Damian Franco

CS 375 – 001

Homework #3:

1) For problem 1, we were given a prompt and equation to solve for the *optimal value* of *h.* In homework 1 we approximated a derivative, now we must compare the results of the exact stencil and computer stencil. I based my understanding of this problem off the hint Professor Lau gave us in the Slack channel, thank you for that and all the other guidance. The MATLAB function below will approximate accordingly. It will first set up the variables for the equation we are approximating and then it will set up the exact stencil equation and the computer stencil equation based off the f(x) equation we were given. Then the rest will just approximate based off these equations and graph them versus *h*. The graphs show that when compared to problem 1 on homework 1, that this is a <u>*highly more efficient*</u> way to find the derivative of a function.

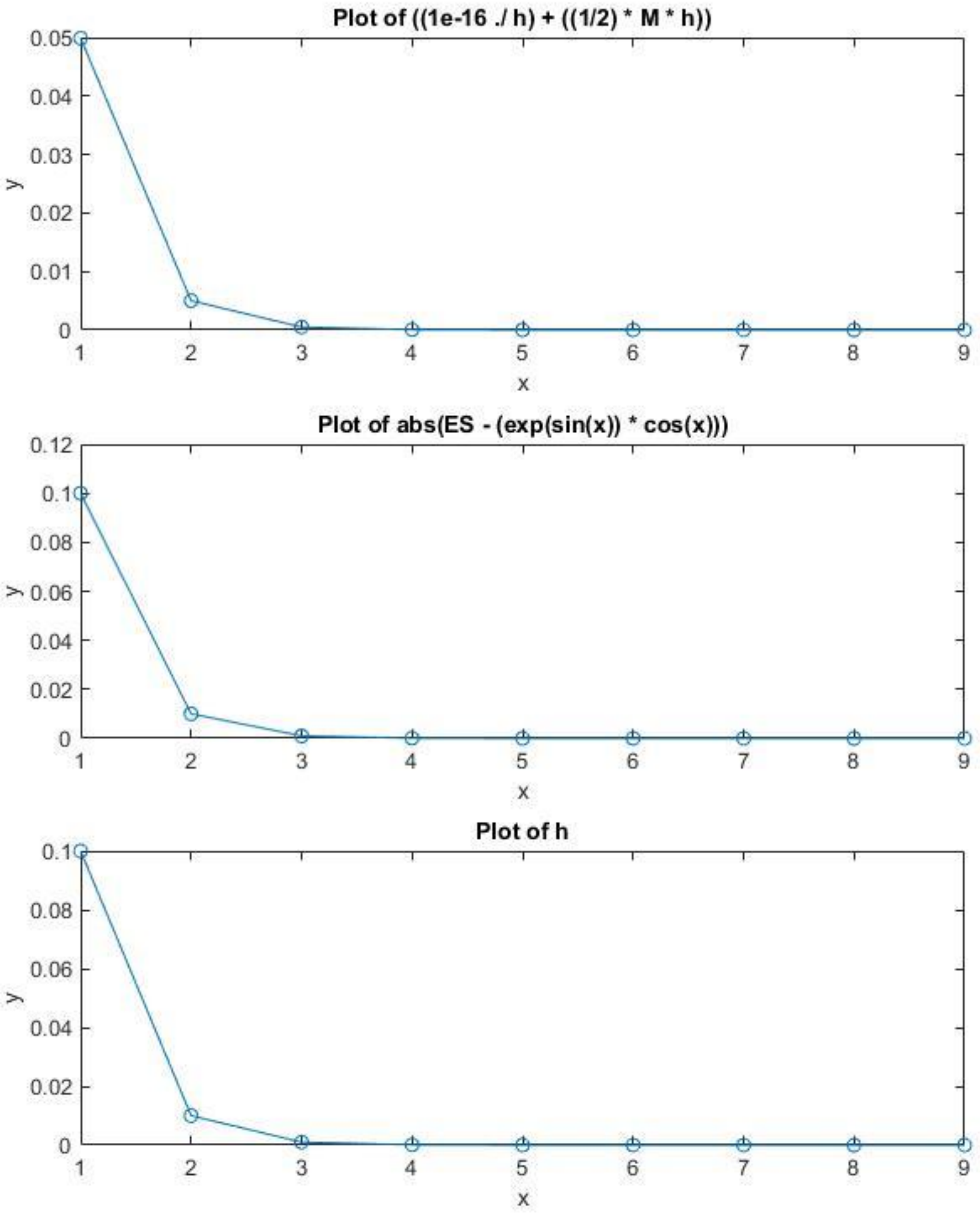
Source Code

HW3_1.m:

```
%
% function: HW3_1
%
% Set up and arithmetic for homework 3, number 1.
% This is based off of the hint Professor Lau
% gave us in the Slack channel. Therefore, this function
% is the estimate of the optimal value of h for the
% derivative of f.
%
% Variable set up
x = pi;
f = @(x) exp(sin(x));
h = logspace(-1,-9,9);

% Orginal eqs
ES = (f(x + h) - f(x)) ./ h;
CS = ES + (1e-16) ./ h;

% New variable set up
c = 0.9952696331; %f''(h)
M = 2;

% Inequality set up
b1 = abs(ES - (exp(sin(x)) * cos(x)));
b2 = ((1e-16 ./ h) + ((1/2) * M * h));

% Graph set up
