Jaime Frade
Computational Finance: Dr. Kopriva
Homework #1

0. **Executive Summary**

There are several problems being solved in this homework assignment, centralized around one common theme, how different computers represent and perform operations on numerical values. For future calculations, I must find what standard my computer(s) was given. I will check if my computer uses the IEE 754 floating point standard system by comparing the machine epsilon, the largest, and smallest floating point numbers. After writing a reusable class in C++ that returns the correct floating point constant, I found that my computer uses the IEEE standard.

1. **Statement of Problem**

Write a program, in which I will create a reusable class that will return the constants for the machine epsilon, the largest and smallest floating point number that a computer uses. From here, I will determine if the computations of my computer use IEEE 754 standard. (For program see appendix)

2. **Description of the Mathematics**

In order to determine how significant the error in computations due to rounding is, I will need to determine the relative error due to rounding. Assuming rounding error, because this is at most $\frac{1}{2}$ of chopping error, the upper bound on this error, the machine epsilon, ϵ, then

$$\epsilon = \frac{\beta^{1-t}}{2} \tag{1}$$

where β be the base, ($\beta \in Z^{++}$), and t gives the number of digits of M, the **matissa**, is.

To check if a computer uses the IEEE standard in single and double precision, I need to determine the upper and lower bounds of the floating point number, as well as, the machine epsilon.

Assuming IEEE, (β=2 and $t = 23$) and from (1), for the floating point numbers in single precision:

$$FL(x) = (-1)^s \, 2^{E-127} 1.\mathrm{M} \tag{2}$$

where the sign ($s = 0$ or 1) determines the sign and $0 < E < 255$. In double precision,

$$FL(x) = (-1)^s \, 2^{E-1023} 1.\mathrm{M} \tag{3}$$

From IEEE 754 floating point standard, the values that are inputed into (2) and (3), for a single precision with 32-bit and double precision having 64-bit storage are as follows:

	SINGLE	DOUBLE
MANTISSA (M)	23	52
EXPONENT (E)	8	11
SIGN (s)	1	1

- The smallest number in the range in IEEE for single precision, from lecture, is determined $E = 1$ and M=0, thus from (2), $\Longrightarrow 2^{-126} \approx 1.18 \,\mathrm{x}\, 10^{-38}$
- The largest number in single precision, $E = 254$ and M=1, thus from (2), $FL(x) = 2^{127}\,(2 - 2^{-23}) \approx 3.4 \,\mathrm{x}\, 10^{38}$
- The machine epsilon is computed for using the upper bound on the relative error (1), $\epsilon = 2^{-23} \approx 1.19 \,\mathrm{x}\, 10^{-7}$

There are exceptions, which IEEE did account for, yet above are not affected.

3. **Description of the Algorithm**

Besides the numerical methods illustrated in the previous section, there was no other calculations used. In the computer program, there are built in commands that store the numbers that I need, so I created a reusable class that returns these numbers.

4. **Results**

From the program, I found the following which can be compared to the IEEE standard.

	SINGLE	DOUBLE
Machine Epsilon	1.19209290E-07	2.220446049E-16
Largest Floating pt Number	3.40282347E+38	1.797693134E+308
Smallest Floating pt Number	1.17549435E-38	2.225073858E-308
Significant Digits (SIGFIGS)	6	15

5. **Conclusion**

The values I obtained are the same values from the IEEE standard to six significant digits for single precision and fifteen significant digits for double precision. Therefore, I conclude that my machine's Intel Pentium processor uses the IEEE standard.

2a. **i.** Determine the total number of normalized floating point numbers that can be represented in the IEEE single precision standard.

From lecture, the finite number system of floating point representation of a number is as follows:

$$FL(x) = \sum_{k=1}^{t} \frac{d_k}{\beta^k} \beta^E \tag{4}$$

where the β =base, $0 \leq d_k \leq \beta - 1$, and E = exponent.

In IEEE, the following values are used in (4) in single precision, $\longrightarrow \beta = 2$, $0 < E < 255$, and the length of the **matissa**, $t = 24$, also takes in to account the first place for the hidden bit.

The total number of normalized floating point numbers that can be represented is computed as

$$2(254 - 1 + 1)(2 - 1)2^2 3 + 1 = 4261412865 \tag{5}$$

ii. What is the smallest spacing between two floating point numbers? What is the largest spacing?

- The smallest spacing between two adjancet points is determined by taking the differece between the smallest and the next smallest numbers:
  (note: the next smallest is calculated by changing the last place in the matissa.) $(2^{-23})(2^{-126}) = 1.4013E - 45$

The largest spacing between two adjancet points is determined bytaking the differece between the largest and the next largest numbers:
(note: the next largest is calculated by changing the last place in the matissa.) $(2^{-23})(2^{127}) = 2.02824E{+}31$

iii. How do the *relative* spacing for these two results above compare?

Need to calculate the relative spacing between the largest and smallest spacing.

