Computer Simulation Assignment I

Valentina Watanabe
21367661

September 2024

1 Introduction

This report explores numerical methods and their applications, intending to study the importance of precision and accuracy in computer simulations. We investigate the Babylonian method for finding square roots, examine the precision limitations of floating-point representations through overflow and underflow phenomena, and determine machine epsilon, which signifies the smallest discernible difference in numerical computations. Additionally, we employ forward and central differentiation techniques on fundamental mathematical functions, assessing their accuracy relative to analytical solutions. This analysis illustrates the practical implementation of numerical methods while also highlighting the computational limitations when approximating data.

2 Section A

This first exercise consisted of building a code that finds the root value of a number using the Babylonian, or Heron's, method. This is an iterative method that takes a value and uses it to approximate the square root. We use the formula, $x_{n+1} = \frac{1}{2}(x_n + \frac{a}{x_n})$ to improve each guess, using an initial guess x_0 to find x_1, and so on. The value of a is the number one wishes to find the square root of.

The procedure used to compute the square root of 2 with this method in Python was mainly a 'for' loop with a range of 10 iterations. The initial guess was $x_0 = 0.5$. We can see from the plots that after n=2, the x_n value stabilizes and seems to settle in a result. The solution produced is 1.414213562373095. While this method can give us an accurate result of up to 16 significant figures, it will never be completely exact because of the machine's limitations when approximating an irrational number.

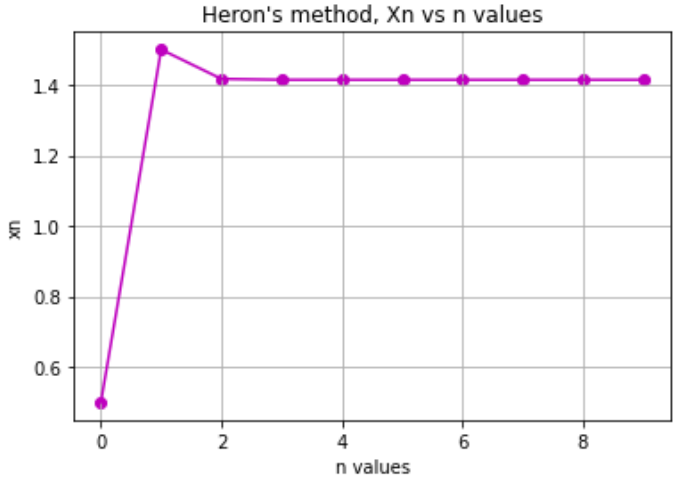

Figure 1: Xn vs. n

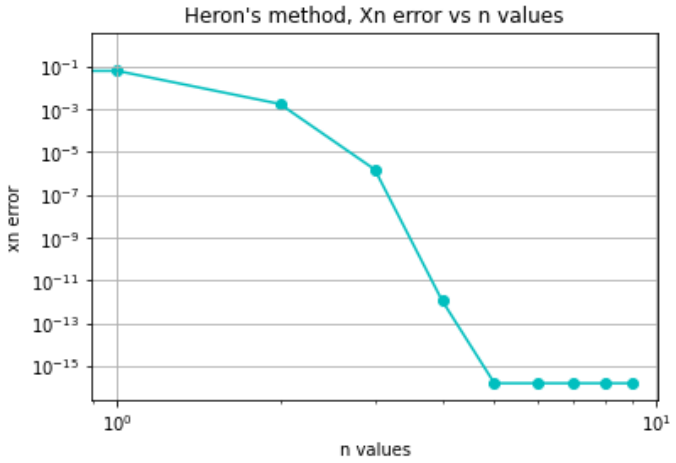

Figure 2: Xn error vs. n

3 Section B

3.1 1.3A

In this section, the precision limitation of the computer is explored. Underflow occurs when the value of an operation is so small that it cannot be represented with precision. Overflow is the opposite, which happens when a floating point number is too big. To find out what iteration this happens in, two loops were created after setting their initial value equal to 1.0. In one loop, the 'over' value

was multiplied by two in each round, until its resulting value exceeds Python's floating-point representation set to infinity. For the other loop, 'under' is divided by 2 until the variable yields a result equal to 0.0, which is when Python's representation is so small that it loses accuracy.

When we run the iterations, we get that the overflow occurred at iteration 1024, and the underflow occurred at 1075.

3.2 1.3B

The code in this section was built to find the epsilon value of the computer, which is the machine's precision. This can also be defined as the maximum possible value that can be added to a stored number without changing its value. The example that is provided in the book "Computational Problems for Physics" (Landau and Paez, 2018) follows the following equation: $1.0_c + \epsilon_m = 1.0_c$.

To do this, the code sets the variable eps = 1.0. N, the iteration number, is initially set to 2000. A loop is run where "eps" is divided by two each time and added to a variable named "one". When this variable is equal to 1.0, the loop ends. This eps value is then multiplied by two to get the machine epsilon. The found value is 2.220446049250313e-16.