subplot(3,1,1)
title('Plot of abs(ES - (exp(sin(x)) * cos(x)))')
xlabel('x')
ylabel('y')
plot(b1, '-o')
subplot(3,1,2)
xlabel('x')
ylabel('y')
title('Plot of ((1e-16 ./ h) + ((1/2) * M * h))')
plot(b2, '-o')
subplot(3,1,3)
xlabel('x')
ylabel('y')
title('Plot of h')
plot(h, '-o')
```

Graphs

HW3-1-Graphs.jpg:



2) For question 2 on the homework, we had to compare the roots of function through two different methods. The first was the bisection method we had learned in the recordings by Professor Lau. He provided us with a clear understanding of how to do the bisection method and make a function to do so in MATLAB, and that is what the *bisection.m* code is based on. The second, was the fzero function call method and check where we get the root at between an interval. I set both functions up in MATLAB and compared the answers on a table. The roots seem to be calculating *close* to each other beside a *few precision differences* on the end of the roots, but besides that, both methods look pretty good. My work for this problem is shown below.

Source code:

HW_2.m:

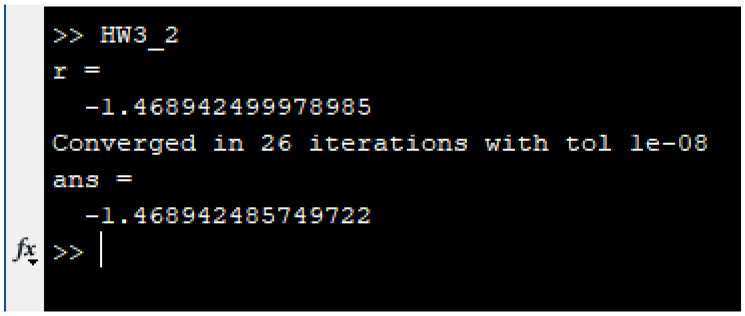
```
%
% function HW3_2
% This sets up and runs all the arithmetic
% behind number 2 on HW3
%
y = @(x) exp(x+2) + x^3 - x;
ftol = 1e-9;
MyOptions = optimset('TolX', ftol);
% fzero method
% r0 = -1 for root1
% r0 = 0 for root2
% r0 = 1 for root3
r0 = 100;
r = fzero(y, r0, MyOptions);
% bisection method
% [-2, -1] for root1
% [0, 0.5] for root2
% [0.5, 1] for root3
bisection(y, 0.5, 1)
```

bisection.m:

