UBC CPSC 302 Num Comp for Alg Problems, 2019W

Assignment 3

by j0k0b #42039157

Tips: Use the tips from Assignment 1 to get the most credit possible on this and future assignments. Replace the [Preferred Name or CS-ID] and [Student Number] above.

Table of Contents

Acknowledgments

You are welcome to work with other students on this assignment, but everything in this file must clearly be your own original work. Complete the following section to recognize your collaborators on this assignment.

Begin acknowledgments

Thanks to q5c1b for their help with understanding the question 2.

Other than the contributions above, this work is my own.

— j0k0b

End acknowledgments

Question 1 Comparison of methods

For $x > 0$ consider the problem

$$f(x) = x + \ln(x) = 0.$$

There is exactly one root $f(x^*) = 0,\ 0 < x^* < \infty$.

Question 1.1

Write MATLAB code to plot a graph of the function on the interval $[0.1,\ 1]$. Include the "`grid on`" or other commands to show approximately where the root is.

Answer 1.1 [2 marks]

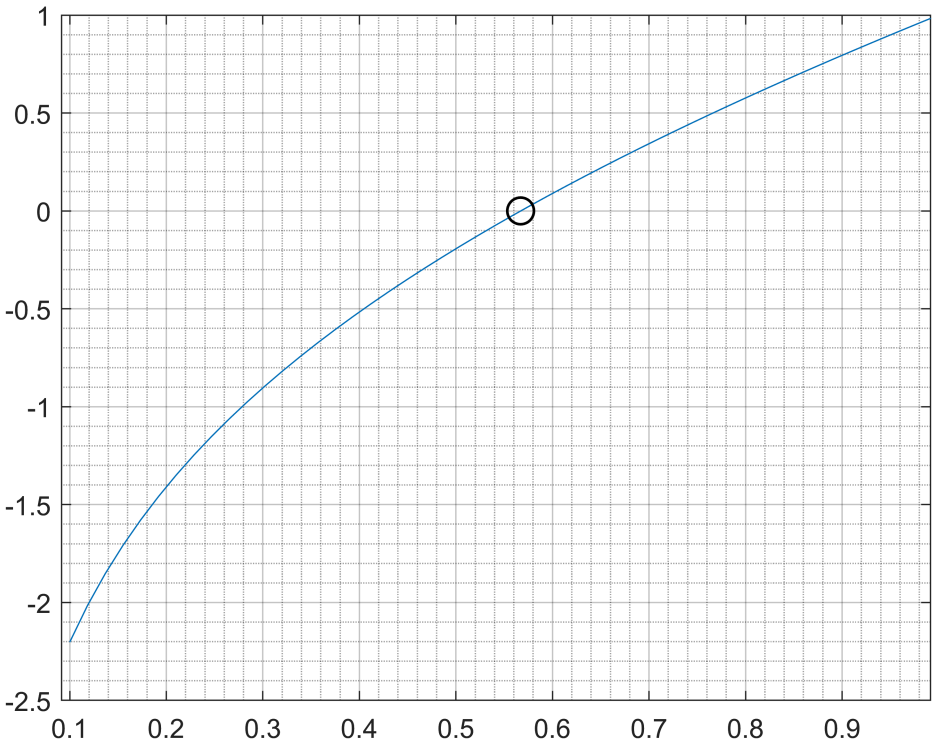
Enter your solution in the space below.

Tips: Insert clear, functioning MATLAB code as necessary. Run your code to produce output (data and/or figures) before saving this MLX file and exporting it as a PDF. Every plot should be titled, contain a legend, contain labeled axes, and use a unique colour, line style and/or marker for each dataset.

Adjust the size (drag corners) of inserted images so they don't get cropped when exporting to PDF.