This same process is done to determine the machine's precision of complex numbers. The epsilon variable is set to be eps1 = 1.0 + 0j. The loop ends once the real part equals 1.0, and the imaginary part to 0. The machine epsilon found is the same, 2.220446049250313e-16 + 0j.

3.3 1.4

The final section consists of using forward and central differentiation in the functions $cos(t)$ and e^t at t = 0.1, 1.0, and 100. The forward difference method uses the equation, $f'(t) \approx \frac{f(t+h)-f(t)}{h}$. The central different algorithm uses $f'(t) \approx \frac{f(t+h)-f(t-h)}{2h}$.

A script was built implementing the two methods of numerical differentiation with varying step sizes. The results were compared with the analytical solutions to assess the impact on accuracy using the relative error. As mentioned previously, the equations $f1 = cos(t)$ and $f2 = e^t$ were defined along with their derivatives, $f1' = -sin(t)$ and $f2' = e^t$. The derivatives were calculated at distinct points t = 0.1, 1.0, 100. The h-step sizes were defined logarithmically from a range of 1 to machine precision $\approx 10^{-16}$. For the functions and t values, the numerical derivative for each h and the relative error against the exact derivative was calculated. The relationship between the step size and the error can be visualized in the log(E) (relative error) vs. log(h) (step size) plots.

When comparing the central and forward differential methods, we can observe that the former consistently yielded lower relative errors across all evaluated points. This can be explained by the fact that the central method uses both ends of the points of differentiation, which results in averaging out errors that occur from truncation. We can also observe that for t=0.1, 1.,0 the central

method yields a much lower relative error, while for t=100 the error remains the same for both methods until around halfway through the h-step range, where the forward method returns to higher error values.

Another point of discussion is the effect of step size. As expected, the analysis of relative error as a function of step size revealed that reducing h leads to higher accuracy. However, it can be observed that after reaching a certain h value, the relative error increases dramatically again. This can be attributed to the fact that the step size approaches the value of machine precision, which results in an error increase due to numerical round-off errors. This increase in error is more dramatic and drastic for the central approximation, resulting in a sudden increasing slope for the final h values. The forward approximations tend to follow the shape of an absolute value function, with a constant increase halfway through the step size range.

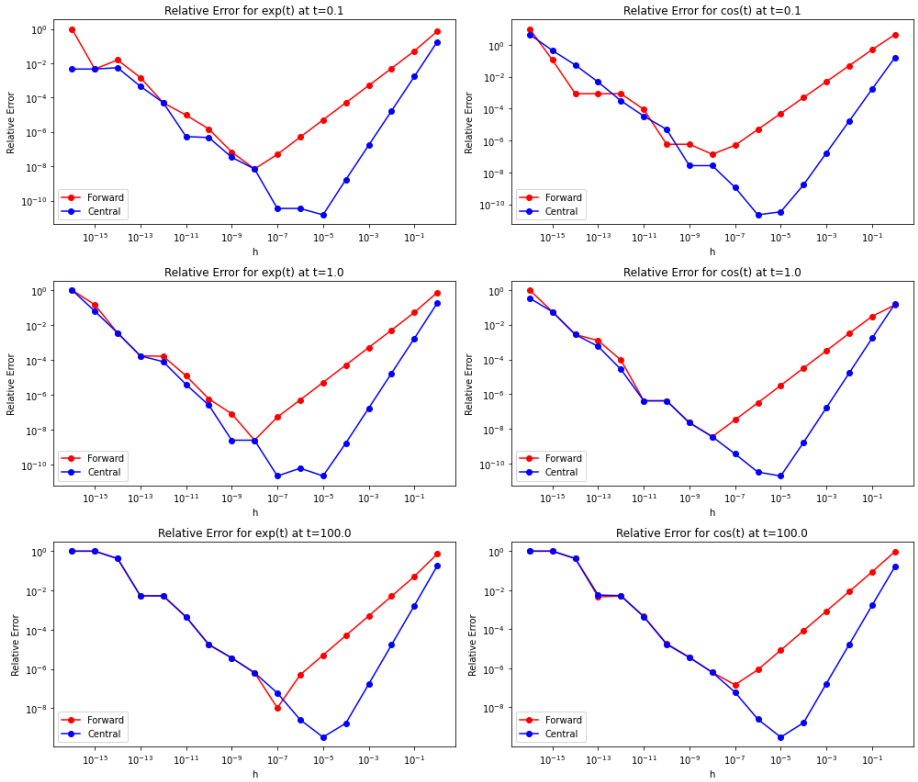

Figure 3: Relative errors vs. step size h for functions e^t and cos(t), t =0.1,1,100.

4 Conclusion

We can conclude that the exercises conducted in this report demonstrate the role that numerical methods play in achieving accurate computations approximations. Through the application of Heron's method, we obtained highly precise estimates of square roots, while our investigation into overflow and underflow demonstrated the limitations of digital number representations. We were also able to successfully determine the machine epsilon. The examination of forward and central differentiation offered valuable perspectives on their respective precision, highlighting the significance of selecting suitable approaches and increment sizes. In summary, these results investigate the role that numerical methods play in scientific computing and their accompanying limitations related to computational data.