Damian Franco

CS 375 – 001

Homework #2:

1. The floating-point representation of 100.2 is…

100 – (1100100)2

0.2 – ()2

* 1. = (1100100. )2

V = (-1)2(1.b1….b52) \* 2F-1023

True exponent = 6

F = 1029

= (1027)10 + (2)10

= (10000000011)2 + (10)2

= (10000000101)2

fl(100.2) = (1.001 0001 1001 1001 1001 1001 1001

1001 1001 1001 1001 1001 1001 1) \* 26

Hex format: 0x40590ccccccccccd

decRep = (1.0010)2 \* 2-53 \* 26

= (1.0010)2 \* 2-47

= (0. )2 \* 2-43

= (0.2)10 \* 2-43

* 2-52 \* 26 = 2-46

fl(100.2) = 100.2 – 0.2 \* 2-43 + 2-46

= 100.2 + 0.1 \* 2-46

* εmach = 2-52

|fl(100.2) – 100.2| / |100.2| < εmach

|100.2 + 0.1 \* 2-46 – 100.2| / 100.2 < εmach

|0.1 \* 2-46 | / 100.2 < εmach

11 / 10020 \* 2-46 < 2-52

1.56 \* 10-17 < 2.22 \* 10-17 ✔

Finding the floating-point representation is very important because that can give us some insight on how much of and error or accurate our computing mathematical operations could be. The first thing when converting a decimal value to do is to convert both that whole number and decimal number to binary. Once we have them in binary, the next step is to put those two binary strings together and plug it in the V equation to find the floating point. From there the floating point should be relatively easy to find. When calculation whether fl(100.2) is less than the machine epsilon, there must be some steps taken there to get the true value of fl(100.2). Once the value of fl(100.2) is found, then when compared to 2^-52, the value for the floating-point representation is smaller than the value of the machine epsilon.

1. Compute the roots of x2 + 201x - 10-12 = 0

The most accurate way to compute the two roots of the equation above is to use the “Citardauq Formula” which is a form of the quadratic formula that will not suffer serious loss of precision. Floating point precision will give slightly wrong results with the quadratic formula.

Source Code (*HW2\_2.m*):

1 % Variable setup.

2 a = 1;

3 b = 201;

4 c = -10^-12;

5

6 % Aproximatation of two roots with precision errors.

7 err1 = (-b + sqrt(b^2 - 4\*a\*c)) / (2\*a);

8 err2 = (-b - sqrt(b^2 - 4\*a\*c)) / (2\*a);

9 err1

10 err2

11

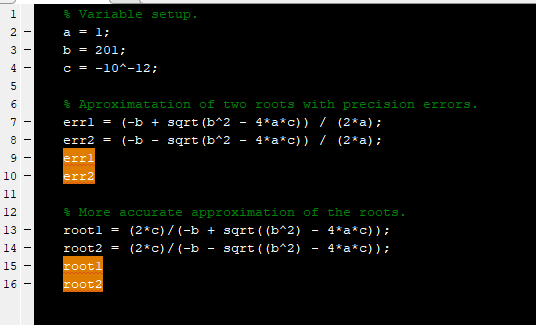
12 % More accurate approximation of the roots.

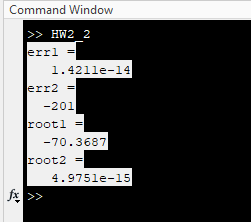
13 root1 = (2\*c)/(-b + sqrt((b^2) - 4\*a\*c));

14 root2 = (2\*c)/(-b - sqrt((b^2) - 4\*a\*c));

15 root1

16 root2



Output from program:

% Roots with errors.

err1 =

1.4211e-14

err2 =

-201

% Roots without errors.

root1 =

-70.3687

root2 =

4.9751e-15

1. For problem #3, the main point for this problem is to find out precision of certain equations compared to other. We were tasked with creating a function that takes in a scale for the zeroth order cylindrical Bessel function. The function should output a vector of values that can be related to the accuracy of that function call, then we had to compare them to MATLAB built in Bessel function. In my finding, I noticed that my numbers are all over the place and frankly does not look too correct so I am hoping for some partial credit on this one. But regardless, I definitely took notice on how some function and equations have a huge impact on the precision and accuracy which could be good or bad impact.

