Homework 5: Exercise 6: Round-Off Assignment

Justin Campbell
UT eID: jsc4348
SDS 335
Dr. Eijkhout

October 20th, 2021

1 Introduction

In this assignment, the author exercised an understanding of floating point operations in C to explore round-off errors in math computations. In particular, a C program was developed that approximates the value of the base for the natural logarithm "e" using the following limit approximation:

$$e = lim((1 + 1/n))^n$$

$$n- > \infty$$

The author used the "float" datatype, and thus, single precision for the first solution, and used the "double" datatype, (double precision), for the second solution while incorporating the limit relation shown above in both solutions.

2 Solution 1: Single Precision

Shown in the code fragment below is the single precision source code that evaluates the limit expression to approximate the value of the base for the natural logarithm for ten different values of n, namely $n = 10^k$ for k=1,2,....,10.

```
///////////////////// Single Precision Solution ////////////////////
  float n, e_app;

  for (int k = 1;k <= 10; k++){

    n = pow(10,k); // Update the value of upper bound
    e_app = pow(1+1/n, n); // Approximated value for base of natural log
    // Return approximated values for the base of natural log for n=10^k (k
     =1,...,10)
    printf("Base of natural log at n= 10^%d is %0.5f\n ", k,e_app);

  }
```

Depicted below is the output as it appears on the terminal window for the single precision solution. It can be seen that the value of the limit expression converges to values that more closely approximate the actual value of the base for the natural logarithm for increasing values of k for k=1,2,...,8. However, for values of k=8,9,10 we notice that the calculated value for "e" is reported as 1.0, a significant round-off error in the floating point computation. The reasoning for this error lies in an overflow of the memory allocated to the variable "*e_app*" as a single precision floating point datatype. In particular, single precision variables have only 32 bits of memory, eight of which are reserved for exponential bit width. When a number that is being calculated in a computation for a single precision variable exceeds eight bits in exponential width, the variable overflows, and reports incorrect values. Since a value of "10^7" exceeds the eight bit exponential width, the variable overflows and reports erroneous values of "3.29397", "1.00000", "1.00000", and "1.00000" for values of k=7,8,9,and 10 respectively. This significant round-off motivated the double precision solution to follow.

```
Base of natural log at n= 10^1 is 2.59374
 Base of natural log at n= 10^2 is 2.70481
 Base of natural log at n= 10^3 is 2.71705
 Base of natural log at n= 10^4 is 2.71860
 Base of natural log at n= 10^5 is 2.72196
 Base of natural log at n= 10^6 is 2.59523
 Base of natural log at n= 10^7 is 3.29397
 Base of natural log at n= 10^8 is 1.00000
 Base of natural log at n= 10^9 is 1.00000
 Base of natural log at n= 10^10 is 1.00000
```

3 Solution 2: Double Precision

To address this round-off error, the author implemented a double precision solution by replacing float datatype definitions with double datatype definitions. This implementation is reflected in the code shown below where the limit approximation was replicated as the value approximation method.

```
//////////////////////  Double Precision  /////////////////

  double n_2, e_app_2;

  for (int k_2 = 1;k_2 <= 10; k_2++){

    n_2 = pow(10,k_2); // Update the value of upper bound
    e_app_2 = pow(1+1/n_2, n_2); // Approximated value for base of natural
     log

    // Return approximated values for the base of natural log for n=10^k (k
     =1,2,...,10)

    printf("Base of natural log at n= 10^%d is %0.10f\n ", k_2,e_app_2);
  }
```

Depicted below is the output for the double precision solution as it appears on the terminal window. In referencing the output, we note that the approximation for the base of the natural logarithm converges very close to the desired value (=2.72) very quickly. In particular, by $k = 10^4$, the approximated value is within 0.002 of the actual value. Thus, we can conclude that implementing a double datatype, which has a 64 bit memory capacity, eliminates the round-off error in approximating the value of the base of the natural logarithm. Put differently, the maximum value in the computation, 10^{10} has an exponential bit width of less than 11, which is the limit for double datatype precision, thus, the approximations for double precision did not overflow.

```
Base of natural log at n= 10^1 is 2.5937424601
Base of natural log at n= 10^2 is 2.7048138294
Base of natural log at n= 10^3 is 2.7169239322
Base of natural log at n= 10^4 is 2.7181459268
Base of natural log at n= 10^5 is 2.7182682372
Base of natural log at n= 10^6 is 2.7182804691
Base of natural log at n= 10^7 is 2.7182816941
Base of natural log at n= 10^8 is 2.7182817983
Base of natural log at n= 10^9 is 2.7182820520
Base of natural log at n= 10^10 is 2.7182820532
```