AE 706: Assignment 1

Pavan R Hebbar - 130010046

January 18, 2017

Contents

1 Introduction:

This assignment deals with the basics of floating point arithmetic and implementing the knowledge gained on the computer. Through this assignment we learn about machine epsilon, how to modify functions so as to minimize the error, finite difference schemes to estimate the derivatives and how they change with h. The first question was implemented using C++ language and the rest of the questions were done in Python 2.7

In this assignment Question 1 (section 2) deals with determining the machine epsilon for different data types in C++ language. Question 2 (section 3) deals with doing the same in Python. Question 3 (section4) deals with plotting a function to show how the errors could be quiet significant and to modify the function to minimize the error. Question 4 (section5) asks us to approximate the derivative of $f(x)$ for various orders of significance. In Question 3 (section 6), we plot how the relative errors change with h in different schemes.

2 Question 1:

In this question we have used the definition of machine epsilon as "The largest positive floating point number such that $1 + \delta = 1$ when represented on the computer".

To estimate the machine epsilon for float data type in C++ following function was defined:

```
float eps_float()
{
    float epsilon = 1;
    while ((float)(1.0 + epsilon) > 1.0)
    {
        epsilon = (float)(epsilon*0.5);
    }
    return epsilon;
}
```

Listing 1: Function to determine epsilon for float

In this code, we see that epsilon was defined to be a variable of type *float* and assigned a value 1. The value of machine epsilon was found by halving the value of epsilon until $1 + \epsilon = 1$. Similar programs were written to find the machine epsilon for *double* and *long double* data types. We see that

- $\epsilon_{float} = 5.96046e{-}8$: Note that this value exactly equals 2^{-24} pointing towards the fact that 23 bits are reserved for mantissa in float (Note that the 1st 1 isn't stored).
- $\epsilon_{double} = 1.11022e - 16$: $= 2^{-53}$ implying that 52 bits are reserved for mantissa in double

- $\epsilon_{longdouble} = 5.42101e-20: = 2^{-64}$ implying that 63 bits are reserved for mantissa in long double

3 Question 2:

Here, we do the same algorithm as above but in Python. The code used is:

```
def machineeps():                    #find epsilon of machine
  epsilon = 1.0
  while (1 + epsilon > 1.0):
    epsilon = epsilon*0.5
  return epsilon
```

Listing 2: Function to determine machine epsilon

In this case we get the result $epsilon = 1.11022302463e-16$ pointing to the fact that in python float always contains 64 bits in total and 53 bits in mantissa.

4 Question 3:

In this question we are asked to plot the graph of $(1-cos(x))/(x*x)$ in the range $[-4e-8, 4e-8]$. Ideally, one must get a values very close to 0.5 for these small values of x

On analyzing the function we can notice that $cos(x)$ will be very close to 1 in the given range and subtracting it from 1 will lead to loosing of significant digits. In other words the function isn't well conditioned. Thus implementing this directly in a computer will lead to large errors.

To reduce this loss of significance we can follow the following methods:

$$\frac{1-cos(x)}{x*x} = \frac{1-cos(x)}{x*x}\frac{1+cos(x)}{1+cos(x)} = \left(\frac{sin(x)}{x}\right)^2 \times \frac{1}{1+cos(x)} \tag{1}$$

OR

$$\frac{1-cos(x)}{x*x} = \frac{2sin^2(x/2)}{x^2} = 2 \times \left(\frac{sin(x/2)}{x}\right)^2 \tag{2}$$

Through both of the above methods we have eliminated subtraction of quantities very close to each other.

Figure 1 is the graph of all the above functions plotted using "matplotlib" module of Python.

