Damian Franco

CS 375 – 001

Homework #4:

1) In question 1, we are given a function and the root to the function (r = 0). We first are asked to find the multiplicity of that function. The multiplicity is 1 at the root r. Next, we are asked to approximate the forward and backward errors on the function of the rA approximate root $r_A = 0.005$. We were taught in the narration in the cond module to find the backward error, we must use the equation $| f(r_A) |$ and to find the forward error, the equation to use is the $| (r_A - r) / r |$. We are given $r = 0$ and $r_A = 0.005$ to plug into the equations. First, I did the forward equation and noticed the that will divide by 0 because r is equal to 0. This will produce an undefined output, or when ran in MATLAB, a ***Inf*** output. The forward error for this equation is ***undefined***. Next, we approximate the backward error by plugging in $r_A = 0.005$ into the function sin(2x) - 2x, this will output the backwards error ***1.6667e-07***. The MATLAB code and windows are shown below.

Source Code:

HW4_1.m:

```
%
% function: HW4_1
%
% This function is the intialization and execution to problem
% 1 on the homework. This will set up r, rA and the function
% f and find the backward and foreward errors
%
r = 0;
ra = 0.005;
f = @(x) sin(2*x) - 2*x;
forward = abs((ra - r)/r);
backward = abs(f(ra));
forward
backward
```

MATLAB Output:

```
>> HW4_1
forward =
   Inf
backward =
   1.6667e-07
>> |
```

2) This problem had us figure out the root of $f(x) = (x - 1)(x - 2)(x - 3)(x - 4) - 10^{-6} * x^6$ using the sensitivity formula that was taught to us in the cond narrations. First, there are variables that need be set up. Here's the main variables, $\varepsilon_{mach} = 10^{-7}$, $r = 4$, $g(x) = x^6$, and $f'(x) = 4x^3 - 30x^2 + 70x - 50 - (3x^5 / 500000)$. Next, r must be plugged in to both f'(x) and g(x) and the following is the result, $f'(r) = 5.99385$ and $g(r) = 4096$. S_r (delta r) is the next step and $S_r = \varepsilon_{mach} * g(r) / f'(r)$ which is equal to $S_r = 0.0004096 / 5.993856$. Now S_r will be equal to 0.0000683366433895. The last step to find the approximation is to add r to S_r which results in *4.00006833664*. That is the approximation at the root r = 4 using the sensitivity formula. When compared to the fzero function in MATLAB, there are some *precision differences* but other than that, they seem to match up well. My source code and work are shown below.

Source Code:

HW4_2.m

```
%
% function: HW4_2
% This function will set up the approximation of the root near r = 4
% with two different ways. The first way is the Sensitivity
% approximation and the second way the MATLAB's fzero approximation.
%
% function f(x) set up
f = @(x) (x - 1) .* (x - 2) .* (x - 3) .* (x - 4) - 10e-6 * x.^6;

%Sensitivity approx
r = 4;
epmach = 1e-7;
g = @(x) x.^6;
fder = @(x) 4*x.^3 - 30*x.^2 + 70*x - 50 - (3*x.^5 / 500000);
fapprox = fder(r);
gapprox = g(r);
deltar = (epmach * gapprox) / fapprox;
ra = r + deltar;
ra

%fzero approx
x0 = 4; % initial point
fz = fzero(f,x0);
fz
```

MATLAB Output:

```
>> HW4_2
ra =
   4.000068336643389
fz =
   4.006811339711854
>>
```

3)

A) In part a, we were assigned to find the root of the function f(x) = 2xcos(x) – 2x + sin(x^3) with MATLAB's fzero function call and then report the forward and backwards errors. When approximating the root with fzero, the function in MATLAB will return the number *1.6881483488857743e-04* which is approximately 0.0001688 which is very close to the number 0 which is the true root. It is noticeable here how the fzero precision may be a little off when trying to calculate the true root. Next, the variables r = 0.1 and r_A = 1.68814e-4 are assigned and used to calculate the forward and backward errors. When calculated, the *forward error will equal 0.99983118516511* and the *backward error will equal 0*.