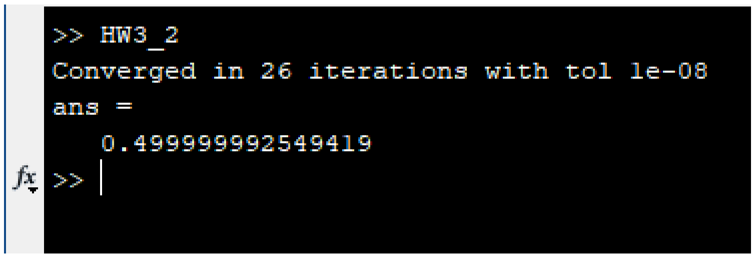
```matlab
function [m k] = bisection(f, a, b, tol, kmax)
%
% function [m k] = bisection(f, a, b, tol, kmax)
% This function will approximate the  root of the
% function being passed through.
%
switch nargin
    case 4,
        kmax = 1e5;
    case 3,
        tol = 1e-8;
        kmax = 1e5;
    case 5,
        % fall through and assign nothing
    otherwise,
        error('Wrong # or args')
end

if (b <= a)
    error('b > a not satisfied');
end
fa = f(a); if fa == 0, m = a; return; end;
fb = f(b); if fb == 0, m = b; return; end;
if (fa * fb >= 0)
    error('Intial interval error');
end

k = 0;
m = 0.5 * (a + b);
err = 0.5 * (b - a);
tol2 = tol + eps * max(abs(a), abs(b));
while (err >= tol2)
    fm = f(m);
    k = k + 1;
    if (fm == 0)
        return;
    end
    if (fa * fm < 0)
        b = m;
        fb = fm;
    else
        a = m;
        fa = fm;
    end
    m = 0.5 * (a + b);
    err = 0.5 * (b - a);
    tol2 = tol + eps * max(abs(a), abs(b));

    if (k >= kmax)
        disp(['Reached max iterations ', num2str(kmax), '. Error is ',
                   num2str(err)]);
    end
end
disp(['Converged in ', num2str(k), ' iterations with tol ',
            num2str(tol)]);
```

Outputs :

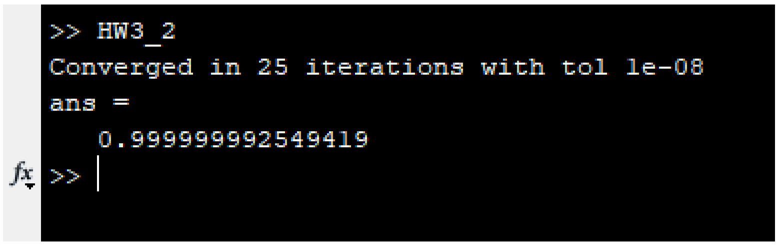
HW3_2-OUTS.jpg:



```
>> HW3_2
r =
  -1.468942499978985
Converged in 26 iterations with tol 1e-08
ans =
  -1.468942485749722
>> 
```



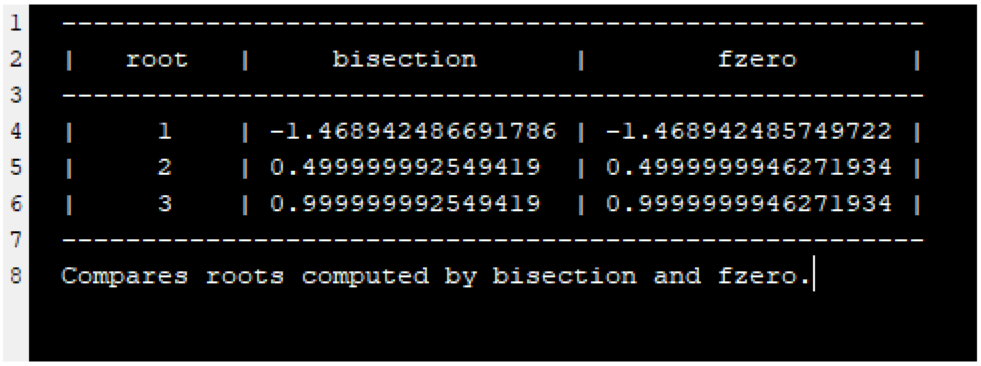
```
>> HW3_2
Converged in 26 iterations with tol 1e-08
ans =
   0.499999992549419
>> 
```



```
>> HW3_2
Converged in 25 iterations with tol 1e-08
ans =
   0.999999992549419
>> 
```

