MA348 Numerical Analysis, Numerical Error in Computing Systems

David Jefts

January 25th, 2019

Introduction

The goal of this lab is to represent error in computers due to truncation and the limits of the mantissa-exponent form of number representation. This lab uses the function $f(x) = -0.1x^4 - 0.15x^3 - 0.5x^2 - 0.25x + 1.2$ as the base function for demonstration purposes. The code used in this lab estimates the derivative of this base function using the second order approximation function, $f'(x) \approx \frac{f(x+h)-f(x-h)}{2h}$, to show how the error changes as you decrease h. The x value for this activity is 0.5

Theory-Analysis

This report assumes that computers have a limited way to represent numbers, therefor causing truncation and round-off errors. In the operating system utilized by the author, the computer stores each number with 64 bits of memory, leading to errors when the numbers used require more bits to be properly represented. The function $f(x)$ was given in the original lab activity, and the derivative function is a finite quotient function that normally is written as a limit, where h approaches 0. However, in this scenario the limit is represented by the steadily decreasing h used in the code. Additionally, each iteration uses a fixed, non-zero value for h.

Numerical Solution

This problem was solved using Fortran code and the second order approximation function for a first derivative function to iteratively find the value of $f'(0.5)$. Each iteration decreased the value of h by $\frac{1}{10}$ and then calculated the error as compared to a hand-calculated "true" value of $f'(0.5) = -0.9125$. gnu-plot was used to plot the graphs shown in the appendices and used for conclusions.

Results and Discussion

The graph in Appendix A shows a plot of the step size vs the error, with step size increasing logarithmically from 10^{-10} to 10^{-1}. Note that both axis

are on a logarithmic scale to help visualize the error change better. A tabular version of the data can be found below as well.

Step Size (h)	**True Error** (ε)
1	0.35
0.1	0.00349998
0.01	0.0000349814
0.001	0.000000331374
0.0001	0.0000000151267
1.00E-05	0.0000000185962
1.00E-06	0.0000000186184
1.00E-07	0.0000000188404
1.00E-08	0.0000000149547
1.00E-09	0.00000000385242
1.00E-10	0.00000005165870

Table 1: Table of values for the derivative of the function $f(x) = -0.1x^4 - 0.15x^3 - 0.5x^2 - 0.25x + 1.2$

As shown, the error reaches a minima when the step size is around 1^{-9} because the second order approximation function is divided by the step size (h). The nature of the function means that eventually the error will "switch" from truncation error to rounding error when the numbers are too small for the computer to be able to store and represent them properly.

Conclusions

The solver worked fairly efficiently, reaching a minimum error after only 10 iterations. The code itself could be optimized by ending the program if the error begins to increase significantly. It is fairly surprising, at least to me, is that the computer appears to only properly represent numbers greater than 10^{-10}. Additionally, after converting both axes on the graph to logarithmic the graph appears to have a weird "hump" between the h values of 1^{-8} and 1^{-4}.