3a)

Source code (*ApproxBesselJ0*.m)

1 %

2 % Function: ApproxBesselJ0(x)

3 % This function will approximate the based of the cylindrical

4 % Bessel function Maclaurin series.

5 %

6 function J0 = ApproxBesselJ0(x)

7 out = [];

8 N = 1e-8;

9 term = 0;

10 J0 = term;

11 k = 0

12 while(20 > k)

13 term = (-1)^k \* (1/factorial(k))^2 \* (x/2)^(2\*k);

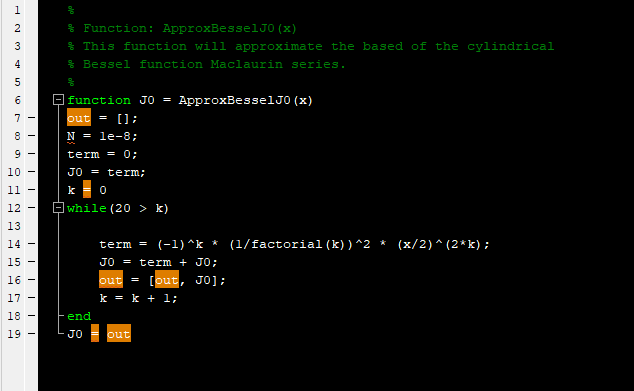
14 J0 = term + J0;

15 out = [out, J0];

16 k = k + 1;

17 end

18 J0 = out



3b) Source code (HW\_3\_Table.m)

1 %

2 % Script: Prob2table

3 % Makes the table for problem 3 on HW 2

4 %

5 function t = HW\_3\_Table(f1, f2, x)

6 fid=fopen('HW3\_Prob3.txt','w');

7 fprintf(fid,'---------------------------------------------------\n');

8 fprintf(fid,'| x | My J0 | MATLABs J0 |\n');

9 fprintf(fid,'---------------------------------------------------\n');

10 for k = 1 : 20

11 re1 = f1(k);

12 re2 = f2(k);

13 fprintf(fid,'| %1.0f | %1.14f | %1.14f |\n', x, re1, re2);

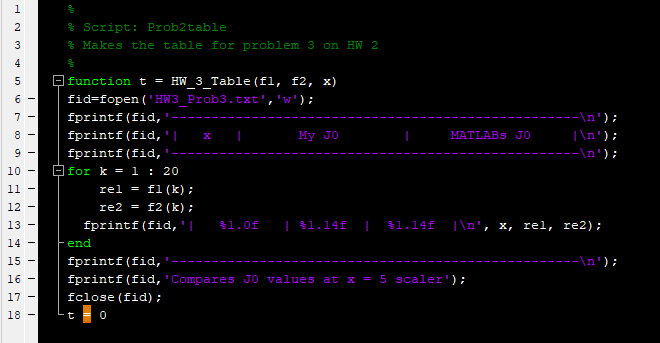
14 end

15 fprintf(fid,'---------------------------------------------------\n');

16 fprintf(fid,'Compares J0 values at x = 5 scaler');

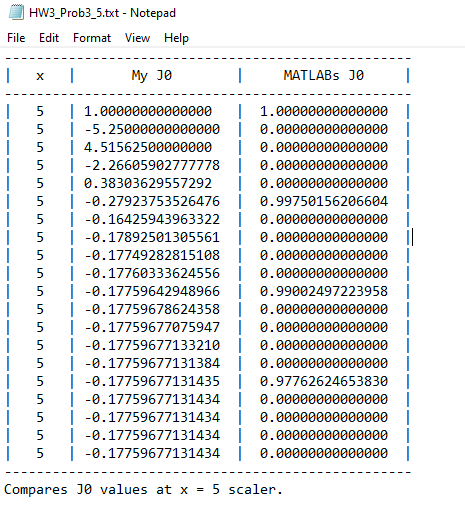
17 fclose(fid);