Begin answer

```
x = linspace(0.1, 1.0, 50); % we are given the interval b/w [0.1, 1.0]
f_x = x + log(x); % f(x) = x + ln(x) = 0
plot(x,f_x) % root between 0.5 and 0.6
hold on
plot(0.567,0,'ko','MarkerSize',10,'LineWidth',1); % x = -ln(x); e^x = 1/x
grid on % shows grid to clarify root location
grid minor % more significant number of grids
hold off
```



End answer

Further instructions for Question 1

For each of the iterative methods below:

- Use $|x_k - x_{k-1}| < 10^{-10}$ as a convergence criterion.
- Store the iterates $x_k,\ k = 1, 2, \ldots$ in a variable for later comparison with other methods.

Question 1.2 Bisection method

Use the Intermediate Value Theorem to argue that the bisection method will converge to the root from the initial interval $[0.5, 0.6]$.

Apply the bisect function (code provided below) using the instructions above to find the root. Demonstrate that you've chosen a tolerance that satisfies the convergence criterion above.

Answer 1.2 Bisection method [2 marks]

Enter your solution in the space below.

Tips: Add your function to the My functions *section below.*

Begin answer

```
bis = bisect('fx', 0.5, 0.6, fx(0.1), fx(1.0), 1.e-10);
table(bis) % the table depicts that it does converge to 0.5671
```

ans = 29×1 table

	bis
1	0.5500
2	0.5750
3	0.5625
4	0.5687
5	0.5656
6	0.5672
7	0.5664
8	0.5668
9	0.5670
10	0.5671
11	0.5671
12	0.5672
13	0.5672
14	0.5671
15	0.5671
16	0.5671
17	0.5671
18	0.5671
19	0.5671
20	0.5671
21	0.5671
22	0.5671
23	0.5671
24	0.5671
25	0.5671
26	0.5671
27	0.5671
28	0.5671
29	0.5671

```
fprintf("x* == %.5f.", bis(end))
```

```
x* == 0.56714.
```

End answer

Question 1.3 Fixed point iteration

Write MATLAB code for finding the root, using a linearly convergent fixed point iteration, with $x_0 = 0.5$. Show that the conditions of the Fixed Point Theorem (for the function *g* you have selected) are satisfied. (You are encouraged to write your code as a function below – for later reuse in Question 2 – just indicate so to the reader so they can find it.)

Apply your fixed point iteration using the instructions above to find the root.

Answer 1.3 Fixed point iteration [2 marks]

Enter your solution in the space below.

Tips: You may use and modify code from lesson Fixed point iteration Section 6.2 in your answer.

Begin answer

```
it = 50;
% fixed point iteration
xk = [];
notdone = 1;
xk(1) = 0.5; % first guess, x_0=0.5
xkplus1 = xk(1);
for k=1:it
    if (notdone == 0)
     break;
    end
    xk(k) = xkplus1;
    xkplus1 = exp(-xk(k));

    if abs(xkplus1-xk(end)) < 1.e-10 || xkplus1 == Inf
        notdone = 0;
    end
end
fix = xk';
table(fix)
```

ans = 38×1 table

	fix
1	0.5000
2	0.6065
3	0.5452
4	0.5797
5	0.5601
6	0.5712

	fix
7	0.5649
8	0.5684
9	0.5664
10	0.5676
11	0.5669
12	0.5673
13	0.5671
14	0.5672
15	0.5671
16	0.5672
17	0.5671
18	0.5671
19	0.5671
20	0.5671
21	0.5671
22	0.5671
23	0.5671
24	0.5671
25	0.5671
26	0.5671
27	0.5671
28	0.5671
29	0.5671
30	0.5671
31	0.5671
32	0.5671
33	0.5671
34	0.5671
35	0.5671
36	0.5671
37	0.5671
38	0.5671

```
fprintf('x* == %.5f.', xkplus1)
```

```
x* == 0.56714.
```

End answer

Question 1.4 Newton's method

Write MATLAB code for finding the root, using Newton's method, with $x_0 = 0.5$. (You are encouraged to write your code as a function below – for later reuse in Question 2 – just indicate so to the reader so they can find it.)

