Lab 2

To: John Salmon
From: Caelan Osman
Date: January 30, 2020
Subject: Machine Epsilon
Enclosures: (a) Source Code
(b) Sample Input-Output

The purpose of this lab is to determine the single and double values of machine epsilon for my computer. We can use the equation $\frac{1}{2^n}$ to estimate machine epsilon, in addition to this we will be writing a computer program in C++ to help us experimentally find machine epsilon to three significant digits. The procedure used followed the class handout entitled "Laboratory Assignment 2." Look at enclosure (a) to see our source code, the description below explains what we did.

To experimentally determine the machine epsilon for float and double we start out with small numbers such as 1e-3 and continue until adding such a small number does not change the value of 1.0. We repeat the process making the number larger and then smaller closing in and limiting on our machine epsilon for single and double.

From these results we found that as the equation $\frac{1}{2^n}$ estimates, the machine epsilon to three significant digits is for float: 1.19e-7 and and for double: 2.22e-16. In the output of the code below you can see the estimate value for machine epsilon for both double and single. We also output the estimates of the digits of precision. For single it is 6, and for double it is 15. As we can see from the outputs both of the experimentally determined machine epsilons are higher than that of the estimate, however because we only got the experimental to three significant digits, the estimated will be of higher accuracy.

We can apply the idea of round-off errors and machine epsilons to other numbers as well. If we look at 50.5, and use a float for representation, the round-off value can be represented by $\frac{2^p}{2^n}$ where $p$represents the highest exponent in the IEEE representation of 50 and n represents the bit number in the mantissa not including the shadow bit. So we see here, with 50.5 the round off error would be $\frac{2^5}{2^23} \approx 3.815e^{-}6$. As stated in class the machine epsilon, E_m, is defined to be the smallest number, which, when added to 1.0 gives something other than 1.0.

From our equation above we can see why the round off error gets larger as the number itself increases. 3000.7 would for example have 2^11 in the numerator. Essentially you are multiplying the machine epsilon by a larger and larger number as your nominal value increases giving you a larger and larger round off error.

Enclosure (a)

Here is our source code in C++

```
//-------------------------------------------------------------
//  File:     ME273lab2.cpp
//  By:      Caelan Osman
//  Date:  30 January 2020
// This program estimates the single and double precision machine
// epsilons experimentally and with the equation e = 2^(-n) and
// n=-log10(e)
//-------------------------------------------------------------
#include <iostream> ///input-output
#include <iomanip> ///input-ouput manipulators
#include <ios>
#include <cmath> ///access the math library

using namespace std; ///the namespace where cout and iostream live

int main()
{
typedef unsigned int uint; // just messing around with typedef not necessary for this pr
float x = 1.0;
float dx = 0.0;
uint flag = 0;
cout.setf(ios::fixed);  /// Fixed point (not scientific)
cout.setf(ios::showpoint); /// Print trailing zeros
cout.precision(22);  /// Number of digits after decimal
long double singleEstimate = 1 / (pow(2.0, 23)); ///comparing these to what we found
long double doubleEstimate = 1 / (pow(2.0, 52));
uint singlePrecision = -log10(singleEstimate); ///These are our precisions
uint doublePrecision = -log10(doubleEstimate);

do
{
cout << "Enter a small value: ";
cin >> dx; ///input our test machine epsilon
x += dx;
cout << x << "  " << dx << endl;
cout << "If you want to exit enter 1:  ";
cin >> flag; ///user inputs if they want to contiue
x = 1.0; ///reset x to 1.0 every single time so we can check
} while(flag != 1);

cout << "Single machine epsilon experimental: " << 2 * dx << endl;
cout << "Machine Epsilon single estimate: " << singleEstimate << endl;
cout << "Estimate digits of precision: " << singlePrecision << endl;
double xd = 1.0;
```

```
double dxd;; /// making new variables of type double
do
{
cout << "Enter a small value: ";
cin >> dxd; ///input our test machine epsilon
xd += dxd;
cout << xd << "  " << dxd << endl;
cout << "If you want to exit enter 1:  ";
cin >> flag; ///user inputs if they want to contiue
xd = 1.0; ///reset x to 1.0 every single time so we can check
} while (flag != 1);

cout << "Double machine epsilon experimental: " << 2 * dxd << endl;
cout << "Machine epsilon double estimate: " << doubleEstimate << endl;
cout << "Estimate digits of precision: " << doublePrecision << endl;
}
```

Enclosure (b)