Source Code:

HW4_3-A.m

```
%
% function: HW4_3-A
% This function will set up and approximate the root
% for the function given to us in problem 3 and find
% the forward and backward error approximations
%
% function set up
f = @(x) 2*x*cos(x)-2*x+sin(x.^3);
% Part A
% fzero approx
x0 = [-0.1 0.2]; % initial point
fz = fzero(f,x0);
fz
r = 0.1;
ra = fz;
forward = abs(ra - r / r);
backward = abs(f(ra));
forward
backward
```

MATLAB Output:

```
>> HW4_3
fz =
     1.688148348885743e-04
forward =
   0.999831185165111
backward =
     0
>>
```

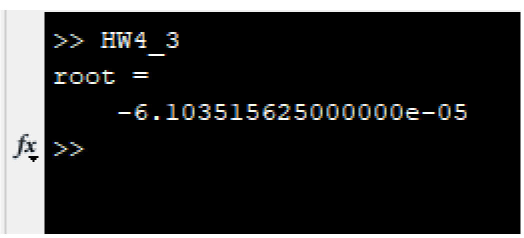
B) For part B, we had to approximate the root in the interval [-0.1, 0.2] with the bisection method instead of the fzero function. The bisection function that I had previously written in a different homework assignment was used and it approximated the root at *-6.103515625e-05*. This root was not far off from the fzero approximation, but it is not very similar to that approximation. It seems to have some sort of precision error when approximating this function. This function also has many roots within the range of -0.01 through 0.01which might be causing these precision errors.

Source Code:

HW4_3-B.m

```
%
% function: HW4_3-B
% This function will set up and approximate the root
% for the function given to us in problem 3 using the
% bisection method.
%
% function set up
f = @(x) 2*x*cos(x)-2*x+sin(x.^3);
% Part B
% bisection approx
x0 = [-0.1 0.2];
root = bisection(f,-0.1,0.2);
root
```

MATLAB Output:



4) This problem consists of us understanding the definition of Big-*O* and applying it to the numerical analysis ε symbol which is very important. This one is pretty straight forward, I just followed the rules given to us in problem and applied my own test numbers to the positive constants η, K, p, and C. The work for each problem is below.

a) $\sin(\varepsilon) = O(1)$ as $\varepsilon \to 0$

$f(\varepsilon) = \sin(\varepsilon)$
$\psi(\varepsilon) = 1$

let $\eta = 2$
$|\ \varepsilon\ | < 2$
$|\ 10^{-7}\ | < 2$ ✓

let $K = 10$
$|\ \sin(\varepsilon)\ | \leq K(1)$
$|\ 1.75 * 10^{-9}\ | \leq 10$ ✓

TRUE

b) $\sin(\varepsilon) = O(\varepsilon)$ as $\varepsilon \to 0$

$f(\varepsilon) = \sin(\varepsilon)$
$\psi(\varepsilon) = 1$

let $\eta = 2$
$|\ \varepsilon\ | < \eta$
$|\ 10^{-7}\ | < 2$ ✓

let $K = 10$
$|\ \sin(\varepsilon)\ | \leq K(\varepsilon)$
$|\ 1.75 * 10^{-9}\ | \leq 1 * 10^{-6}$ ✓

TRUE

c) $\sin(\varepsilon) = O(\varepsilon^2)$ as $\varepsilon \to 0$

$f(\varepsilon) = \sin(\varepsilon)$
$\psi(\varepsilon) = \varepsilon^2$

let $\eta = 2$
$| \varepsilon | < 2$
$| 10^{-7} | < 2$ ✓

let $K = 1{,}000{,}000$
$| \sin(\varepsilon) | \leq K(\varepsilon^2)$
$| 1.75 * 10^{-9} | \leq 1 * 10^{-8}$ ✓

