Assignment 3

ECSE 543

Professor: Prof. Dennis  Giannacopoulos

Student Name: Chris Morin

Student Number: 260344722

Question 1

The files used in this solution are “Polynomial.py” and “LagrangeInterpolate.py”. “Polynomial.py” contains a class that represents a polynomial. This class has mathematical functionality such as adding or multiplying polynomials. It can also print polynomials in a user friendly format which I use to see the final solutions. “LagrangeInterpolate.py” runs the interpolation algorithm by executing the proper polynomial operations.

1. This is done in “LagrangeInterpolate.py”. The result is as follows:



After plotting the graph, I can say that it is smooth and is probably quite a good representation over the range. I would have been suspicious if there were squiggles on the graph but there were none here.

1. The solution for this interpolation was as follows:





Above is the graph of the polynomial returned by my program. As can be seen, it’s nothing like what we would expect and clearly doesn’t represent the HvB function correctly even though it passes through all the points we provided it.

1. For all the points except the two at the ends of the range, I would set the derivative of a given point to the slope of the line joining the two points adjacent to it. There is actually a mathematical law that says that for any smooth function, there must exists a third point between two points with a slope equal to that of the line formed by the two points. The slopes at the end points will just be equal to the slopes formed by the line joining them and their single adjacent point.

Question 2

The code for this question is in “MagneticCircuit.py”.

1. F(ψ)= 39.78873577e6 \* ψ + 0.3\*H (ψ) - 8e3 = 0  
   where H is computed via interpolation of the table
2. This is computed by newtonRaph(x, tolerance). Direct output from my program: “Flux of 0.00016126936944713414 achieved[sic] after 3 iterations.” This is about 161e-6.
3. At first, I had it under the form ψ = 8000/(39.78873577e6 + 0.3\*H (ψ)/ ψ). This didn’t converge for several initial conditions so I changed the form to ψ =(8000-0.3\*H(ψ)) / 39.78873577e6. This did converge to the correct value (around 161micro).

Question 3

The code for this problem is in “ElectricCircuit.mw”. I chose to do this question in Maple to avoid computing the jacobian by hand and to make computing Vn+1 –Vn simpler.



b) Following is the maple code. As can be seen, I chose the error function to be the quotient of the magnitude of F(vn) and the value E of 220e-3 for a sense of scale. When this error dropped below 1micro, the result was returned  **> **

**> **

**> **

**> **

**> **

**> **

**> **

**> **

And the results:

























The final voltage values are v1 = 0.198134V and v2 = 0.09057V where v1 is the voltage above the first diode and v2 is the voltage above the second. Here is a graph of the error vs N and it indeed is quadratic:  
  
  
Question 4

The method of computing the integral is equivalent to simply breaking the function up into segments and adding up the products of the width of the segment and the value of the function at the midpoint of the segment. All of the code is contained in “Integration.py”. Functional programming techniques were used to make the code much more elegant!

1. Following is the error graph:  
   

The error obviously decreases with increasing N, but what is interesting about this graph is that when the axes are scaled logarithmically, the error becomes a line. In more formal terms, this means that the log of the error decreases with the log of N at a constant rate. This means that there is a diminishing returns thing going on so the higher N, the less of a benefit you receive by increasing it.

1. Here is the graph for the error when computing the integral of the natural log function:



Unsurprisingly, the error follows the same logarithmic properties as the error from the previous integral approximation.

1. Since the ln(x) function has a higher derivative (changes faster) at lower values of x, I figured it might be worthwhile to increase the density of segments there while decreasing it at the more stable higher values. My program takes as input the relative widths of the segments. Following are the relative segment widths I used. Notice they follow an exponential curve:

[1,2,4,8,16,32,64,128,256, 512]

The error found when using these 10 segments was 0.0197262 compared to 0.0342409 when using segments of equal width. The segments of unequal width clearly beat out the equally spaced ones.