Table of Contents


```
%
 ==========================================================================
% AUTHOR ..... [Huang Lishan, Yang, Wenshuo, Hoffman, Nicholas,
 Tamkee, Michael]
% UPDATED .... [Jan 18]
% Provide the results for each of the 3 tasks
```

*newtons Method *

newtonsMethod.m

```
%
 ==========================================================================
% AUTHOR ..... [Huang Lishan, Yang, Wenshuo, Hoffman, Nicholas,
 Tamkee, Michael]
% UPDATED .... [Jan 15]
% evaluate the roots of a given function by Newton's method
% INPUT
%   f       ....   The function that you want to solve.
%   df      ....   The derivative of the function that you want to
 solve.
%   x0      ....   The initial guess
%   tol     ....   The tolerance specifying the accuracy wanted in the
 solution
%   maxIter ....   The maximum number of iterations allowed
%
 ==========================================================================

function N = newtonsMethod(f, df, x0, tol, maxIter)

    % Initialize the first two values
    x=x0;
    xprev=0;

    % Create a vector that stores 0 only with the length of maxIter
    N=zeros(1,maxIter);

    % define k as the element in the vector to place x into
    k = 0;

    % loop the function while abs(x-xprev) is greater than the
 tolerance,
```

```
    % and the number of iterations is smaller than the maximum defined
    while abs(x-xprev) > tol & k <maxIter

        % add one to the value of k
        k = k + 1;

        % insert value of x(n) into the kth element of the vector
        N(k)=x;

        % definition of Newton's method
        xprev = x;
        x = x - f(x)/df(x);

    end

    % delete the tail zeros in the vector
    N=N(1:k);

    format long

end
```

*secant Method *

secantMethod.m

```
%
 ==========================================================================
% AUTHOR ..... [Huang Lishan, Yang, Wenshuo, Hoffman, Nicholas,
 Tamkee, Michael]
% UPDATED .... [Jan 15]
% evaluate the roots of a given function by the secant method
% INPUT
%   f       ....   The function that you want to solve.
%   x0      ....   The first initial guess.
%   x1      ....   The second initial guess.
%   tol     ....   The tolerance specifying the accuracy wanted in the
 solution
%   maxIter ....   The maximum number of iterations allowed
%
 ==========================================================================

function S = secantMethod(f, x0, x1, tol, maxIter)

    % Initialize the first two values
    a=x0;
    b=x1;

    % Create a vector that stores 0 only with the length of maxIter
    S = zeros(1,maxIter);
```

```
    % Store the first initial guess into the zeros vector
    S(1) = a;

    % Begin k at 1 as the first element of the vector is already
 filled
    k=1;

    % loop the function as long as abs(b-a) is greater than the
 tolerance,
    % and the number of iterations is smaller than the maximum defined
    while abs(b-a) > tol & k < maxIter

        % add one to the value of k
        k = k + 1;

        % add the new value of x into the kth element of the vector
        S(k) = b;

        % definition of secantMethod
        c = a;
        a = b;
        b = a-(f(a)*(a-c)/(f(a)-f(c)));

    end

    % delete the tail zeros in the vector
    S=S(1:k);

end
```

task 1

```
clf;
close all;
clear all;

disp('*TASK 1*')

format long

% Define the function
f = @(x) (x + 1).*(x - 1/2);
df = @(x) 2*x + 1/2;
x0 = -1.2;
x1 = -0.9;
x2 = 0.4;
x3 = 0.6;
tol = 1e-20;
maxIter = 20;
```

```
% Find roots using the Newton's method
task1_newton_root1 = newtonsMethod(f, df, x0, tol, maxIter)
task1_newton_root2 = newtonsMethod(f, df, x2, tol, maxIter)

% Find roots using the secant method
task1_secant_root1 = secantMethod(f, x0, x1, tol, maxIter)
task1_secant_root2 = secantMethod(f, x2, x3, tol, maxIter)
```

```
*TASK 1*

task1_newton_root1 =

  Columns 1 through 3

  -1.200000000000000  -1.021052631578947  -1.000287407939644

  Columns 4 through 6

  -1.000000055047788  -1.000000000000002  -1.000000000000000


task1_newton_root2 =

  Columns 1 through 3

   0.400000000000000   0.507692307692308   0.500039047247169

  Columns 4 through 5

   0.500000001016405   0.500000000000000


task1_secant_root1 =

  Columns 1 through 3

  -1.200000000000000  -0.900000000000000  -0.987500000000000

  Columns 4 through 6

  -1.000900900900901  -0.999992433986532  -0.999999995458552

  Columns 7 through 8

  -1.000000000000023  -1.000000000000000


task1_secant_root2 =

  Columns 1 through 3

   0.400000000000000   0.600000000000000   0.493333333333333
```

```
  Columns 4 through 6

   0.499581589958159   0.500001868425478   0.499999999478677

  Columns 7 through 8

   0.499999999999999   0.500000000000000
```

