Math 5610 – Homework 1

Andrew Sheridan

September 2016

Problem 1

Implement an algorithm to compute the machine epsilon for your computer.

```
double doubleEpsilon = 1.0;
int exponent = 0;

while(doubleEpsilon + 1 != 1){
    exponent--;
    doubleEpsilon /= 2;
}
std::cout << "Epsilon w/ double: " << doubleEpsilon << std::endl;
std::cout << "Exponent: " << exponent << std::endl;

float singleEpsilon = 1.0;
exponent = 0;

while(singleEpsilon + 1 != 1){
    exponent--;
    singleEpsilon /= 2;
}
std::cout << "Epsilon w/ float: " << singleEpsilon << std::endl;
std::cout << "Exponent: " << exponent << std::endl;
```

CONSOLE OUTPUT

```
Epsilon w/ double: 1.11022e-016
Exponent: -53
Epsilon w/ float: 5.96046e-008
Exponent: -24
```

Problem 2

Approximate the derivative of the function $f(x) = e^{-2x}$ evaluated at $x_0 = 0.5$. Observe similarities and differences by comparing your graph against that in figure 1.3.

Example 1.3 uses the formula $|f'(x_0) - \frac{f(x_0+h)-f(x_0)}{h}| \approx \frac{h}{2}|f''(x_0)|$ to estimate the discretization error. For our $f(x)$ we have the derivative $f'(x) = -2e^{-2x}$, and the second derivative $f''(x) = 4e^{-2x}$. If we substitute these into the formula from the example, we get

$$| -2e^{-2x_0} - \frac{e^{-2(x_0-h)} - e^{-2x_0}}{h}| \approx \frac{h}{2}|4e^{-2x_0}|$$

We are also given the value of x_0, which we can insert into the formula above to get

$$| -2e^{-1} - \frac{e^{-1-2h} - e^{-1}}{h}| \approx \frac{h}{2}|4e^{-1}|$$

which then simplifies to

$$|\frac{-2}{e} - \frac{e^{-2h-1}}{h} - \frac{1}{eh}| \approx 2h|\frac{1}{e}|$$

We now have two formulas we can plug into C++ to get some sample data. Below is the code used to generate our results as a text file.

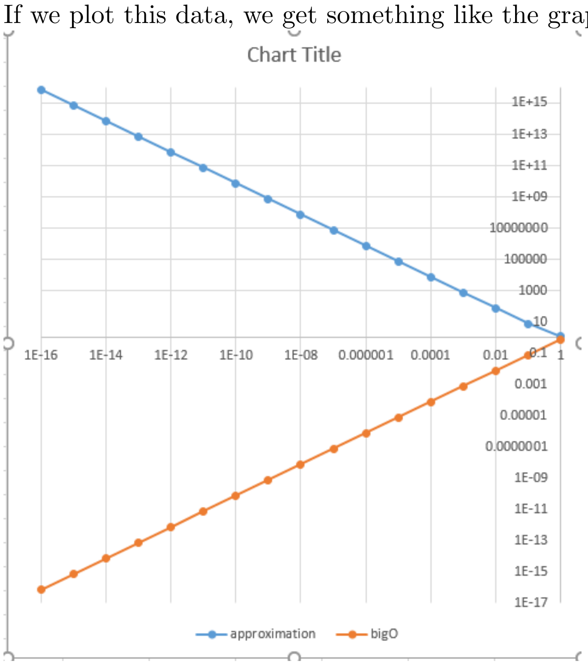
```
std::ofstream myfile;
myfile.open("problem2.txt");
double stoppingPoint = pow(10, -16);

std::cout << "h \t" << "approximation \t" << "bigO" << std::endl;
myfile << "h \t" << "approximation \t" << "bigO" << std::endl;

for (double h = 1.0; h > stoppingPoint; h *= 0.1) {
    double approximation = (-2 / exp(1)) - (exp(-1 - 2*h) / h) - (1 / (exp(1)*h));
    approximation = abs(approximation);

    double bigO = 2 * h * abs(exp(-1));

    std::cout << h << "\t" << approximation << "\t" << bigO << std::endl;
    myfile << h << "\t" << approximation << "\t" << bigO << std::endl;
}
myfile.close();
```

This code outputs the following results:

```
h      approximation  bigO
1     1.15343         0.735759
```

```
0.1     7.4265          0.0735759
0.01 73.5832          0.00735759
0.001 735.76          0.000735759
0.0001 7357.59       7.35759e-05
1e-05 73575.9       7.35759e-06
1e-06 735759          7.35759e-07
1e-07 7.35759e+06     7.35759e-08
1e-08 7.35759e+07     7.35759e-09
1e-09 7.35759e+08     7.35759e-10
1e-10 7.35759e+09     7.35759e-11
1e-11 7.35759e+10     7.35759e-12
1e-12 7.35759e+11     7.35759e-13
1e-13 7.35759e+12     7.35759e-14
1e-14 7.35759e+13     7.35759e-15
1e-15 7.35759e+14     7.35759e-16
1e-16 7.35759e+15     7.35759e-17
```

If we plot this data, we get something like the graph shown.