Apply your Newton's method using the instructions above to find the root.

Answer 1.4 Newton's method [2 marks]

Enter your solution in the space below.

Tips: You may use and modify code from lesson Newton's method and variants Section 4.2 in your answer.

Begin answer

```
it = 50;
% fixed point iteration
xk = [];
notdone = 1;
xk(1) = 0.5; % first guess, x_0=0.5
xkplus1 = xk(1);
for k=1:it
    % next guess
    if (notdone == 0)
        break;
    end
    xk(k) = xkplus1;
    xkplus1 = xk(k) - (xk(k)+ log(xk(k)))/(1+1 / xk(k));
    % iterate
    if (abs(xkplus1-xk(end)) < 1.e-10 || xkplus1 == Inf)
        notdone = 0;
    end
end
new = xk';
table(new)
```

ans = 4×1 table

	new
1	0.5000
2	0.5644
3	0.5671
4	0.5671

```
fprintf('x* == %.5f.', xkplus1)
```

```
x* == 0.56714.
```

End answer

Question 1.5 Secant method

Write MATLAB code for finding the root, using the secant method, with $x_0 = 0.5$ and $x_1 = 0.6$. (You are encouraged to write your code as a function below – for later reuse in Question 2 – just indicate so to the reader so they can find it.)

Apply your secant method using the instructions above to find the root.

Answer 1.5 Secant method [2 marks]

Enter your solution in the space below.

Tips: You may use and modify code from lesson Newton's method and variants Section 8 in your answer.

Begin answer

```
% draw g(x) and guide y=x
it = 50;
notdone = 1;
% fixed point iteration
xk = [];
xkminus1 = 0.6;
f_minus = fx(xkminus1);
xkplus1 = 0.5;
for k=1:it
    if (notdone == 0)
        break;
    end
    xk(end+1) = xkplus1;
    f_k = fx(xk(end));
    xkplus1 = xk(end) - f_k * (xk(end) - xkminus1)/(f_k - f_minus);

    if (abs(xkplus1-xk(end)) < 1.e-10 || xkplus1 == Inf)
        notdone = 0;
    end
    xkminus1 = xk(end);
    f_minus = f_k;
end
sec = xk';
table(sec)
```

```
ans = 5×1 table
```

	sec
1	0.5000
2	0.5684
3	0.5672

	sec
4	0.5671
5	0.5671

```
fprintf('x* == %.5f.', xkplus1)
```

```
x* == 0.56714.
```

End answer

Question 1.6 Explain

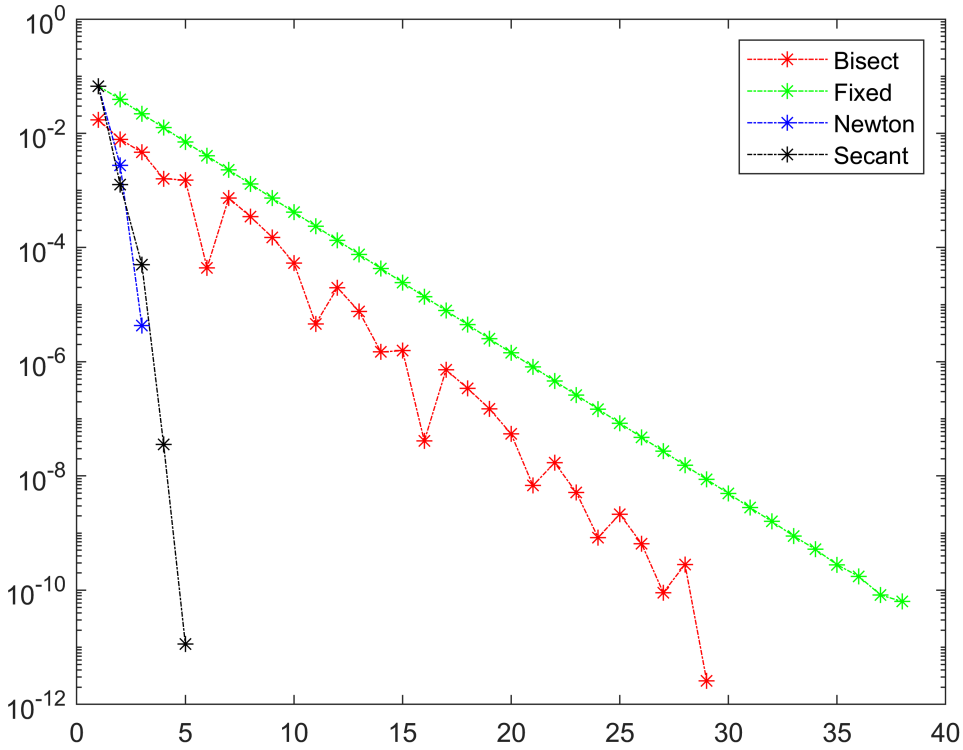
Plot the iterates for all of your above methods (§1.2–1.5) on one graph in order to show their progression. Explain the convergence behavior of each and how it matches theoretical expectations.

