Q2

9 / 9

Write a MATLAB program `secant.m` for the secant method. Suppose we want to find the real root of $f(x) = x^3 - 2x - 5$. Plot the graph of $y = f(x)$ on an appropriate interval by Matlab (check how to use Matlab build-in function `plot`). Use your `secant.m` to compute the root. Also use the bisection method, the Newton method to find the root. For the bisection method, use $[2, 3]$ as the initial interval, for the Newton method, use $x_0 = 2$ as the initial point, for the secant method, use $x_0 = 2$ and $x_1 = 3$ as the two initial points. Take tolerances `xtol=1.e-12` and `ftol=1.e-12` for Newton's method, and the secant method, and take `delta=1.e-12` for the bisection method. Set a big number for the maximum number of iterations of the secant method and Newton's method such that the iteration stops only when `xtol=1.e-12` or `ftol=1.e-12` is satisfied. Print out the graph of $y = f(x)$ and the commands you used to plot the graph, your program `secant.m`, and other M-files related to $f(x)$. Also print out the results of each iteration step. You can use M-files `newton.m` and `bisection.m` on the course web site.

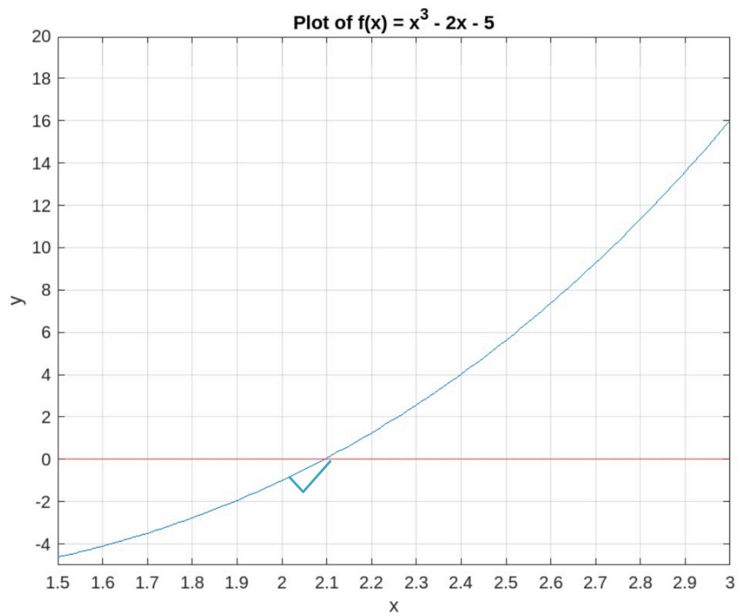
Secant.m:

```
function r = secant(f,x0,x1,xtol,ftol,nmax,display)
% Secant method for solving f(x)=0.
% r = secant(f,x0,x1,xtol,ftol,n_max,display)
% input:  f is the handle of the function f(x).
%         x0 and x1 are the initial points
%         xtol and ftol are termination tolerances
%         nmax is the maximum number of iterations
%         display = 1 if step-by-step display is desired,
%                 = 0 otherwise
% output: root is the computed root of f(x)=0
n = 0;
f0 = f(x0);
f1 = f(x1);
if display
   disp('   n                x                          f(x)');
   disp('------------------------------------------------------');
   disp(sprintf('%4d %23.15e %23.15e', n, x1, f1));
end
for n = 1:nmax
   d = ((x1-x0)/(f1-f0))*f1;
   x0 = x1;
   f0 = f1;
   x1 = x1 - d;
   f1 = f(x1);
   if display
       disp(sprintf('%4d %23.15e %23.15e', n, x1, f1));
   end
   if abs(d) <= xtol | abs(f1) <= ftol
       r = x1;
       return
   end
end
r = x1;
```