Table Comparison:

HW3_2-Table.txt:



```
-----------------------------------------------------
|   root    |     bisection      |        fzero       |
-----------------------------------------------------
|     1     | -1.468942486691786 | -1.468942485749722 |
|     2     | 0.499999992549419  | 0.4999999946271934 |
|     3     | 0.999999992549419  | 0.9999999946271934 |
-----------------------------------------------------
Compares roots computed by bisection and fzero.
```

3) Question 3 on the homework asks us to find the fixed point of each function given in part (a) and part (b) and decide whether it is locally convergent to that fixed points. These given functions did not look too crazy, so my initial guess was that both were going to converge locally in very few iterations. I followed Professor Lau's rules for fixed point in the narrations of root2 to solve this one and I came up with the following:

   Part (a):
   For this part, the MATLAB function *iteration.m* shows that function has a fixed point at the value *1.5*. The function converges with the tolerance of 1e-8 in *two iterations*.

   Part (b):
   For this part, we do the same by plugging in the equation to the MATLAB function *iteration.m*. This shows that function has a fixed point at the value *-0.5*. The function converges with the tolerance of 1e-8 in *two iterations*, just like in part (a).

Source code:

HW3_3.m:

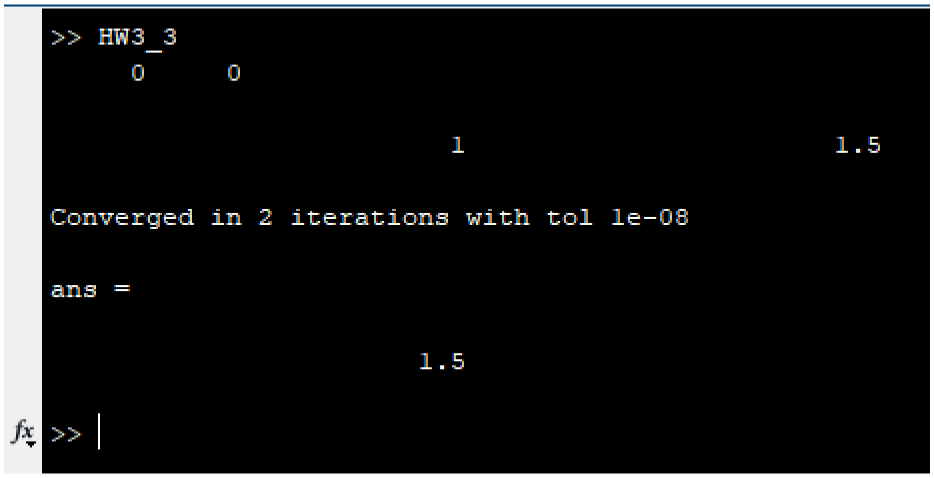
```
%
% function HW3_3
% This sets up and runs all the arithmetic
% behind number 3 on HW3
%
% 3a:
a = @(x) x^2 - 3/2 * x + 3/2;
% 3b:
b = @(x) x^2 + 1/2 * x - 1/2;
% Test cases set up
x0 = 0;
tol = 1e-8;
kmax = 1e5;
% Test
iteration(b, x0, tol, kmax)
```

iteration.m:

```
function x = iteration(f, x0, tol, kmax)
%
% function x = iteration(f, x0, tol, kmax)
% This function will find the fixed point of the function
% and show whether it converges to it or if it computes
% an error.
%
switch nargin
    case 2,
        tol = 1e-8;
        kmax = 1e5;
    case 3,
        kmax = 1e5;
    case 4,
        % fall through and assign nothing
    otherwise,
        error('Wrong # or args')
end
x = x0;
err = 100;
k = 0;
while (err >= tol)
    format long g;
    if (k < 20)
        disp([k x0]);
    end
    x = f(x0);
    err = abs(x - x0);
    x0 = x;
    k = k + 1;
    if (k >= kmax)
        disp(['No convergence after ', num2str(kmax) ' iterations'])
        disp(['Error at this stage is ' num2str(err)]);
        return;
    end
end
if (isfinite(x))
disp(['Converged in ', num2str(k) ' iterations with tol ' ...
            num2str(tol)])
else
    disp(['No convergence after ', num2str(k) ' iterations. Iterations'
            ...
            'overflowed to infinity'])
end
```

