ME 571 Homework 2

Gurpal Singh

March 5, 2017

Abstract

Numbers can be very tricky in the c language. A divergent and taylor series are explored using the float and double type. The results show that although accuracy is lost with single precision, it is a much faster way to run computations than using double precision. However, if one is examining a problem were precision is valuable, then double precision must be used which will cost more time.

1 Introduction

There are two sections to this report. In the first section a divergent series is explored using both single and double precision. In the second part, a taylor expansion of the exponential function is explored using single and double precision. As it will become clear later, single and double precision may represent numbers very differently and have different capabilities. There is not always one correct type to use. Rather, it is important to consider the task at hand and pick which one to use on a case by case basis.

2 Problem 1

In this problem the following divergent infinite series is analyzed. The corresponding source code file to this problem is named HW2.c. The results are also provided in table format.

$$\sum_{n=1}^{\infty} \frac{1}{n} \tag{1}$$

2.1 Part a

The largest positive integer that can be represented on a 64-bit computer is 18446744073709551615. This is the value for the unsigned maximum value of a long integer. It is appropriate to use the unsigned long integer when dealing with only positive integers which is the case for the infinite series shown above.

Table 1: Table of Results for Problem 1

	Single Precision	Double Precision
Machine Epsilon	1E-5	1E-9
Value of n	8388608	2000000000001
Value of sum	15.403683	28.9013836030399958
Time	0.024421 s	12863.106700 s

2.2 Part b

The float.h library was used to obtain the following values for the machine epsilon value for single precision and double precision. The results produced from the code are listed below. They can also be found in the text file Homework2.txt.

Single Precision: FLT_EPSILON = 1E-5
Double Precision: DBL_EPSILON = 1E-9

2.3 Part c

For single precision the sum stopped changing with the value of n = 8388608. For double precision this was a little trickier. The long int type was used for n in the double type since it can extend the range of integers. It was found that n reached a value of 2000000000001 with an elapsed time of 12863.11 seconds.

2.4 Part d

The larges value for the divergent series that can be obtained on the computer depends if the computations are being done in single precision or double precision. For single precision the largest value for the divergent series is 15.403683. For double precision the sum reached a value of 28.90 after the computation was run continuously for 12863.11 seconds.

2.5 Part e

There was a very noticeable difference in the time it took to run the single precision series and the double precision series. The results are shown below. It should be noted that the double precision series was run for a very long time and the computation did not run until completion. Knowing the machine epsilon and the value of n at at the time the computation was stopped the time top run the full computation was estimated. It should be noted that when the problem allows, computations should be done in single precision to save time. However, it may not always be appropriate to use single precision depending on the problem at hand.

Single Precision: 0.024421 s
Double Precision: 12863.106700 s Estimating the time needed to run the full computation:

$$\frac{1}{n} = \frac{1}{20000000001} = 5E - 11 \tag{2}$$

$$\frac{12863.106700s}{t} = \frac{5E - 11}{1E - 9} \rightarrow t = 257262.134s \tag{3}$$

So, it would take approximately 71 hours, 27 minutes, and 42 seconds to complete the double precision calculation.

3 Problem 2

In this problem the taylor series expansion of e^x was analyzed in single precision as well as double precision and compared with the built in exponential function in the math library. For parts a and b the corresponding source code file is named HW2_problem2.c. For part c the source code file is HW2_partc.c.

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + ... = \sum_{n=1}^{\infty} \frac{x^n}{n!} \tag{4}$$

3.1 Results

Table 2: Problem 2 Results

x	Single Precision	Double Precision	Math Function exp(x)
1	2.718282	2.718281828459046	2.718281828459045
5	148.413177	148.413159102576572	148.413159102576600
10	22026.466797	22026.465794806710619	22026.465794806717895
15	3268994.000000	3269017.372472110670060	3269017.372472110670060
20	484442848.000000	485165195.409790158271790	485165195.409790277481079
-1	0.000000	0.000000000000000	0.367879441171442
-5	-4.000000	-4.000000000000000	0.006737946999085
-10	-9.000000	-9.000000000000000	0.000045399929762
-15	-14.000000	-14.000000000000000	0.000000305902321
-20	-19.000000	-19.000000000000000	0.000000002061154

3.2 Part a

Summing in the natural order, I used the stopping criteria when the $\frac{x^n}{n}$ term becomes smaller than the machine epsilon for single precision or double precision. After this point, the least significant digit is not representable by the computer. This is the stopping criteria that is implemented in the code attached.

3.3 Part b

It is clear that the method used to compute e^x for values of $x < 0$ does not work. To obtain more accurate results we can compute e^{-x} by using our current series and evaluating $\frac{1}{e^x}$. So, to compute e^{-1} we would compute the sum of the series for e^1. The to get the result for e^{-1} we evaluate $\frac{1}{e^1}$. The results for negative values of x via this method are listed below in Table 2.

Table 3: Using the series for $\frac{1}{e^x}$ to compute e^{-x}

x	Single Precision	Double Precision	Math Function exp(x)
-1	0.367879	0.367879441171442	0.367879441171442
-5	0.006738	0.006737946999085	0.006737946999085
-10	0.000045	0.000045399929762	0.000045399929762
-15	0.000000	0.000000305902321	0.000000305902321
-20	0.000000	0.000000002061154	0.000000002061154

3.4 Part c

To compute e^{-x} more accurately, the terms in the series can be rearranged as shown below. The results from this method are shown in table 3. The criterion used in the while loop was $x^n > FLT\_EPSILON * n!$ for single precision and $x^n > DBL\_EPSILON * n!$. It should be noted that this does not work for all negative values in single precision because single precision is limited to 6 significant figures. However, the method shows that it is successful when comparing the results from the double precision computation to the built in exponential function in the math library.

$$e^{-x} = 1 - \frac{x}{1!} + \frac{x^2}{2!} - \frac{x^3}{3!} + ... = \sum_{n=1}^{\infty} (-1)^n \frac{x^n}{n!} \tag{5}$$

Table 4: Rearranged Series Computation of e^{-x}

x	Single Precision	Double Precision	Math Function exp(x)
-1	0.367879	0.367879441171442	0.367879441171442
-5	0.006738	0.006737946999087	0.006737946999085
-10	0.000103	0.000045399929434	0.000045399929762
-15	9.909435	0.000000305912762	0.000000305902321
-20	213007.062500	0.000000004992639	0.000000002061154