This graph is more than a bit interesting, as we can see that the resulting lines are near perfect reflections of one another across the x-axis.

Problem 3

Carry out derivation using the expression $\frac{f(x_0+h)-f(x_0-h)}{2h}$. Show that the error is $O(h^2)$. More precisely, the leading term of the error is $-\frac{h^2}{3}f'''(x_0)$ when $f'''(x_0) \neq 0$.

We are given the expression $\frac{f(x_0+h)-f(x_0-h)}{2h}$ to carry out. Both $f(x_0+h)$ and $f(x_0-h)$ can be expanded as Taylor Series expansions as follows:

$$f(x_0+h) = f(x_0) + hf'(x_0) + \frac{h^2}{2}f''(x_0) + \frac{h^3}{6}f'''(x_0) + ...$$
$$f(x_0-h) = f(x_0) - hf'(x_0) + \frac{h^2}{2}f''(x_0) - \frac{h^3}{6}f'''(x_0) + ...$$

We can then subtract $f(x_0-h)$ from $f(x_0+h)$ which yields

$$f(x_0+h) - f(x_0-h) = 2hf'(x_0) + \frac{2h^3}{6}f'''(x_0) + ...$$

If we then truncate the Taylor Series expansion to only contain these two statements, we have

$$f(x_0+h) - f(x_0-h) \approx 2hf'(x_0) + \frac{2h^3}{6}f'''(x_0)$$

After some simplification, we can isolate $f'(x_0)$, which gives the expression

$$f'(x_0) \approx \frac{f(x_0+h) - f(x_0-h)}{2h} - \frac{h^2}{6}f'''(x_0)$$

We can see here that the error is $\frac{h^2}{6}f'''(x_0)$, or $O(h^2)$, as desired (given that $f'''(x_0) \neq 0$).

Problem 4

Carry out similar calculations to those of Example 1.3 using your approximation from Exercise 2. Observe similarities and differences by comparing your graph against that in figure 1.3.

We start with the formula $f'(x_0) \approx \frac{f(x_0+h)-f(x_0-h)}{2h} - \frac{h^2}{6}f'''(x_0)$. We are told that $f(x) = sin(x)$ and that $x_0 = 1.2$. If we plug these values into our original formula and do some rearranging, we get

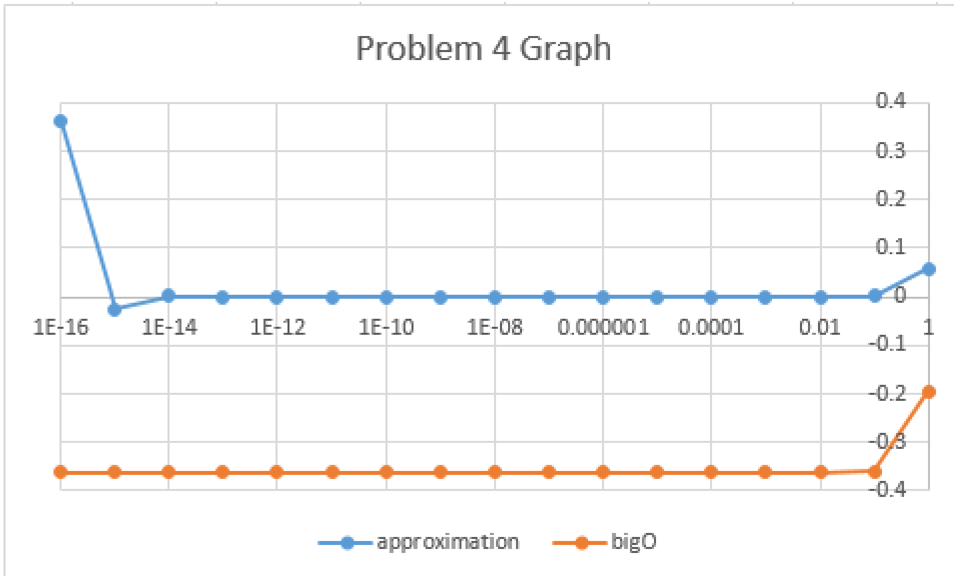
$$cos(1.2) - \frac{sin(1.2+h) - sin(1.2-h)}{2h} \approx \frac{h^2}{6} - cos(1.2)$$