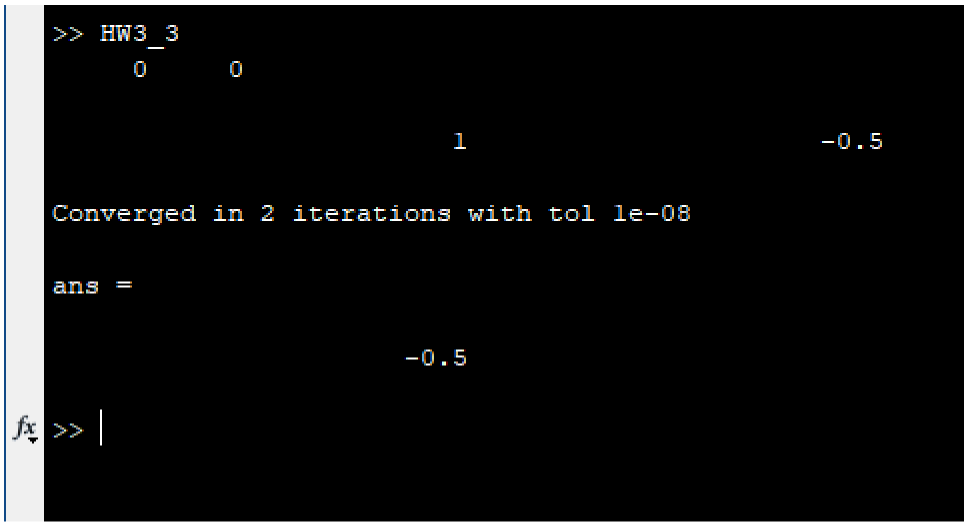
Outputs:

HW3_3-PART-A.jpg



```
>> HW3_3
     0      0

                          1                        1.5

Converged in 2 iterations with tol 1e-08

ans =

                        1.5

>> 
```

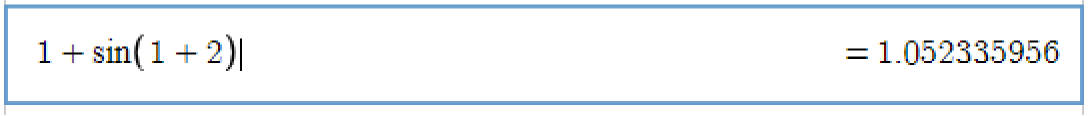
HW3_3-PART-B.jpg



```
>> HW3_3
     0      0

                          1                       -0.5

Converged in 2 iterations with tol 1e-08

ans =

                       -0.5

>> 
```

4) For problem 4, we were given a set of fixed points on the function phi(x). We were then given some rules to follow to help us solve this problem. First, the fixed points are set at (-3, -3), (1, 1) and (2, 2). Second, the slope at (-3, -3) is equal to 2.4. Lastly the fixed point 2 converges to 2, therefore -1 < fphi'(2) < 0. This problem was, once again, based off Professor Lau's help in the Slack channel. First, I drew the line y = x and plotted the fixed points on them. Then I noticed a pattern and figured out the equation must be y = x + sin(x – FP) for FP is equal to fixed point. Then I plotted that in MATLAB. After, I took that equation to Desmos calculator online and calculated the fphi'(1) must be equal to approximately _1.052335956_. The graphical representation is shown below, as well as the MATLAB code and the Desmos calculator.