A4Q2.m:

```
% Assignment 4 - Question 2
f = @(x) x.^3 - 2*x - 5;
% Plot the graph first of f(x) = x^3 - 2x - 5 on the interval [1.5,3]
% Make a vector x for [1.5,3] of points incremented by 0.01
% Make a vector y of f(x) for each of the points in x
x = 1.5:1/100:3;
y = f(x);
disp("Plotting the following points for f(x)")
for n = 1:length(x)
   fprintf("(%1.2f,%23.15e)", x(n), y(n));
   disp(" ");
end
plot(x,y);
axis on;
ax = gca; % Get current axes
ax.XTick = 1.5:1/10:3; % Set x-axis ticks at intervals of 1/10
ax.YTick = -4:2:20; % Set y-axis ticks at intervals of 2
yline(0, 'Color', [1, 0, 0], 'LineWidth', 0.5); % Emphasize line of y=0
xlabel('x'); % Label for x-axis
ylabel('y'); % Label for y-axis
title('Plot of f(x) = x^3 - 2x - 5'); % Title of the plot
grid on; % Adds a grid for better visualization
% Compute root using secant method
disp(" ");
disp ("Secant method: ");
r = secant(f, 2, 3, 1.e-12, 1.e-12, 100, 1);
disp(" ");
disp("Final computed result for root using secant method:");
fprintf("r = %23.15e", r);
disp(" ");
% Compute root using newton's method
disp(" ");
disp ("Newton's method: ");
fd = @(x) 3*x.^2 - 2;
r = newton(f, fd, 2, 1.e-12, 1.e-12, 100, 1);
disp(" ");
disp("Final computed result for root using newton's method:");
fprintf("r = %23.15e", r);
disp(" ");
% Compute root using bisection method
disp(" ");
disp ("Bisection method: ");
r = bisection(f, 2, 3, 1.e-12, 1);
disp(" ");
disp("Final computed result for root using bisection method:");
fprintf("r = %23.15e", r);
```

Output of running A4Q2.m (plot):



