisect(f, xL, xu, et)
%bisect: bisection root locator that finds roots between XL and xu
% Inputs:
% f - function to be evaluated
% xL, xu - lower and upper bounds, respectively
% et - maximum allowable error (default 0.0001%)
% Outputs:
% xr - estimated root location
% fx - input function value at the estimated root location
% ea - magnitude of approximate relative error (%)

% Created by: Isheeta Ranade
% 22 January, 2018

if nargin<3, error('At least 3 input arguments required'), end % Check to ensure 3 inputs are i
test = f(xL)*f(xu); % Create a variable test which is used to see if the sign changes between x
if test > 0, error('No sign change f(xL) and f(xu)'), end
if nargin < 4 | isempty(et), et=0.0001; end % Ensure et is specified

xr = xL;
ea = 100;
for i = 1:50
    xrold = xr;
    xr = (xL + xu)/2;
   sgnchng = f(xL)*f(xr);
   if sgnchng < 0
       xu = xr;
       ea = abs((xr-xrold)/xr)*100;
   elseif sgnchng > 0
       xL = xr;
       ea = abs((xr-xrold)/xr)*100;
   else
       ea = 0;
   end
       if ea < et
           break
       end
end % end of for loop
 fx = f(xr);
end
```

False position function

```
function[xr, i, fx, ea] = falsePos(f, xL, xu, et)
%bisect: bisection root locator that finds roots between XL and xu
% Inputs:
% f - function to be evaluated
```

```
% xL, xu - lower and upper bounds, respectively
% et - maximum allowable error (default 0.0001%)
% Outputs:
% xr - estimated root location
% fx - input function value at the estimated root location
% ea - magnitude of approximate relative error (%)

% Created by: Isheeta Ranade
% 22 January, 2018

% Modified by: Jackson Lee
% 7 February, 2023

if nargin<3, error('At least 3 input arguments required'), end % Check to ensure 3 inputs are i
test = f(xL)*f(xu); % Create a variable test which is used to see if the sign changes between x
if test > 0, error('No sign change f(xL) and f(xu)'), end
if nargin < 4 | isempty(et), et=0.0001; end % Ensure et is specified

xr = xL;
ea = 100;
for i = 1:500
    xrold = xr;
    xr = xu - ((f(xu)*(xL-xu))/(f(xL)-f(xu)));  %Changes for False Position
   sgnchng = f(xL)*f(xr);
   if sgnchng < 0
       xu = xr;
       ea = abs((xr-xrold)/xr)*100;
   elseif sgnchng > 0
       xL = xr;
       ea = abs((xr-xrold)/xr)*100;
   else
       ea = 0;
   end
       if ea < et
           break
       end
end % end of for loop
 fx = f(xr);
end
```

Golden-section search function

```
function [x,fx,ea,iter]=goldmin(f,x1,xu,es,maxit,varargin) % goldmin: minimization golden secti
% [x,fx,ea,iter]=goldmin(f,x1,xu,es,maxit,p1,p2,...):
%     uses golden section search to find the minimum of f
% input:
% f = name of function
%   x1, xu = lower and upper guesses
%   es = desired relative error (default = 0.0001%)
%   maxit = maximum allowable iterations (default = 50)
```

```
%   p1, p2,... = additional parameters used by f
% output:
%   x = location of minimum
%   fx = minimum function value
%   ea = approximate relative error (%)
%   iter = number of iterations

if nargin<3, error('at least 3 input arguments required'), end
if nargin<4|isempty(es), es=0.0001; end
if nargin<5|isempty(maxit), maxit=50; end
phi=(1+sqrt(5))/2; iter = 0;
d = (phi-1)*(xu - x1);
x1 = x1 + d; x2 = xu - d;
f1 = f(x1,varargin{:}); f2 = f(x2, varargin{:});
while(1)
    xint = xu - x1;
    if f1 < f2
        xopt = x1; x1 = x2; x2 = x1; f2 = f1;
        x1 = x1 + (phi-1)*(xu-x1); f1 = f(x1, varargin{:});
    else
        xopt = x2; xu = x1; x1 = x2; f1 = f2;
        x2 = xu - (phi-1)*(xu-x1); f2 = f(x2, varargin{:});
    end
    iter = iter +1;
    if xopt~=0, ea = (2 - phi) * abs(xint / xopt) * 100; end
    if ea <= es | iter >= maxit, break, end
end
x=xopt; fx=f(xopt, varargin{:});
end
```

Parabolic interpolation function

```
function [xmin,fmin,ea,iter] = paramin(f,xL,xu,xm,es,maxIt,varargin)
% paramin: minimization using parabolic interpolation
%   [xmin,fmin,ea,iter] = paramin(f,xL,xu,es,maxIt,p1,p2,...)
% Inputs:
%   f = function to be evaluated
%   xL, xu = lower and upper bounds, respectively
%   xm = first guess at minimum (default (xL+xu)/2)
%   es = allowable relative error (default = 0.0001%)
%   maxIt = maximum allowable iterations (default = 50)
% Outputs:
%   xmin = location of minimum
%   fmin = minimum function value
%   ea   = magnitude of approximate relative error (%)
%   iter = number of iterations required to find the minimum

% Created by: Isheeta Ranade
% Edited by: Kim Xu
% Today's date: 9/14/2021
```

```matlab
if nargin<3, error('At least 3 input arguments required'), end
if nargin<4||isempty(xm), xm=(xL+xu)/2; end
if nargin<5||isempty(es), es=0.0001; end
if nargin<6||isempty(maxIt), maxIt=50; end


iter = 0;
x4 = xu;
while(1)
    xold = x4;
    iter = iter + 1;
    x4 = xm - 0.5*((xm-xL)^2*(f(xm)-f(xu))-(xm-xu)^2*(f(xm)-f(xL)))/...
        ((xm-xL)*(f(xm)-f(xu))-(xm-xu)*(f(xm)-f(xL)));
    if x4 <= xu
        xL = xm;
        xm = x4;
    else
        xu = xm;
        xm = x4;
    end

    if x4 ~= 0
        ea = abs((x4 - xold)/x4)*100;
    end
    if ea <= es || iter >= maxIt, break, end

end
xmin = x4;
fmin = f(x4);
end
```