task 2

```
clf;
close all;
clear all;

disp('*TASK 2*')

format long

% Define the function
f = @(y) y^3-2*y-5;
df = @(y) 3*y^2 - 2;
y0 = 2;
y1 = 2.1;
tol = -1;
maxIter = 7;

% Finding roots using different methods

    % Find a root using Newton's method
    task2_newton = newtonsMethod(f, df, y0, tol, maxIter)

    % Find a root using the secant method
    task2_secant = secantMethod(f,y0,y1, tol, maxIter)

    % Find roots using the roots function
    p = [1 0 -2 -5];
    task2_roots=roots(p)

% Find the roots when the intial guess is 1i

y0 = 1i;

    % For Newton's method - increase MaxIter -> 10 to allow
 convergence to
    % the roots method
    task2_newton_1i = newtonsMethod(f, df, y0, tol, 10)
```

```
*TASK 2*

task2_newton =

  Columns 1 through 3
```

```
   2.000000000000000   2.100000000000000   2.094568121104185

  Columns 4 through 6

   2.094551481698199   2.094551481542327   2.094551481542327

  Column 7

   2.094551481542327


task2_secant =

  Columns 1 through 3

   2.000000000000000   2.100000000000000   2.094250706880302

  Columns 4 through 6

   2.094550560860480   2.094551481698244   2.094551481542327

  Column 7

   2.094551481542327


task2_roots =

  2.094551481542328 + 0.000000000000000i
 -1.047275740771163 + 1.135939889088928i
 -1.047275740771163 - 1.135939889088928i


task2_newton_1i =

  Column 1

  0.000000000000000 + 1.000000000000000i

  Column 2

 -1.000000000000000 + 0.400000000000000i

  Column 3

 -0.562748739718758 + 1.771928893605731i

  Column 4

 -0.778372261504509 + 1.186040259620651i

  Column 5
```

```
 -1.042308494125078 + 1.090009717606734i

  Column 6

 -1.046911787596000 + 1.137242813136446i

  Column 7

 -1.047274787446181 + 1.135940470414528i

  Column 8

 -1.047275740770723 + 1.135939889088307i

  Column 9

 -1.047275740771163 + 1.135939889088928i

  Column 10

 -1.047275740771163 + 1.135939889088928i
```

task 3

```
clf;
close all;
clear all;

disp('*TASK 3*')

format long

% Define the variables
M= 174.796 * pi / 180;
e = 0.205630;
K = @(E) E - e * sin(E) - M;
dK =@(E) 1- e * cos(E);
ddK =@(E) e * sin(E);
E0=M;
E1= M + e*sin(M);
tol = 1e-20;
maxIter = 20;

% Perform Newton's Method

    % Define the function
    x_newton = newtonsMethod(K, dK, E0, tol, maxIter);

    % take the last value of the output that is closest to the
 convergence
    task3_newtonsMethod=x_newton(length(x_newton))
```