We can use C++ to get some sample data for various value h. Our code is functionally similar to that written in Problem 2. We simply substitute the actual approximation and bigO to be the following:

```
double approx = cos(1.2) - (sin(1.2+h) - sin(1.2-h))/(2*h);
double bigO = pow(h, 2) / 6 - cos(1.2);
```

This yields the following results:

h	approximation	bigO
1	0.0574442	-0.195691
0.1	0.000603628	-0.360691
0.01	6.03927e-06	-0.362341
0.001	6.0393e-08	-0.362358
0.0001	6.03619e-10	-0.362358
1e-05	2.43311e-12	-0.362358
1e-06	7.98417e-12	-0.362358
1e-07	1.19007e-10	-0.362358
1e-08	-4.36105e-10	-0.362358
1e-09	-4.36105e-10	-0.362358
1e-10	3.88142e-07	-0.362358
1e-11	-2.38742e-06	-0.362358
1e-12	-7.45519e-05	-0.362358
1e-13	-0.000130063	-0.362358
1e-14	0.00153527	-0.362358
1e-15	-0.0262203	-0.362358
1e-16	0.362358	-0.362358


Problem 4 Graph
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
1E-16
1E-14
1E-12
1E-10
1E-08
0.000001
0.0001
0.01
1
approximation
bigO

Problem 5

Following Example 1.5, assess the conditioning of the problem of evaluating

$$g(x) = tanh(cx) = \frac{exp(cx) - exp(-cx)}{exp(cx) + exp(-cx)}$$

near $x = 0$ as the positive parameter c grows.

To start, we will find the derivative of $g(x)$, which is $g'(x) = c \times sech^2(cx)$. If we plug in 0 as x, we then are left with $g'(0) = c \times sech^2(0) = c$. As c grows, the result will become large, which means that this problem is ill-conditioned.

Problem 6

A. Derive a formula for approximately computing these integrals based on evaluating y_{n-1} given y_n

To derive the formula which evaluates y_{n-1} given y_n, we can start with the formula given in the book of $y_n + 10y_{n-1} = \frac{1}{n}$. This can be rearranged and give us the function $y_{n-1} = \frac{1}{10}(\frac{1}{n} - y_n)$. We can then rewrite this for convenience in our upcoming recursive expansion to be $y_n = \frac{1}{10}(\frac{1}{n+1} - y_{n+1})$

We can now start to expand this function as follows:

$$y_n = \frac{1}{10}(\frac{1}{n+1} - \frac{1}{10}(\frac{1}{n+2} - y_{n+2}))$$

$$y_n = \frac{1}{10}(\frac{1}{n+1} - \frac{1}{10}(\frac{1}{n+2} - \frac{1}{10}(\frac{1}{n+3} - y_{n+3})))$$

$$...$$

$$y_n = \frac{1}{10}\frac{1}{n+1} - \frac{1}{10^2}\frac{1}{n+2} + \frac{1}{10^3}\frac{1}{n+3} - ... + \frac{(-1)^{k-1}}{10^k}(\frac{1}{n+k} - y_{n+k})$$

In this formula, our error is being divided by 10 with each iteration, rather than being multiplied by 10.

B. Show that for any given value $\epsilon > 0$ and positive integer n_0, there exists an integer $n_1 \geq n_0$ such that taking $y_{n_1} = 0$ as a starting value will produce integral evaluations y_n with an absolute error smaller than ϵ for all $0 < n \leq n_0$.

Due to shortage of time, I'm unable to fully provide a proof here. All I know is that, given the formula we created in part A, $y_n \leq \frac{1}{10(n+1)}$ for all n.

C. Explain why your algorithm is stable.

Because for any $\bar{x}$ close to x, the result will be close to x because the error ϵ converges.

D. Write a function that computes the value of y_{20} within an absolute error of at most 10^{-5}. Explain how you choose n_1 in this case.

NOT COMPLETED

Note: After being unable to come to anything close to a solution to this problem on my own, and putting a good deal of effort into the problem, I found help at http://math.stackexchange.com/questions/496912/error-accumulation-in-an-approximating-numerical-algorithm-for-y-n-int-01, and a comment pushed me in the right direction.