PYU33C01

Assignment 1: Numerical Differentiation

Aaron Miller, 17322856

September 22, 2019

Contents

1 Introduction, and theory

The analytical second derivative of $\cos(x)$, is $-\cos(x)$. The formula for the numerical value of the second derivative of the function at a point x was calculated by the following formula

$$f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} = \frac{\cos(x+h) - 2\cos(x) + \cos(x-h)}{h^2} \tag{1.1}$$

for some small quantity h. The absolute value, and relative error of the function was calculated using

$$\text{Absolute error } = |\text{ True value - Approximative value }|$$

and

$$\text{Relative error } = \left|\frac{\text{Absolute error}}{\text{True value}}\right|$$

Note that throughout this assignment, I have used a step size of $\Delta x = 0.001$, for any calculations, and in plotting all graphs.

Rounding errors occur due to the computer being only able to store a finite number of digits of a floating point number, so that the last digit is rounded up or down. For example $1000.0 + 0.025 = 1000.025 \approx 1000.0$. Subtractive cancellation emerges from rounding errors, which cause float addition to not be associative. It means that when one subtracts two similarly large numbers, the resulting small number contains only the least significant figures, and the result has less precision then the large numbers. In order to mitigate this effect when performing a sequence of additions, the numbers should be added in the order of the smallest in magnitude to the largest in magnitude.

2 Plot of analytical and numerical results in the range $x \in [0 : 4\pi]$

Figure 2.1 illustrates the analytical, and numerical (Equation 1.1) solutions to the second derivative, and the function itself $\cos x$, in the range $x \in [0, 4\pi]$, for a couple of different values of h. In comparison of the numerical graphs, and the analytical graph we see that for $h = 0.1$ the two graphs look virtually identical. But for $h = 1.0$ there is error between them. This is due to the value of the error, being of order $O(h^2) = -h^2/12 f^{(4)}(x) - h^4/360 f^{(6)}(x) + \cdots$, meaning as the error increases, the higher powers (ie: powers of 2,4...) of h increase the error dramatically when $h < 1$.