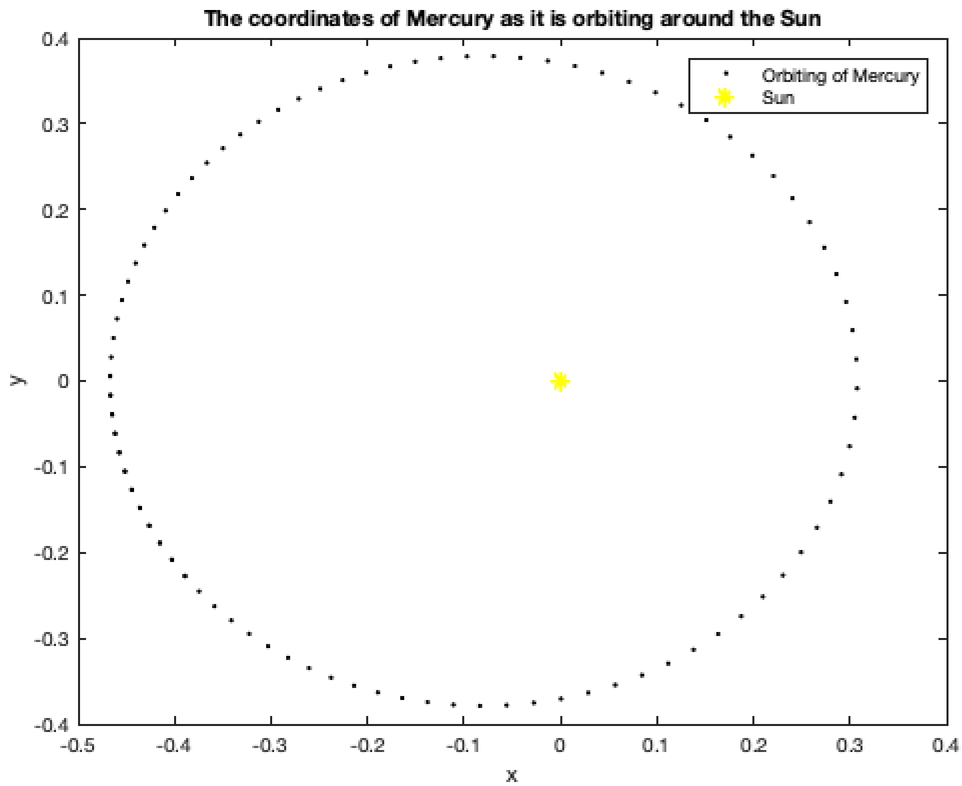
```
% Perform Halley's method

    % Define additional variables
    x=E0;
    xprev=0;

    % Set up Halley's method
        k = 0;
        N=zeros(1,maxIter);
            while abs(x - xprev) > tol & k < maxIter
                k = k + 1;
                N(k)=x;
                %definition of Halley's method
                xprev = x;
                x = x - K(x)/(dK(x)-0.5 * K(x) * ddK(x)/dK(x));
            end

        %delete the tail zeros value
        x_Halley=N(1:k);

    % End of Halley's method function

    %take the last value of the output that closest to the covergence
    task3_halleysMethod=x_Halley(length(x_Halley))

% Perform the Secant Method

    % Define the Secant method function
    x_secant = secantMethod(K, E0, E1, tol, maxIter);

    %take the last value of the output that closest to the covergence
    task3_secantMethod=x_secant(length(x_secant))

% plot Mercury's orbit

    disp('*Plotting Mercurys orbit around the Sun using Newtons
 method*')

    % initialize the input value
    T = 87.9691;
    t0 = 0;
    t=linspace(1, 100 ,100 );
    M0 =174.796*pi/180;
    M= 2*pi.*(t-t0)./T+M0;
    E_value = zeros(1,length(M));

    %use for loop to compute E of corresponding value of M and store
 it into
    %the zeros vector
        for t=1:length(M)
            K = @(E) E - e * sin(E) - M(t);
            dK =@(E) 1- e * cos(E);
            a=newtonsMethod(K, dK, M(t), tol, maxIter);
```

```
            E_value(t)=a(length(a));
        end

    % define a and compute b
    a = 0.387098;
    b=sqrt(a^2-(a*e)^2);

    % define functions of x and y
    x = a*(cos(E_value) - e);
    y = b*sin(E_value);

    % plot the orbit of Merury around the Sun
    plot(x,y,'k.',0,0,'y*')
    title('The coordinates of Mercury as it is orbiting around the
 Sun')
    legend('Orbiting of Mercury','Sun')
    xlabel('x')
    ylabel('y')
```

TASK 3

```
task3_newtonsMethod =

   3.066244834970549


task3_halleysMethod =

   3.066244834970549


task3_secantMethod =

   3.066244834970549
```

Plotting Mercurys orbit around the Sun using Newtons method



Published with MATLAB® R2017b