Output of running A4Q2.m (command window):

```
>> A4Q2
Plotting the following points for f(x)
(1.50, -4.625000000000000e+00)
(1.51, -4.577049000000001e+00)
(1.52, -4.528192000000000e+00)
(1.53, -4.478422999999999e+00)
(1.54, -4.427735999999999e+00)
(1.55, -4.376125000000000e+00)
(1.56, -4.323584000000000e+00)
(1.57, -4.270106999999999e+00)
(1.58, -4.215688000000000e+00)
(1.59, -4.160321000000000e+00)
(1.60, -4.103999999999999e+00)
(1.61, -4.046719000000000e+00)
(1.62, -3.988472000000000e+00)
(1.63, -3.929253000000001e+00)
(1.64, -3.869056000000000e+00)
(1.65, -3.807875000000000e+00)
(1.66, -3.745704000000000e+00)
(1.67, -3.682537000000000e+00)
(1.68, -3.618368000000001e+00)
(1.69, -3.553191000000000e+00)
(1.70, -3.487000000000001e+00)
(1.71, -3.419789000000000e+00)
(1.72, -3.351552000000000e+00)
(1.73, -3.282283000000001e+00)
(1.74, -3.211976000000000e+00)
(1.75, -3.140625000000000e+00)
(1.76, -3.068223999999999e+00)
(1.77, -2.994767000000000e+00)
(1.78, -2.920248000000000e+00)
(1.79, -2.844661000000000e+00)
(1.80, -2.767999999999999e+00)
(1.81, -2.690259000000000e+00)
(1.82, -2.611431999999999e+00)
(1.83, -2.531512999999999e+00)
(1.84, -2.450496000000000e+00)
(1.85, -2.368374999999999e+00)
(1.86, -2.285144000000001e+00)
```

```
(1.87, -2.200796999999999e+00)
(1.88, -2.115328000000001e+00)
(1.89, -2.028730999999999e+00)
(1.90, -1.941000000000001e+00)
(1.91, -1.852128999999999e+00)
(1.92, -1.762112000000001e+00)
(1.93, -1.670943000000001e+00)
(1.94, -1.578616000000000e+00)
(1.95, -1.485125000000000e+00)
(1.96, -1.390464000000001e+00)
(1.97, -1.294627000000001e+00)
(1.98, -1.197608000000000e+00)
(1.99, -1.099401000000000e+00)
(2.00, -1.000000000000000e+00)
(2.01, -8.993990000000025e-01)
(2.02, -7.975919999999990e-01)
(2.03, -6.945729999999966e-01)
(2.04, -5.903359999999989e-01)
(2.05, -4.848750000000024e-01)
(2.06, -3.781840000000001e-01)
(2.07, -2.702569999999973e-01)
(2.08, -1.610879999999995e-01)
(2.09, -5.067100000000124e-02)
(2.10,  6.100000000000083e-02)
(2.11,  1.739309999999987e-01)
(2.12,  2.881280000000022e-01)
(2.13,  4.035969999999995e-01)
(2.14,  5.203440000000024e-01)
(2.15,  6.383749999999990e-01)
(2.16,  7.576960000000028e-01)
(2.17,  8.783129999999986e-01)
(2.18,  1.000232000000001e+00)
(2.19,  1.123459000000000e+00)
(2.20,  1.248000000000003e+00)
(2.21,  1.373861000000000e+00)
(2.22,  1.501047999999997e+00)
(2.23,  1.629566999999999e+00)
(2.24,  1.759424000000003e+00)
(2.25,  1.890625000000000e+00)
(2.26,  2.023175999999996e+00)
(2.27,  2.157082999999999e+00)
(2.28,  2.292352000000003e+00)
(2.29,  2.428989000000000e+00)
(2.30,  2.566999999999998e+00)
(2.31,  2.706391000000001e+00)
(2.32,  2.847167999999997e+00)
(2.33,  2.989337000000001e+00)
(2.34,  3.132903999999998e+00)
(2.35,  3.277875000000002e+00)
(2.36,  3.424256000000000e+00)
(2.37,  3.572053000000002e+00)
(2.38,  3.721271999999999e+00)
(2.39,  3.871919000000002e+00)
(2.40,  4.023999999999997e+00)
(2.41,  4.177521000000002e+00)
(2.42,  4.332487999999998e+00)
(2.43,  4.488906999999996e+00)
(2.44,  4.646784000000000e+00)
(2.45,  4.806125000000003e+00)
(2.46,  4.966935999999999e+00)
(2.47,  5.129222999999996e+00)
(2.48,  5.292991999999998e+00)
(2.49,  5.458249000000004e+00)
(2.50,  5.625000000000000e+00)
(2.51,  5.793250999999996e+00)
(2.52,  5.963008000000002e+00)
(2.53,  6.134276999999997e+00)
(2.54,  6.307064000000002e+00)
(2.55,  6.481374999999998e+00)
(2.56,  6.657216000000002e+00)
(2.57,  6.834592999999995e+00)
(2.58,  7.013512000000002e+00)
(2.59,  7.193978999999999e+00)
(2.60,  7.376000000000001e+00)
(2.61,  7.559580999999998e+00)
(2.62,  7.744728000000004e+00)
(2.63,  7.931446999999997e+00)
(2.64,  8.119744000000001e+00)
(2.65,  8.309624999999997e+00)
(2.66,  8.501096000000004e+00)
(2.67,  8.694163000000000e+00)
(2.68,  8.888832000000004e+00)
(2.69,  9.085108999999999e+00)
(2.70,  9.283000000000003e+00)
(2.71,  9.482511000000001e+00)
(2.72,  9.683647999999996e+00)
(2.73,  9.886416999999998e+00)
(2.74,  1.009082400000000e+01)
(2.75,  1.029687500000000e+01)
(2.76,  1.050457600000000e+01)
(2.77,  1.071393300000000e+01)