Here is an image of the input & output

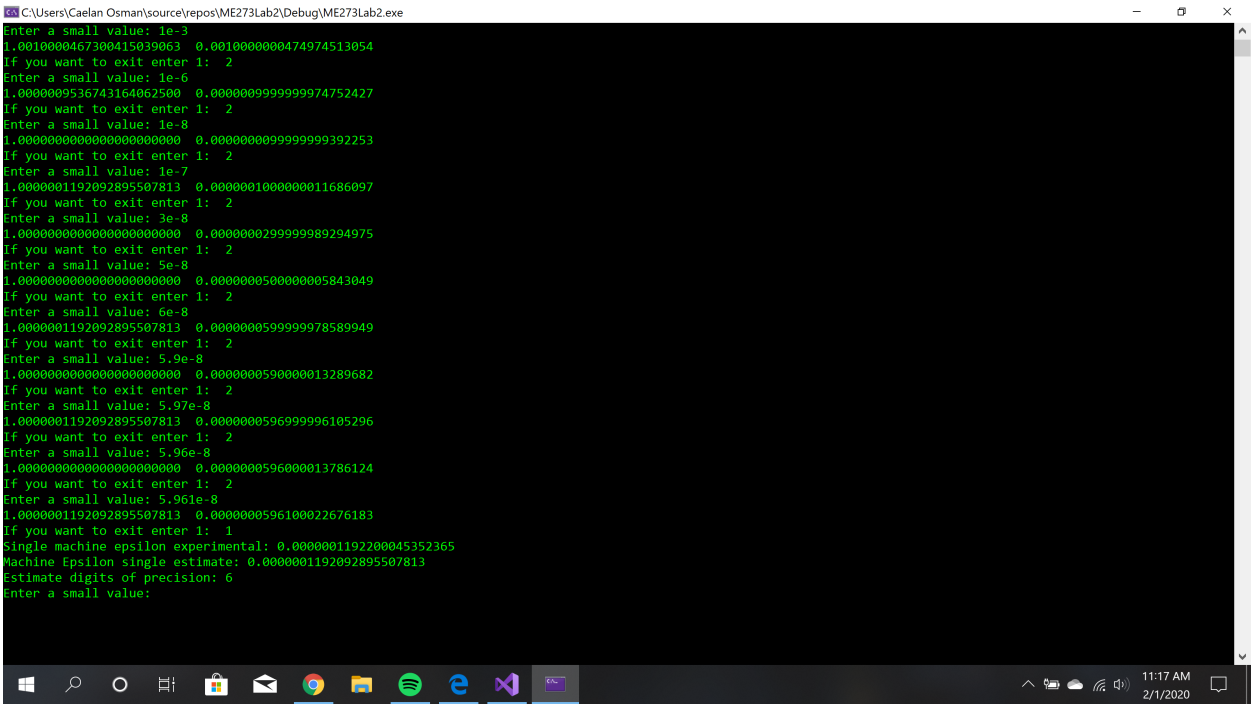

Figure 1: The input and output for experimentally finding the single machine epsilon

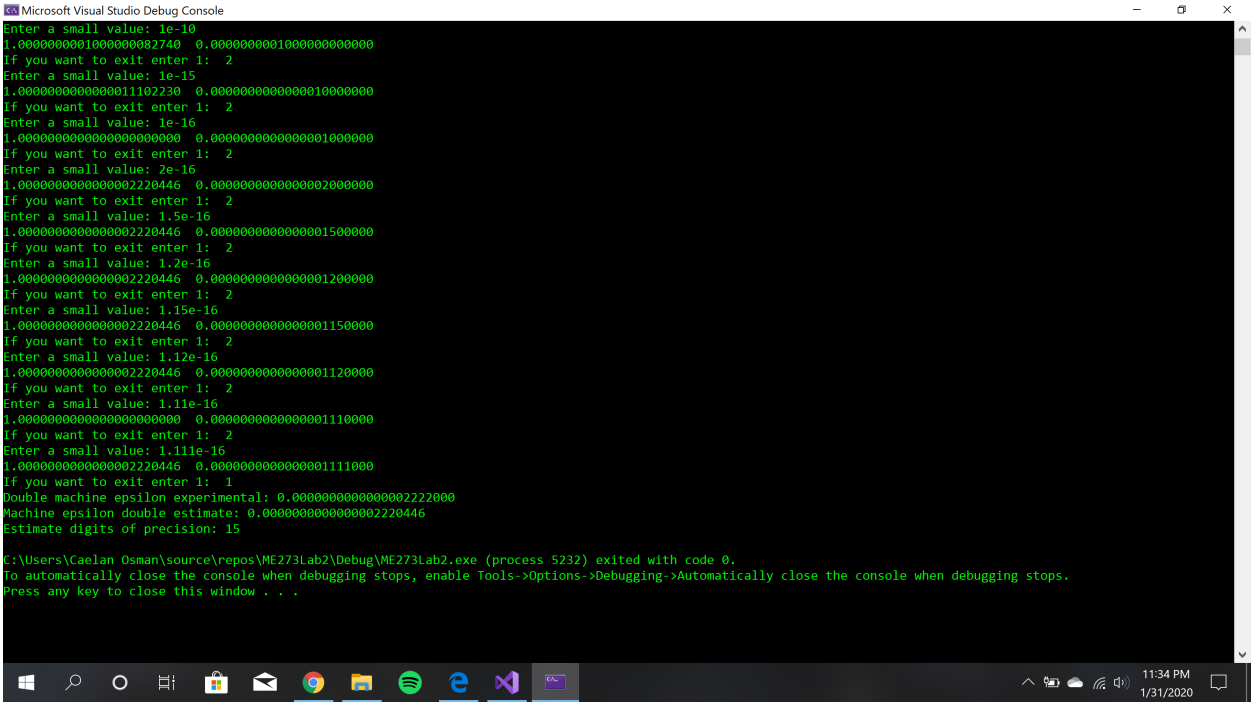

Figure 2: The input and output for experimentally finding the double machine epsilon