18 t=0

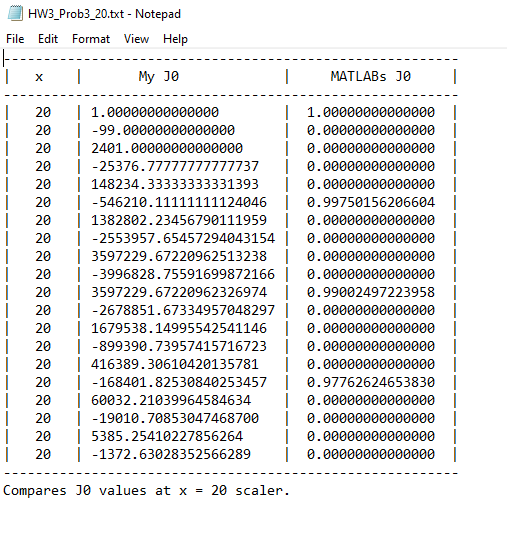


Text (table) files:

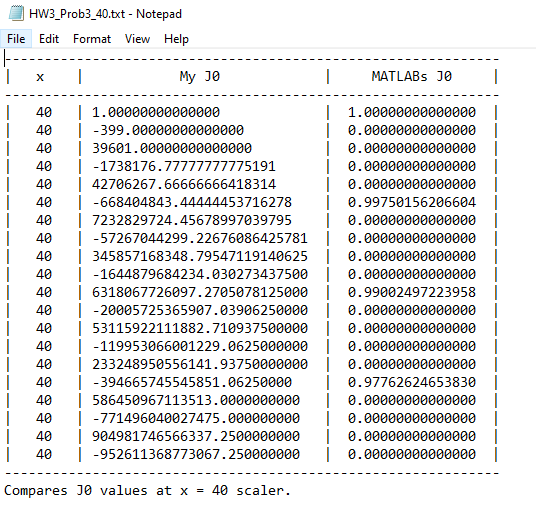
* Scaler Multiplier = 5 (HW3\_Prob3\_5.txt)



* Scaler Multiplier = 20 (HW3\_Prob3\_20)



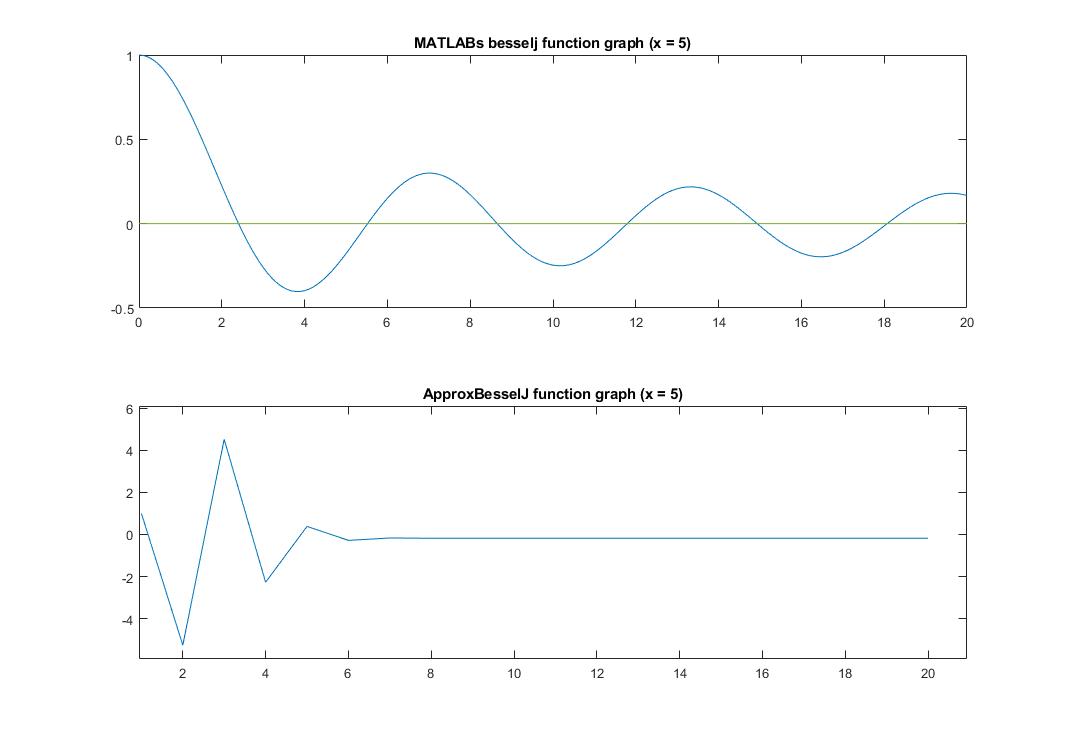
* Scaler Multiplier = 40 (HW3\_Prob3\_40)