(2.78,  1.092495200000000e+01)
(2.79,  1.113763900000000e+01)
```

```
(2.80,  1.135200000000000e+01)
(2.81,  1.156804100000000e+01)
(2.82,  1.178576800000000e+01)
(2.83,  1.200518700000000e+01)
(2.84,  1.222630400000000e+01)
(2.85,  1.244912500000000e+01)
(2.86,  1.267365600000000e+01)
(2.87,  1.289990300000000e+01)
(2.88,  1.312787200000000e+01)
(2.89,  1.335756900000000e+01)
(2.90,  1.358900000000000e+01)
(2.91,  1.382217100000000e+01)
(2.92,  1.405708800000000e+01)
(2.93,  1.429375700000001e+01)
(2.94,  1.453218400000000e+01)
(2.95,  1.477237500000000e+01)
(2.96,  1.501433600000000e+01)
(2.97,  1.525807300000000e+01)
(2.98,  1.550359200000000e+01)
(2.99,  1.575089900000001e+01)
(3.00,  1.600000000000000e+01)

Secant method:
   n            x                     f(x)
---------------------------------------------------------
   0   3.000000000000000e+00   1.600000000000000e+01
   1   2.058823529411764e+00  -3.907999185833528e-01
   2   2.081263659845023e+00  -1.472040595537543e-01
   3   2.094824146094052e+00   3.043795598889787e-03
   4   2.094549431035247e+00  -2.288658065374705e-05
   5   2.094551481227599e+00  -3.512811375117053e-09
   6   2.094551481542327e+00   3.552713678800501e-15

Final computed result for root using secant method:
r =   2.094551481542327e+00

Newton's method:
   n            x                     f(x)
---------------------------------------------------------
   0   2.000000000000000e+00  -1.000000000000000e+00
   1   2.100000000000000e+00   6.100000000000083e-02
   2   2.094568121104185e+00   1.857231732707021e-04
   3   2.094551481698199e+00   1.739761223973346e-09
   4   2.094551481542327e+00  -8.881784197001252e-16

Final computed result for root using newton's method:
r =   2.094551481542327e+00

Bisection method:
   n            c                     f(c)
---------------------------------------------------------
   1   2.500000000000000e+00   5.625000000000000e+00
   2   2.250000000000000e+00   1.890625000000000e+00
   3   2.125000000000000e+00   3.457031250000000e-01
   4   2.062500000000000e+00  -3.513183593750000e-01
   5   2.093750000000000e+00  -8.941650390625000e-03
   6   2.109375000000000e+00   1.668357849121094e-01
   7   2.101562500000000e+00   7.856225967407227e-02
   8   2.097656250000000e+00   3.471428155899048e-02
   9   2.095703125000000e+00   1.286233216524124e-02
  10   2.094726562500000e+00   1.954347826540470e-03
  11   2.094238281250000e+00  -3.495149197988212e-03
  12   2.094482421875000e+00  -7.707752083661035e-04
  13   2.094604492187500e+00   5.916926729696570e-04
  14   2.094543457031250e+00  -8.956467604548379e-05
  15   2.094573974609375e+00   2.510581462900063e-04
  16   2.094558715820312e+00   8.074527208989934e-05
  17   2.094551086425781e+00  -4.410067734994527e-06
  18   2.094554901123047e+00   3.816751073770774e-05
  19   2.094552993774414e+00   1.687869864142044e-05
  20   2.094552040100098e+00   6.234309738673005e-06
  21   2.094551563262939e+00   9.121195727601616e-07
  22   2.094551324844360e+00  -1.748974437276729e-06
  23   2.094551444053650e+00  -4.184275219643041e-07
  24   2.094551503658295e+00   2.468460031934683e-07
  25   2.094551473855972e+00  -8.579076471448843e-08
  26   2.094551488757133e+00   8.052761835131150e-08
  27   2.094551481306553e+00  -2.631573181588465e-09
  28   2.094551485031843e+00   3.894802169668310e-08
  29   2.094551483169198e+00   1.815822514572574e-08
  30   2.094551482237875e+00   7.763325982068636e-09
  31   2.094551481772214e+00   2.565876400240086e-09
  32   2.094551481539384e+00  -3.284839067418943e-11
  33   2.094551481655799e+00   1.266514004782948e-09
  34   2.094551481597591e+00   6.168328070543794e-10
  35   2.094551481568487e+00   2.919922081900950e-10
  36   2.094551481553935e+00   1.295710205795331e-10
  37   2.094551481546659e+00   4.836131495267182e-11
  38   2.094551481543022e+00   7.755573960821494e-12
  39   2.094551481541203e+00  -1.254640835668397e-11
  40   2.094551481542112e+00  -2.394529019511538e-12

Final computed result for root using bisection method:
```

```
r =   2.094551481542112e+00
```