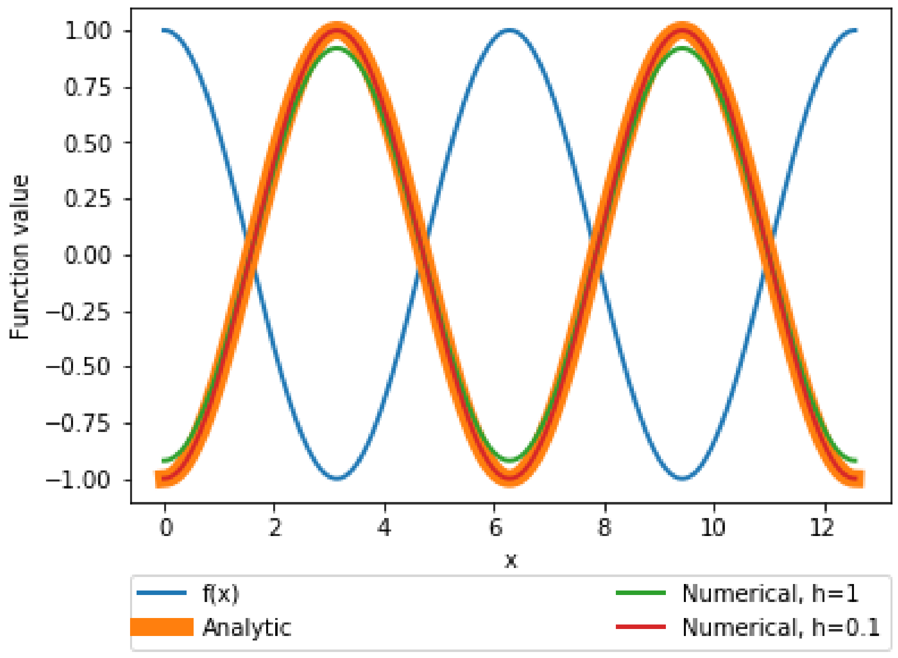

Figure 2.1: Plot of analytical and numerical result for $h = 0.1$

3 Plot of the variation of absolute error in the range, $x \in [0 : 4\pi]$

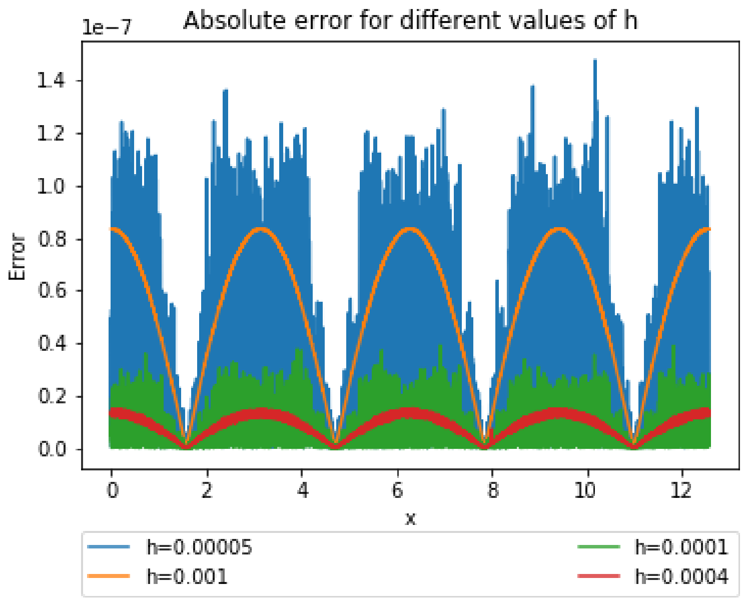

Figure 3.1: Plot of the variation of absolute error in the range, $x \in [0 : 4\pi]$

Figure 3.1 shows a plot of the variation of the absolute error in the numerical solution to the function. Here we note that as the value of h decreases (0.001 to 0.0004), the absolute error decreases as expected. But as the value decreases (0.0004 to 0.0001 to 0.00005) we see the error increases. An anomaly which we will address in the next section.

4 Highest accuracy of relative error

The anomaly noted in the previous section (ie: the point where further reduction of error does not reduce the error further) was due to what is called a floating point rounding error. This error occurs due to the way the machine stores extremely small floating point numbers in its memory. How this translates to our given problem, is that when the value of h gets smaller the relative error gets smaller as expected. But at a certain h value, the error starts to increase when the value of relative error gets small enough for rounding errors to become significant. I used an incremented while loop to find this value (which i knew was between 0.001 and 0.0001 from figure 3.1) of $h = 0.000398$, with the corresponding highest accuracy in the relative error that we can achieve being 1.324×10^{-08}. This is illustrated in figure 4.1 below.

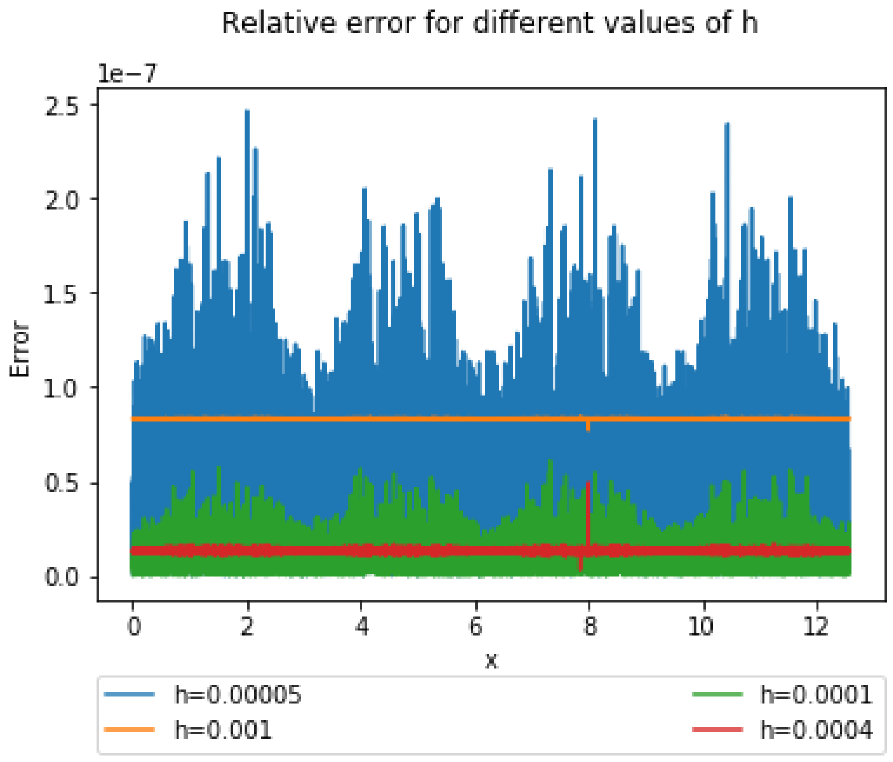

Figure 4.1: Plot of the variation of relative error in the range, $x \in [0 : 4\pi]$

In analysing figure 4.1, we see that indeed our derived value of h has the lowest relative error across the range. As h decreases further we see the error becomes 'hairier', and increases as rounding errors become relevant.

