Roy Chan
chanroyc@sfu.ca
301202770

MACM 316 Assignment 1

Synopsis

The following report is an analysis on the *Floating Point Number System* with the use of the following code below.

```
1  for i=1:n
2      x=sqrt(x);
3  end
4
5  for i=1:n
6      x=x^2;
7  end
```

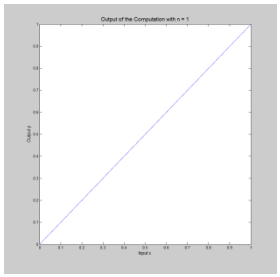
(i) Code for *Matlab*

Image (i): "Let $x \geq 0$ be an arbitrary number and n a nonnegative integer. In exact arithmetic, the following computation leaves x unchanged".

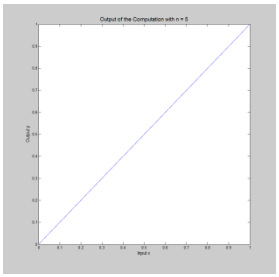
In *Floating Point Arithmetic*, however, there may be round-off errors that lead to inaccuracy of the computation. The square-root and exponential operations leave x as a floating point number. This report will investigate values of n and for x in the range $0 \leq x \leq 5$.

Discussion

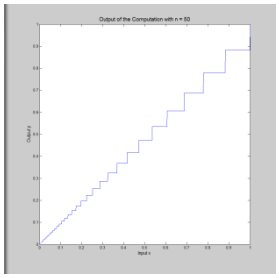
The graphs below are plotted results of the computation with size n to analyze the changes in error as n becomes increasingly large. (Scale of input x: 0 to 5, by increments of 0.1)



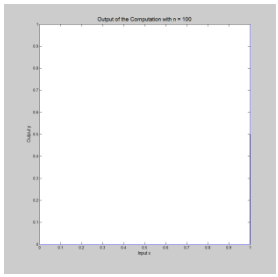
(ii) Plot with $n = 1$



(iii) Plot with $n = 5$



(iv) Plot with $n = 50$



(v) Plot with $n = 100$

(a) As we can see, from images (ii) - (iv), as n increases, the difference between inputs x and inputs y becomes increasingly apparent. Inputs y represents *Floating Point Numbers* and inputs x represents arbitrary integer(s). The relationship between the two inputs is the *Round-off Error* from chopping. Through numerous computations, *MATLAB* computes *Floating Point* arithmetic with 4-digit chopping.

(b) Machine Epsilon, or "ε" in *MATLAB* is 2.2204e-16 (computed in script). What this means is that MATLAB can only compute up to 15-16 digits of *Floating Points* with **precision**. The max error of n=1 is 1.0e-15, although the error is not physically visible, a single digit of precision could have possibly been chopped already, thus differing the exact arithmetic computation (based on the capabilities of *MATLAB*).

(c) As mentioned in (b), Machine Epsilon tells us that MATLAB can only compute up to 15-16 digits with precision. As MATLAB computes using 4-digit chopping, digits, or more precisely, precision is lost with any error larger than Machine Epsilon. Thus as n becomes larger and higher precision is needed, the limiting behaviour of n becomes relative to the number of digits the software can compute with precision which is explained by Machine Epsilon.

(d) In conclusion, a larger n requires higher precision, in other words, the ability to compute more decimal places. *MATLAB* utilizes 4-digit chopping, thus 11-12 digits are chopped, rendering the computations inaccurate on top of the bounds from Machine Epsilon which limit only 15-16 digits. When n nears infinity, *Finite Precision Arithmetic* can no longer compute with accuracy, as it is limited by Machine Epsilon.