Q3

6 / 6

In the Newton method, we progress in each step from a given point x to a new point $x - d$, where $d = f(x)/f'(x)$. It is possible that $|f(x-d)| \geq |f(x)|$ and the method may not converge. A refinement of the method is as follows: If $|f(x-d)| \geq |f(x)|$, then reject the value of d and use $d := d/2$ instead. We continue this process until $|f(x-d)| < |f(x)|$, and then the new point is defined as $x - d$. This method is called the Newton trust region method. Write a MATLAB program for this method and use it to solve $\arctan(x) = 0$. For comparison, also use the Newton method to solve it. In your test, for both methods, take $x_0 = 4$, `xtol=1.e-12`, `ftol=1.e-12` and `nmax=20`. Print out results for each iteration step and the MATLAB code for the modified method.

$\checkmark$

Newton_trust_region.m:

```
function r = newton_trust_region(f,fd,x,xtol,ftol,nmax,display)
% Newton’s method for solving f(x)=0.
% r = newton_trust_region(f,fd,x,xtol,ftol,n_max,display)
% input:  f is the handle of the function f(x).
%         fd is the handle of the derivative f’(x).
%         x is the initial point
%         xtol and ftol are termination tolerances
%         nmax is the maximum number of iterations
%         display = 1 if step-by-step display is desired,
%                 = 0 otherwise
% output: root is the computed root of f(x)=0
n = 0;
fx = f(x); % f is either a string name or the handle of f(x)
if display
   disp('   n              x                     f(x)');
   disp('-------------------------------------------------------');
   disp(sprintf('%4d %23.15e %23.15e', n, x, fx));
end
if abs(fx) <= ftol, r = x; return, end
for n = 1:nmax
   fdx = fd(x);
   d = fx/fdx;
   % Check if we need to adjust the step size
   fx_new = f(x - d);
   while abs(fx_new) >= abs(fx)
       d = d / 2; % Keep halving until improvement
       fx_new = f(x - d);
   end
   x = x - d;
   fx = f(x);
   if display, disp(sprintf('%4d %23.15e %23.15e', n, x, fx)), end

   if abs(d) <= xtol | abs(fx) <= ftol
       r = x;
       return
   end
end
r = x;
end
```

A4Q3.m:

```
% Assignment 4 - Question 3
f = @(x) atan(x);
fd = @(x) 1 ./ (1 + x.^2);
% Normal newton method
disp ("Newton's method: ");
r = newton(f, fd, 4, 1.e-12, 1.e-12, 20, 1);
disp(" ");
disp("Final computed result for root using newton's method:");
fprintf("r = %23.15e", r);
disp(" ");
% Newton trust region method
disp(" ");
disp ("Newton trust region method: ");
r = newton_trust_region(f, fd, 4, 1.e-12, 1.e-12, 20, 1);
disp(" ");
disp("Final computed result for root using newton trust region method:");
fprintf("r = %23.15e", r);
disp(" ");
```

Output of running A4Q3.m (command window):

```
>> A4Q3
Newton's method:
   n              x                     f(x)
-------------------------------------------------------
   0   4.000000000000000e+00   1.325817663668033e+00
   1  -1.853890028235655e+01  -1.516907918610638e+00
   2   5.043253396289842e+02   1.568813482367175e+00
   3  -3.985156154371373e+05  -1.570793817482937e+00
   4   2.494647436911540e+11   1.570796326790888e+00
   5  -9.775483113434105e+22  -1.570796326794897e+00
   6   1.501054071029691e+46   1.570796326794897e+00
   7  -3.539260673251353e+92  -1.570796326794897e+00
```

```
Newton trust region method:
   n              x                     f(x)
-------------------------------------------------------
   0   4.000000000000000e+00   1.325817663668033e+00
   1  -1.634725070589139e+00  -1.021801056701445e+00
   2   2.414682519082165e-01   2.369328040224536e-01
   3  -9.279373377040656e-03  -9.279107051842752e-03
   4   5.326687413237385e-07   5.326687413236881e-07
   5  -1.007969207232617e-19  -1.007969207232617e-19

Final computed result for root using newton trust region method:
r =  -1.007969207232617e-19
```