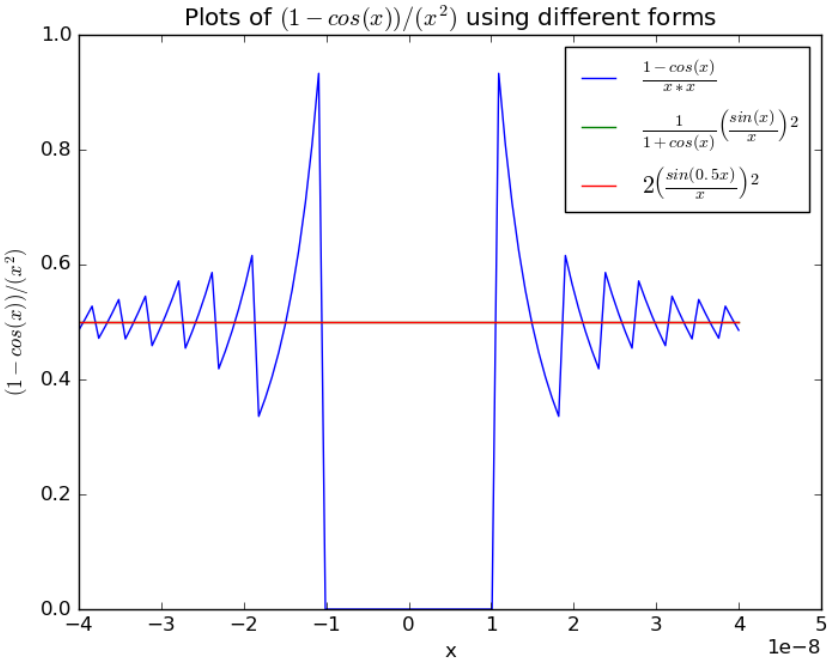

Figure 1: Graph of $(1 - cos(x))/(x^2)$ plotted using various methods

In the above figure we can see that the direct implementation of $(1 - cos(x))/(x * x)$ leads to incorrect results due to loosing of significant digits. The oscillations in this plot most probably arise due to rounding. We can see that both the above specified modifications of the functions lead to correct results.

5 Question 4:

The general methods for approximating $f(x)$ involve the use of Taylor series.

$$f(x+h) = \sum_{k=0}^{\infty} \frac{f^{(k)}(x)}{k!} h^k \tag{3}$$

$$f(x-h) = \sum_{k=0}^{\infty} \frac{(-1)^k f^{(k)}(x)}{k!} h^k \tag{4}$$

5.1 First Order Approximation:

5.1.1 Forward difference

Subtracting $f(x)$ from equation 3 we get:

$$f(x+h) - f(x) = hf'(x) + \left[\frac{h^2}{2!}f''(x) + \frac{h^3}{3!}f'''(x)...\right]$$

which can also be written as

$$f(x+h) - f(x) = hf'(x) + \frac{h^2}{2}f''(\zeta)$$

$$\implies f'(x) = \frac{f(x+h) - f(x)}{h} - \left[\frac{h}{2}f''(\zeta)\right] \tag{5}$$

i.e first order approximation of $f'(x) = \dfrac{f(x+h) - f(x)}{h}$ and the error is $\left|\dfrac{h}{2}f''(\zeta)\right|$

5.1.2 Backward difference

Subtracting equation 4 from $f(x)$ we get:

$$f(x) - f(x-h) = hf'(x) + \left[-\frac{h^2}{2!}f''(x) + \frac{h^3}{3!}f'''(x)...\right]$$

which can also be written as

$$f(x) - f(x-h) = hf'(x) - \frac{h^2}{2}f''(\zeta)$$

$$\implies f'(x) = \frac{f(x) - f(x-h)}{h} + \left[\frac{h}{2}f''(\zeta)\right] \tag{6}$$

i.e first order approximation of $f'(x) = \dfrac{f(x) - f(x-h)}{h}$ and the error is $\left|\dfrac{h}{2}f''(\zeta)\right|$

5.2 Second Order Approximation - Central difference

Subtracting equation 4 from equation 3, we get:

$$f(x+h) - f(x-h) = 2hf'(x) + \left[\frac{2h^3}{3!}f'''(x) + \frac{2h^5}{5!}....\right] \tag{7}$$

which can also be written as

$$f(x+h) - f(x-h) = 2hf'(x) + \frac{2h^3}{3!}f'''(\zeta)$$

$$\implies f'(x) = \frac{f(x+h) - f(x-h)}{2h} + \left[-\frac{h^2}{6}f'''(\zeta)\right] \tag{8}$$

Thus the second order approximation of $f'(x) = \dfrac{f(x+h) - f(x-h)}{2h}$ and the error is $\left|\dfrac{h^2}{6}f'''(\zeta)\right|$

5.3 Fourth order approximation

The fourth order approximation of $f'(x)$ can be derived from its second order approximation using Richardson's Extrapolation technique. Let us denote the central difference formula as $D(h)$ i.e

$$D(h) = \frac{f(x+h) - f(x-h)}{2h}$$

From equation 7 we have

$$f'(x) = D(h) - \frac{h^2}{3!}f^{(3)}(x) - \frac{h^4}{5!}f^{(5)}(x)... \tag{9}$$