The smallest relative spacing is bounded between the following intervals:

$$\left[\frac{\text{smallest spacing}}{(2 - 2^{-23})\, 2^{-126}}\,,\ \frac{\text{smallest spacing}}{2^{-126}}\right]$$

$$\left[\frac{2^{-23}}{(2 - 2^{-23})}\,,\ 2^{-23}\right]$$

$$\left[2^{-24}\,,\ 2^{-23}\right] \tag{6}$$

The largest relative spacing is bounded between the following intervals:

$$\left[\frac{\text{largest spacing}}{(2 - 2^{-23})\, 2^{127}}\,,\ \frac{\text{largest spacing}}{2^{127}}\right]$$

$$\left[\frac{2^{-23}}{(2 - 2^{-23})}\,,\ 2^{-23}\right]$$

$$\left[2^{-24}\,,\ 2^{-23}\right] \tag{7}$$

Comparing the results in (6) and (6), can see that the intervals are the same with whichever numbers chosen.

2b. What are the largest and smallest (nonzero) *absolute* errors due to rounding in the IEEE number system?

From lecture, in single precision, the largest and smallest *absolute* errors due to rounding is $\frac{1}{2}$ of the largest and smallest spacing.

Smallest *abs* error $= \dfrac{\text{smallest spacing}}{2} = 7.00649E{-}46$

Largest *abs* error $= \dfrac{\text{largest spacing}}{2} = 1.01412E{+}31$

3a. Show by example that floating point rounded or chopped arithmetic (unlike real arithmetic!) is not associative. That is show it is not necessarily true that $a + (b + c) = (a + b) + c$, if a, b, and c are floating point numbers.

Note:

- Associative law of addition: $(a + (b + c) = (a + b) + c)$
- Distributive law: $(a + b)c = ac + bc$
- Let $FL(x)$ be the floating representation for real number x.
- Assume the following for computation in the floating point system.
  $\longrightarrow \beta = 10, t = 3$, and rounding for **3a.** and **3b.**

PROOF: by counterexample

Let a=0.1111, b=0.1111, and c=0.6666

$$\begin{aligned} FL(a + FL(b + c)) &= FL(0.1111 + FL(0.1111 + 0.6666)) \\ &= FL(0.1111 + FL(0.7777)) \\ &= FL(0.1111 + 0.778) \\ &= 0.889 \end{aligned} \tag{8}$$

$$\begin{aligned} FL(FL(a + b) + c)) &= FL(FL(0.1111 + 0.1111) + 0.6666) \\ &= FL(FL(0.2222) + 0.6666)) \\ &= FL(0.222 + 0.6666) \\ &= FL(0.8886) \\ &= 0.8888 \end{aligned} \tag{9}$$

$$Q.E.D \tag{10}$$

3b. Is floating point arthmetic distributive?

PROOF: by counterexample

Let a=2.2333, b=-2.2222, and c=1

$$\begin{aligned} FL(FL(a + b)c)) &= FL(FL(0.0111)1) \\ &= FL(0.0111) \\ &= 0.0111 \end{aligned} \tag{11}$$

$$\begin{aligned} FL(FL(ab) + FL(ac)) &= FL(FL(2.2333) + FL(-2.2222)) \\ &= FL(2.23 - 2.22)) \\ &= FL(0.01) \\ &= 0.0100 \end{aligned} \tag{12}$$

$$Q.E.D \tag{13}$$

From (10) and (13, the examples have illustrated how the associative and distributive law do not hold due to roundoff errors occuring at the place where the calculations are made.

```
//Floatpt.h
//Floatpt class definitions
//member functions in Floatpt.cpp
class Floatpt {
public:
double dbl_epsilon();
double dbl_max();
double dbl_min();
float flt_epsilon();
float flt_max() const;
float flt_min();
};


////////////////+++++++///////////////


//Floatpt.cpp
//member functions for class Floatpt

#include <iostream>
#include "float.h"
#include "Floatpt.h"
double Floatpt::dbl_epsilon()
{
return DBL_EPSILON;
}
double Floatpt::dbl_max()
{
return DBL_MAX;
}
double Floatpt::dbl_min()
{
return DBL_MIN;
}
float Floatpt::flt_epsilon()
{
return FLT_EPSILON;
}
float Floatpt::flt_max() const
{
return FLT_MAX;
}
float Floatpt::flt_min()
{
return FLT_MIN;
}

////////////////+++++++///////////////
```

```
//demonstrating Floatpt function
//compile this program with Floatpt.cpp

//include flt class defintions from Floatpt.h

#include "float.h"
#include "Floatpt.h"
#include <iostream>
using std::cout;
using std::endl;
int main()
{
Floatpt f;
double macheps, maxdbl, mindbl, flteps, maxflt, minflt;

macheps = f.dbl_epsilon();
cout << "The Machine Epsilon is " << macheps << endl; //remember this one!

cout << endl;

maxdbl = f.dbl_max();
cout << "The Maximum Double is " << maxdbl << endl ;
cout << endl;

mindbl = f.dbl_min();
cout << "The Minimum Double is " << mindbl << endl;
cout << endl;

flteps = f.flt_epsilon();
cout << "The Float Epsilon is " << flteps << endl ;
cout << endl;

maxflt = f.flt_max();
cout << "The Maximum Float is " << maxflt << endl ;
cout << endl;

minflt = f.flt_min();
cout << "The Minimum Float is " << minflt << endl ;
cout << endl;

return 0;
}
```