Answer 1.6 Explain [2 marks]

Enter your solution in the space below.

Begin answer

```
f_bis = 1:1:length(bis);
f_fix = 1:1:length(fix);
f_new = 1:1:length(new);
f_sec = 1:1:length(sec);
best = new(end);
semilogy(f_bis, abs(bis - best), '-.r*');
hold on;
semilogy(f_fix, abs(fix - best), '-.g*');
semilogy(f_new, abs(new - best), '-.b*');
semilogy(f_sec, abs(sec - best), '-.k*');
legend('Bisect', 'Fixed','Newton', 'Secant');
hold off;
```



It is apparent when we look at the semilog graph (base 10), we can see that Newton's method is the fastest followed by the secant method and bisection. Fixed point iteration depicts linear convergence, while bisection tends to be linear with a few peaks and troughs. Secant shows superlinear convergence, while Newton's shows a more quadratic output for convergence.

End answer

Question 2 Find the global minimum

Write a MATLAB program to minimize this smooth, scalar function in one variable

$$\phi(x) = \cos(x)\left(\frac{1}{2} - xe^{-x^2}\right)$$

over the interval $[-25, 25]$.

Your program should:

1. First find and display all critical points, i.e., zeros of $\phi'(x)$.
2. Then it should determine which of these correspond to a local minimum by checking the sign of ϕ'', and display the local minimum points.
3. Finally, it should determine the global minimum point(s), value and arguments, by minimizing over the local minimum values.

Briefly justify your choice of algorithm(s) to find the critical points. (I.e. what algorithmic properties are most/least important to you for this problem and how does your choice reflect your priorities?)

(If you coded any of your algorithms for Question 1 as functions, you are welcome to use them here as part of your solution.)

Answer 2 Find the global minimum [8 marks]

Enter your solution in the space below.

Begin answer

```
syms phi(x);
phi(x) = cos(x)*(1/2 - x*exp(-x^2));
syms phi_1(x);
phi_1(x) = diff(phi, x);
syms phi_2(x);
phi_2(x) = diff(phi_1, x);
f = matlabFunction(phi_1);
g = matlabFunction(phi_2);
n = -8:1:8;
i = n*pi;
roots = unique(newt(f, g, i)); % same function as used in 1, but extra outer loop
table(roots')
```

ans = 17×1 table

	Var1
1	-25.1327
2	-21.9911
3	-18.8496
4	-15.7080
5	-12.5664
6	-9.4248
7	-6.2832
8	-3.1416
9	0
10	3.1416
11	6.2832
12	9.4248
13	12.5664
14	15.7080
15	18.8496
16	21.9911
17	25.1327

```
phi_2_roots = [];
for i=1:1:length(roots)
    phi_2_roots(i) = phi_2(roots(i));
end

table(roots', phi_2_roots')
```