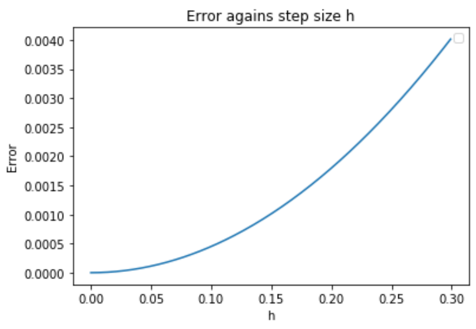

Figure 4.2: A figure

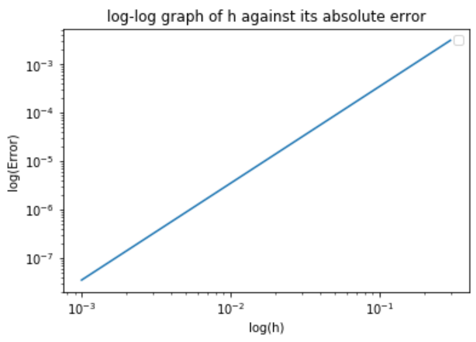

Figure 4.3: Another figure

Figure 4.2 and 4.3 verify what we already know, that as h increases, the absolute error increases exponentially.

5 Investigation of subtractive cancellation

For this part, we input the central difference method in two ways. I defined the previous equation 1.1 as a function and I also defined

$$f^{(2)}(x)_{CD} = \frac{(f(x+h) - f(x)) - (f(x) - f(x-h))}{h^2} \tag{5.1}$$

as a function. Here we note that analytically, 5.1 and 1.1 are equivalent, but due to the way that python processes how the functions are defined we get different results. Below is a graph of how this difference (subtractive cancellation error) was realised over the given range.

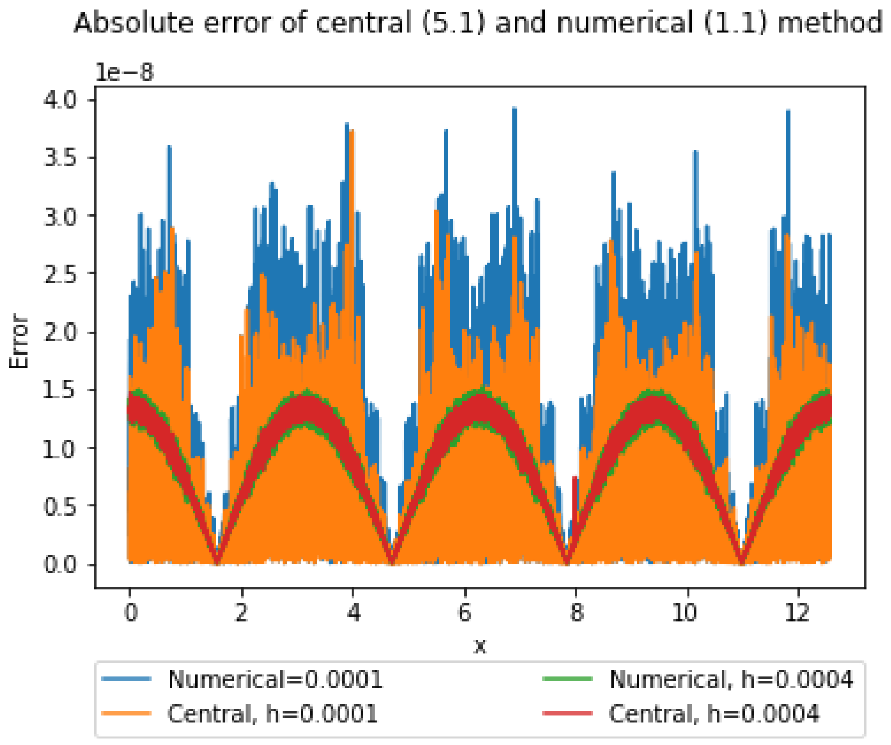

Figure 5.1: Variation of absolute error in the range, $x \in [0 : 4\pi]$

We can see in this figure that the absolute error of 1.1 is greater then the error of 5.1, over this range. At $h = 0.0001$ the error is about 1.0×10^{-8}, and at $h = 0.0004$ the error is about 0.05×10^{-8}. The difference in magnitude of this error is because the cancellation effect only becomes note able, when h becomes small enough for rounding error to take place. The difference is obvious when we consider our discussion, in the introduction, on subtractive cancellation. When the machine reads the code for 1.1 it preforms the operations in order with the inner brackets first,

$$[(f(x+h) - 2f(x)) + f(x-h)]. \tag{5.2}$$

And for 5.1,

$$[(f(x+h) - f(x)) - (f(x) - f(x-h)]. \tag{5.3}$$

Now the subtractive cancellation has the greater effect in 5.2 as there is greater difference between each of the terms being summed. But in 5.3, there is less cancellation error because $-2f(x)$ is split in two parts, and the evaluated brackets are closer in value to each other.