We can also write $f'(x)$ for $2h$ in a similar way

$$f'(x) = D(2h) - 4\frac{h^2}{3!}f^{(3)}(x) - 16\frac{h^4}{5!}f^{(5)}(x)... \tag{10}$$

Multiplying equation 9 by 4 and subtracting from 10,

$$3f'(x) = 4D(h) - D(2h) + 12\frac{h^4}{5!}f^{(5)}(x)... \tag{11}$$

Thus,

$$f'(x) = \frac{4D(h) - D(2h)}{3} + 4\frac{h^4}{5!}f^{(5)}(x)... \tag{12}$$

$$\implies f'(x) = \frac{-f(x+2h) + 8f(x+h) - 8f(x-h) + f(x-2h)}{12h} + O(h^4) \tag{13}$$

Thus the fourth order approximation of
$f'(x) = \dfrac{-f(x+2h) + 8f(x+h) - 8f(x-h) + f(x-2h)}{12h}$ and the error is
$\left|4\frac{h^4}{5!}f^{(5)}(x)\right|$

6 Question 5:

In this question we analyze how the errors in the derivative of $sin(\pi/4)$ vary with changing value of h using different finite difference schemes. We particularly analyze the finite difference schemes discussed in the previous question. Figure 2 shows the plots of the errors using the above schemes in logarithmic scales.

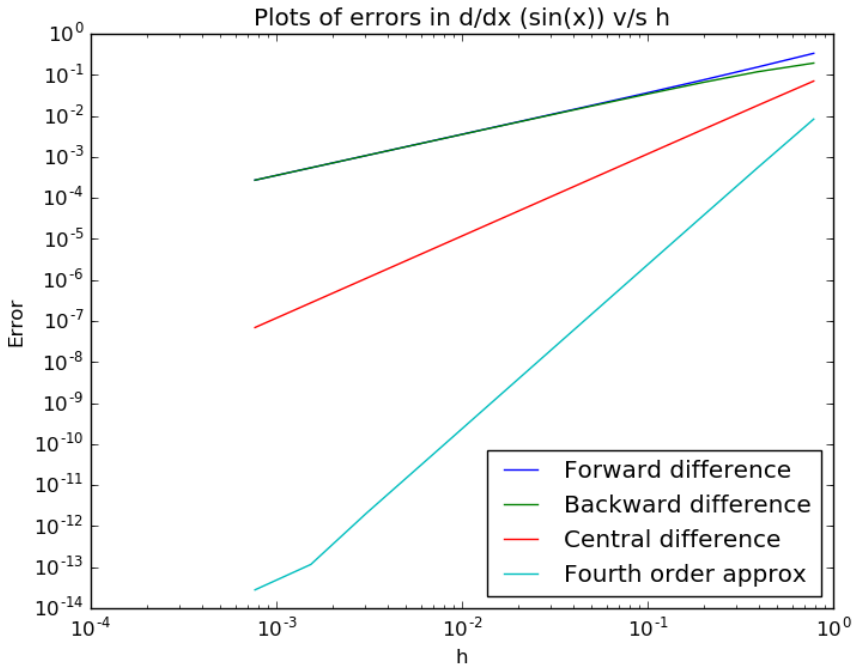

Figure 2: Errors in estimating $sin(\pi/4)$ using different difference schemes

From the above plot we can clearly see that:

- In case of forward and backward difference schemes $log(errors)$ vs $log(h)$ has slope roughly equal to 1 showing that $error \propto h$
- In case of central difference scheme $log(errors)$ vs $log(h)$ has a rough slope of 2 showing that $error \propto h^2$
- In case of the 4th order approximation that we arrived at in the previous question $log(errors)$ vs $log(h)$ has a rough slope of 4 showing that $error \propto h^4$

7 Conclusion

This assignment gives a better insight into floating point arithmetic, the risks involved in it and how to reduce the errors. The value of epsilon gives a better understanding of the system storage techniques and the data types used in C++. Deriving the finite difference schemes for first derivative to 1^{st}, 2^{nd} and 4^{th} orders of accuracy introduces different techniques that can be used to estimate a function to a better accuracy. The last question gave us a visualization of the finite difference schemes and the errors involved in them.