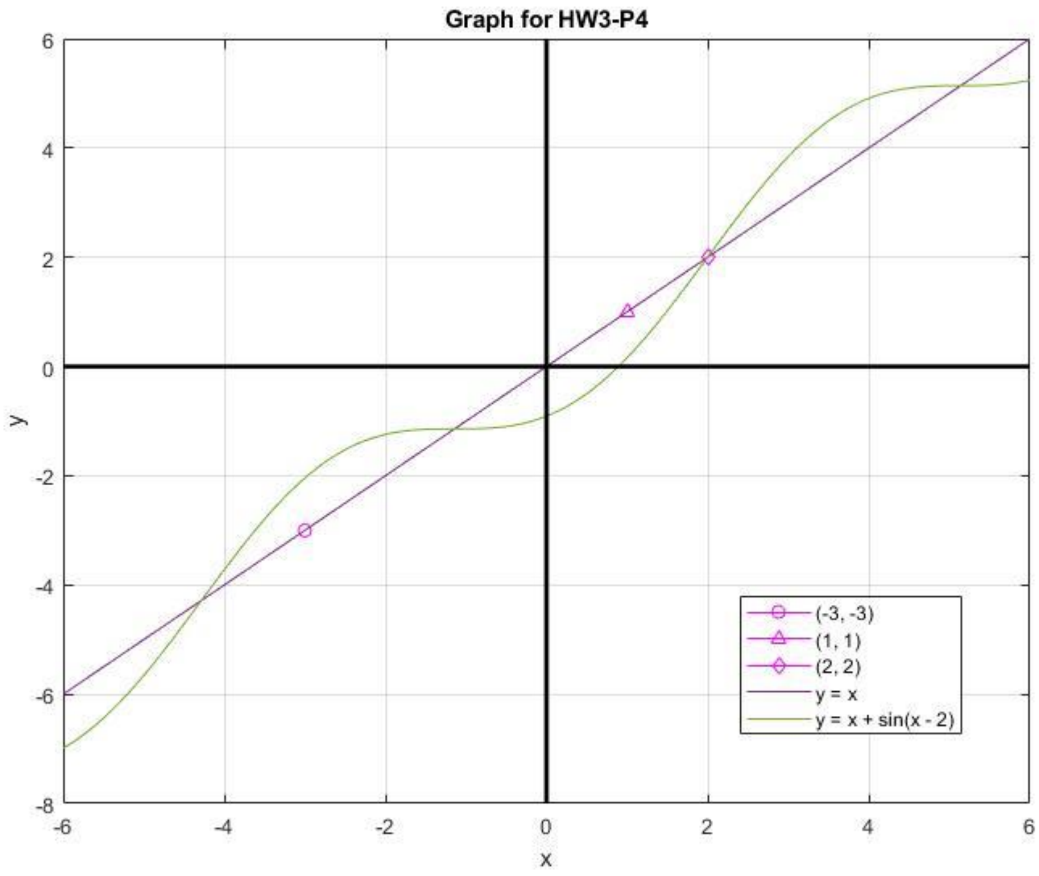
Desmos calculation:



$1 + \sin(1 + 2)$ $= 1.052335956$

Graphical Representation:

HW_3-4-Graph.jpg:



Source Code:

HW3_4.m:

```
%
% function HW3_3
% This sets up and runs all the arithmetic
% behind number 4 on HW3
%
% Set up equation
x = linspace(-6,6,100);
f = x;
fphi = x + sin(x - 2); % Guess of f(phi)
xaxis = 0;
yaxis = 0;
% Set up graph
plot(-3, -3, '-mo')
hold on
plot(1, 1, '-m^')
plot(2, 2, '-md')
plot(x, f);
plot(x, fphi);
line([0,0], ylim, 'Color', 'k', 'LineWidth', 2); % y axis
line(xlim, [0,0], 'Color', 'k', 'LineWidth', 2); % x axis
title('Graph for HW3-P4')
xlabel('x')
ylabel('y')
legend('(-3, -3)', '(1, 1)', '(2, 2)', 'y = x', 'y = x + sin(x - 2)')
hold off
grid on
```