Round Off of e

Michel Gonzalez

October 17, 2021

In this assignment we were instructed to test a limit approximation of the value $e \approx 2.72$. The value e was given by $e = \lim_{n\to\infty}(1+\frac{1}{n})^n$. where $n = 10^k$ for $k = 1, \ldots, 10$. The code used to compute this is given below,

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

float main()
{
  float e; //initialize e

  int i, n, k;

  printf("Enter a number
   from 1 to 10: ");

  // takes input value
  scanf("%d", &k);

  // uses exponent to get n
  // this will limit the value of n
  n = pow(10.0, (float) k);

  float j;

  // the division will cause error
  // when n gets large

  j = 1 + ((float) 1 / (float) n);

  e = pow( j, (float) n);

  printf("when n is %d, e is equal
   to %f \n", n, e);

  return 0;

}
```

The following table shows the results for $n = 10^1, \ldots, 10^{10}$.

```
when n is 10, e is
equal to 2.593743

when n is 100, e is
equal to 2.704811

when n is 1000, e is
equal to 2.717051

when n is 10000, e is
equal to 2.718597
```

```
when n is 100000, e is
equal to 2.721962

when n is 1000000, e is
equal to 2.595227

when n is 10000000, e is
equal to 3.293968

when n is 100000000, e is
equal to 1.000000

when n is 1000000000, e is
equal to 1.000000

when n is -2147483648, e is
equal to 0.000000
```

As n gets significantly large the computation begins to have trouble. This is due to the amount of storage a single precise floating point number can store. In single precise, it can store up to 32-bits of information where 8-bits are given to the exponent, 23 to the Matisse, and 1 t the sign of the number with a bias of 127. It begins to have trouble if n is larger than 10^5. Past this, the values of e begin to be affected by the bias between $n = 10^6$ and $n = 10^8$.

Once n gets larger then the value of $\frac{1}{n}$ gets to small to store since the decimal is over 7 digits long. Thus, the computer rounds $\frac{1}{n} \approx 0$, so function is computed to equal 1.000000. The last notable error is when $n = 10^{1}0$ which causes n to become $\approx -\infty$. This indicates that the power calculation must have exceeded the upper limit of the single pierces floating point number casuing it to give the special value of $-\infty$.

I was able to get a better computation when i chagned the calcualtion of e to be $e = \sum_{i=0}^{n} \frac{1}{!n}$. The code used to compute e is below

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

int factorial(int n)
{
  if (n == 0) {
    return 1;
  }
  return n * factorial(n - 1);
}

float main()
{
  int i, k, n;
  float e;

  printf("Enter a number
  between 1 to 10: ");
  scanf("%d", &n);
  float j;

  for(i = 0; i <= n; i++){
    j = factorial(i);
    e += (1.0 / (float) j);
  }
  printf("When n is %d, e is
  equal to %f\n", n, e);
  return 0;
}
```

The following table shows the results for $n = 1, 3, 5, 7, 9, 10$. Anything above $n = 9$ computes but does not change the single precise value of e.

```
When n is 1, e is
equal to 2.000000

When n is 3, e is
equal to 2.666667

When n is 5, e is
equal to 2.716667

When n is 7, e is
equal to 2.718254

When n is 9, e is
equal to 2.718282

When n is 10, e is
equal to 2.718282
```

Testing the values above $n = 9$ will compute until $\frac{1}{!n}$ becomes too small to register causing the program to set e equal to ∞. Regardless, testing above $n = 9$ resulted in the same values for e thus, this method of approximating e is better since it does not reacquire any large number calculations.

In actual computing, e is computed using the $\sum_{i=0}^{n} \frac{1}{!n}$ method, however double precision is used instead of single precision.