TRUE (but only if $K > 175{,}000$)

d) $100x = O(x^{1.1})$ as $x \to \infty$

$g(x) = 100x$
$\phi(x) = x^{1.1}$

let $p = 2$, $x = 10$
$x < p$
$10 < 2$ ✓

let $C = 80$
$| 100x | \leq C(x^{1.1})$
$1000 \leq 1000.7$ ✓

TRUE (but only if $C > 79.43$)

5) For the fifth problem, we were given a function that multiplies two nonzero real numbers, and we must show that the algorithm is backwards stable. This was based off the example that Professor Lau had shown us in the narrations of the cond module. The work for this is shown below. The perturbation of the inputs is ***uniquely*** formed based on the variables. For example, the perturbation of the x1 is different from x2 because both perturbations have itself in them and not the other variables or share a common variable.

Proof:

On the next page…

$$y = h(x_1, x_2) = x_1 \cdot x_2$$

$$y_A = h_A(x_1, x_2) = fl(x_1) \odot f(x_2)$$
$$= x_1(1+\alpha) \odot x_2(1+\beta)$$
$$= x_1 \cdot x_2 (1+\alpha)(1+\beta)(1+\gamma)$$
$$= \left| \frac{x_1 \cdot x_2 (1+\alpha)(1+\beta)(1+\gamma) - x_1 \cdot x_2}{x_1 \cdot x_2} \right|$$
$$= |(1+\alpha)(1+\beta)(1+\gamma) - 1| = O(\varepsilon_{mach})$$

$$\downarrow$$

$$\underline{\delta x_1} = x_1 [(1+\alpha)(1+\beta)(1+\gamma) - 1]$$
$$\underline{\delta x_2} = x_2 [(1+\alpha)(1+\beta)(1+\gamma) - 1]$$

$$h_A x_1 = x_2 \cdot x_1 (1+\alpha)(1+\beta)(1+\gamma)$$
$$= x_2 [x_1(1+\alpha)(1+\beta)(1+\gamma) - x_1 + x_1]$$
$$= x_2 (x_1 + \delta x_1)$$
$$\underline{= h(x_1 + \delta x_1)}$$

$$h_A x_2 = x_1 \cdot x_2 (1+\alpha)(1+\beta)(1+\gamma)$$
$$= x_1 [x_2(1+\alpha)(1+\beta)(1+\gamma) - x_2 + x_2]$$
$$= x_1 (x_2 + \delta x_2)$$
$$\underline{= h(x_2 + \delta x_2)}$$

$$\left| \frac{\delta x_1}{x_1} \right| = \left| \frac{\cancel{x_1} [(1+\alpha)(1+\beta)(1+\gamma) - 1]}{\cancel{x_1}} \right|$$
$$= (1+\alpha)(1+\beta)(1+\gamma) - 1$$

$$O(\varepsilon_{mach}) = (1+\alpha)(1+\beta)(1+\gamma) - 1 \quad \boxed{\checkmark}$$

$$\left| \frac{\delta x_2}{x_2} \right| = \left| \frac{\cancel{x_2} [(1+\alpha)(1+\beta)(1+\gamma) - 1]}{\cancel{x_2}} \right|$$
$$= (1+\alpha)(1+\beta)(1+\gamma) - 1$$

$$O(\varepsilon_{mach}) = (1+\alpha)(1+\beta)(1+\gamma) - 1 \quad \boxed{\checkmark}$$

$$h_A(x_1, x_2) = h(x_1 + \delta x_1, x_2 + \delta x_2)$$

$$h_A x_1 = h(x_1 + \delta x_1) \qquad h_A x_2 = (x_2 + \delta x_2)$$

$$h(x_1 + \delta x_1, x_2 + \delta x_2) = h(x_1 + \delta x_1, x_2 + \delta x_2) \quad \boxed{\checkmark}$$