Appendix A

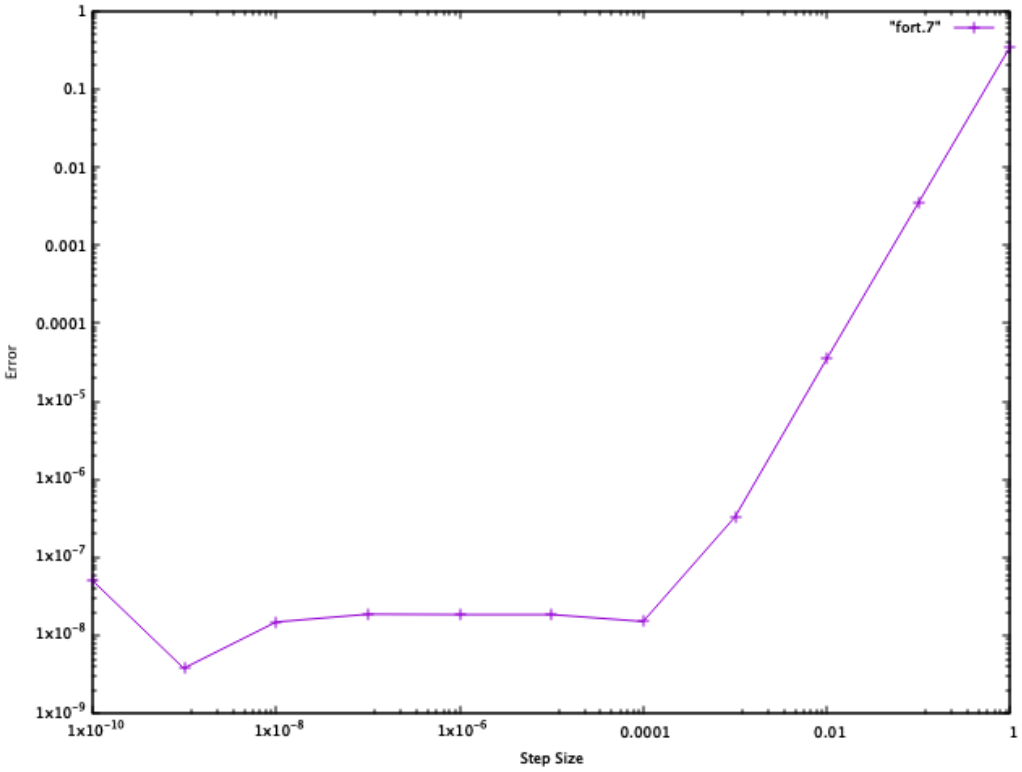

Figure 1: Graph of step-size vs error

Appendix B

```fortran
!-----------------------------------------------------------------------!
PROGRAM Errors
    ! Purpose:
    ! Evaluate F(x) = a - b*(x**3) at N+1 equidistant points in [0,1],
    ! and output the pairs x , F(x) on the screenand in file fort.7
    ! for plotting.
    ! (-0.1)*x**4-0.15*x**3-0.5*x**2-0.25*x+1.2
    !-------------------------------------------------------------------
    IMPLICIT NONE
    ! Declare the variables used in this program
    double precision x, h, val, true, error
    integer i

    x = 0.5
    true = (-0.9125)
    ! Compute F(x) = a - b*(x**3) and print x F(x):
    h = 1.0
    DO i = 1, 11
        val = (base(x + h) - base(x - h)) / (2 * h)
        error = Abs((val - true))

        write(*, *) ''
        write(*, *) '     error: ', error
        write(*, *) ' step size: ', h
        write(*, *) '   log(dx): ', log10(h)
        write(*, *) 'log(error): ', log10(error)
        write(7, *) h, error
        h = h / 10
    ENDDO
    ! Exit:
    CALL SYSTEM('gnuplot -p script.sh')
    write(*, *) 'All  Done,  BYE !'

CONTAINS

    doubleprecision FUNCTION base(x)
        IMPLICIT NONE
        doubleprecision :: x
        base = (-0.1) * x**4 - 0.15 * x**3 - 0.5 * x**2 - 0.25 * x + 1.2
    END FUNCTION base

END PROGRAM errors
```

Figure 2: Fortran program code

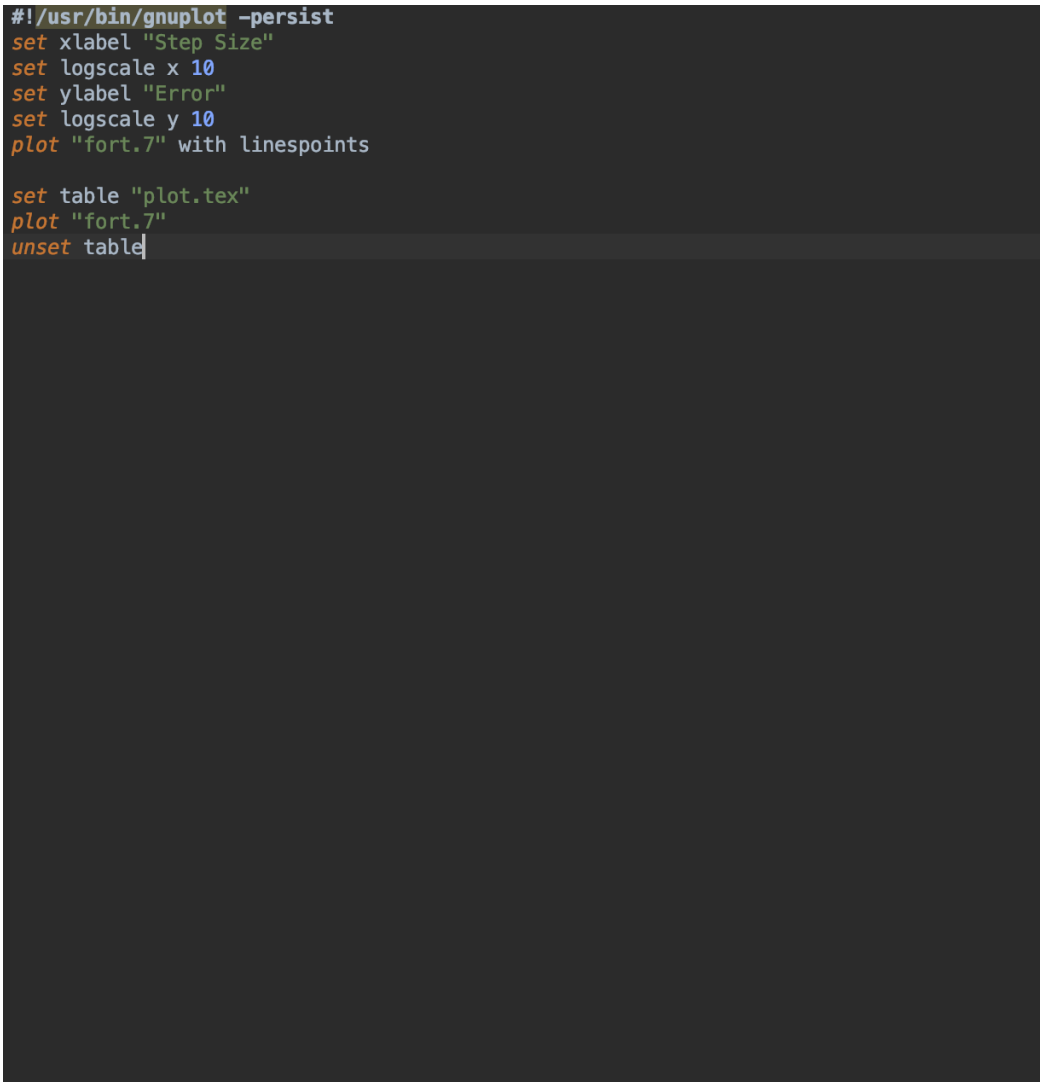

Figure 3: Script for gnuplot to set up the graph correctly