ans = 17×2 table

	Var1	Var2
1	-25.1327	-0.5000
2	-21.9911	0.5000
3	-18.8496	-0.5000
4	-15.7080	0.5000
5	-12.5664	-0.5000
6	-9.4248	0.5000
7	-6.2832	-0.5000
8	-3.1416	0.4947
9	0	-0.5000
10	3.1416	0.5053
11	6.2832	-0.5000
12	9.4248	0.5000
13	12.5664	-0.5000
14	15.7080	0.5000
15	18.8496	-0.5000
16	21.9911	0.5000
17	25.1327	-0.5000

```
roots = roots(find(phi_2_roots > 0))';
table(roots)
```

ans = 8×1 table

	roots
1	-21.9911
2	-15.7080
3	-9.4248
4	-3.1416
5	3.1416
6	9.4248
7	15.7080

	roots
8	21.9911

```
roots(end + 1) = -25;
roots(end + 1) = 25;

phi_roots = [];
for i=1:1:10
    phi_roots(end + 1) = phi(roots(i));
end

phi_roots = round(phi_roots, 1)';
table(roots, phi_roots)
```

ans = 10×2 table

	roots	phi_roots
1	-21.9911	-0.5000
2	-15.7080	-0.5000
3	-9.4248	-0.5000
4	-3.1416	-0.5000
5	3.1416	-0.5000
6	9.4248	-0.5000
7	15.7080	-0.5000
8	21.9911	-0.5000
9	-25.0000	0.5000
10	25.0000	0.5000

```
% Global Minimum
roots = min(roots(find(phi_roots == -0.5)))
```

```
roots = -21.9911
```

End answer

3 My functions

MATLAB requires all functions to be at the end of the script. Enter functions you wrote for this assignment in the space below.

Tips: You only need to complete this section if you developed and used your own MATLAB function(s) in your code above. If you do define your own functions, organize your code; use indenting and reasonable variables (e.g. no unused variables or ambiguous names). Document your code clearly and effectively with comments (`% This is a comment`). Note the specific purpose of each function; indicate input requirements and output results.

Begin functions

```
function roots = newt(f, g, i)
it = 50;
roots = [];
% fixed point iteration

for x0 = i
    it = 0;
    notdone = 1;
    xk = [];
    xk(1) = x0;
    xkplus1 = x0;

    for k=1:it
        if (notdone == 0)
            break;
        end
        xk(k) = xkplus1;
        xkplus1 = xk(k) - f(xk(k))/g(xk(k));
        if abs(xkplus1-xk) < 1.e-8 || xkplus1 == Inf
            notdone = 0;
        end
    end
    roots(end+1) = xkplus1;
end
end

function f = fx(x)
f = x + log(x);
end
```

End functions

4 Provided functions

Additional utility functions are provided here. You should not need to modify these functions and may use them "as-is" in your code.

4.1 bisect function

```
function p = bisect(func, a,b, fa,fb, atol)
%
% function p = bisect(func,a,b,fa,fb,atol)
%
% Assuming fa = func(a), fb = func(b), and fa * fb < 0,
% there is a value root in (a,b) such that func(root) = 0.
% This function returns in p the bisection iterates such that
% | p(end) - root | < atol
% and in length(p) the number of iterations required. Bisection root-finding method
```

```
% modified from lesson "Nonlinear equations in one variable.  Bisection method"
% and [AG Book] Section 3.2.
if (a >= b) || (fa*fb >= 0) || (atol <= 0)
    disp('something wrong with the input: quitting');
    p = NaN;
    return;
end
n = ceil( log2(b-a) - log2(2*atol) );
p = zeros(n,1); % pre-allocate storage array of midpoints, p[1..n]
for k=1:n
    p(k) = (a+b)/2;
    fp = feval(func,p(k));
    if fa * fp < 0
        b = p(k);
        %fb = fp;    % value assigned to fb here is unused
    else
        a = p(k);
        fa = fp;
    end
end
p(k) = (a+b)/2;
end % bisect
```