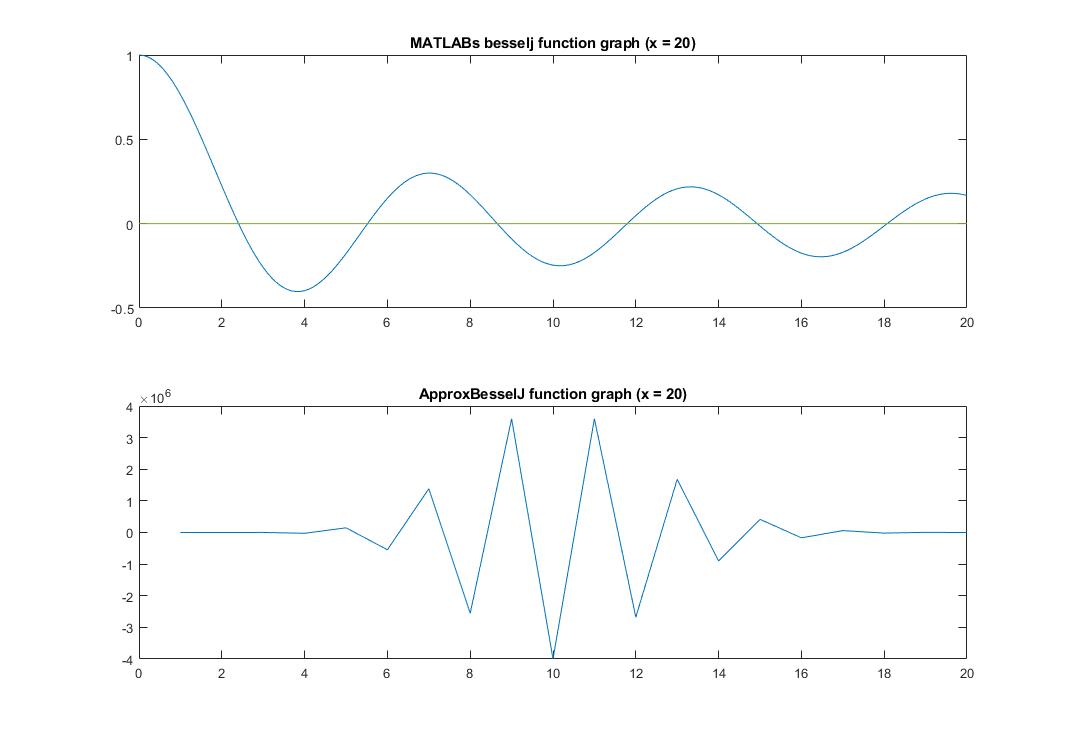
3c) We can see that the loss of precision is very prevalent when using some function and equations in numerical computing in part (b). The values were vastly different and we can see that the effective ness of some equations and functions is much better than others.

Graphs of findings:

* For scaler = 5 (*Prob3C\_5*.m)



* For scaler = 5 (*Prob3C\_20*.m)



* For scaler = 5 (*Prob3C\_5*.m)

