Matthew Noah Leger Midterm Project Report

For the final project, I have selected the eighth option that was provided. The physical problem that I have selected entails the simulation of a water droplet falling through a cloud of water vapour. Given some initial conditions, it will be my task to determine the functions representing the position, velocity, and acceleration of the water droplet as it falls through this water vapour. All these functions will be functions of time. That being, it will be my task to rearrange the given formulas to determine how time affects position, velocity, and acceleration respectively. Afterwards, it is my task to determine how fast the water droplet grows, that being, I must determine how quickly the volume of the droplet changes as a function of time.

My initial approach that I have in mind for this project will be to create a data set of coordinates consisting of time and velocity. This will allow me to interpolate over these coordinates to estimate the velocity of the water droplet as a function of time. To do this interpolation, I have already written a natural spline function for one our previous assignments. I can use this function to do my interpolation. The reason I would like to use natural splines is because this prevents wild osculation in my estimated polynomial, and it also allows me to generate an estimation without knowing the derivative of my function. From there, I can use numeric integration over the interpolated velocity points to determine the position as a function of time. I can do this since the derivative of a position function is the velocity of that position function. Likewise, I can then do numeric differentiation on my velocity function to get my acceleration function. Again, this is because the derivative of a position function represents velocity, and the derivative of velocity represents the acceleration of the position function. To do the numeric integration, there are several methods that we have learned throughout the course. Since the error term of Simpson’s Rule is of the highest order (in comparison to the Midpoint and Trapezoidal Rules), meaning it has the lowest error bound, I will use this form of numeric integration. Again, we have created Python functions for this numeric integration in previous assignments. For numeric differentiation, I will have to use one of the methods we have previously discussed in class. As Dr. Lovekin mentioned, the most common forms of numeric differentiation that are used are the three-point and five-point formulae using Lagrange polynomials. For this, I will have to make numeric differentiation functions, as I have yet to do so. Finally, to determine the rate of change of the volume of the water droplet, I will again have to use numeric differentiation. To document my findings, I will use built in Python libraries such as matplotlib.pyplot to graph the functions I have interpolated. After numeric differentiation and integration, I can again use this library to graph the derivative and integral functions for further documentation. Finally, my project will include a written report explaining and analyzing the results of my experimentation. Overall, this project will be an enjoyable challenge to both test my abilities in understanding the course content as well as sharpen my abilities with Python.

Attached below is sample code for some of the functions I will use:

Function for natural spline interpolation:

def naturalSpline(x, a):

n = len(x)

#defining several lists

h = []

alpha = []

b = []

c = []

d = []

mu = []

L = []

z = []

#setting the lists to contain n zeros

for i in range(0, n):

h.append(0)

alpha.append(0)

b.append(0)

c.append(0)

d.append(0)

mu.append(0)

L.append(0)

z.append(0)

#determine the width between each x point and store it in h

for i in range(0, n - 1):

h[i] = x[i + 1] - x[i]

#initialize the matrix equation of Ax = b (as mentioned in claSS)

for i in range(1, n - 1):

alpha[i] = 3 / h[i] \* (a[i + 1] - a[i]) - 3 / h[i - 1] \* (a[i] - a[i - 1])

#solve the system of linear equations

L[0] = 1

for i in range(1, n - 1):

L[i] = 2 \* (x[i + 1] - x[i - 1]) - h[i - 1] \* mu[i - 1]

mu[i] = h[i] / L[i]

z[i] = (alpha[i] - h[i-1] \* z[i - 1]) / L[i]

L[n - 1] = 1

#calculate the value of the coefficients for our piecewise cubic polynimals

for j in range(n - 2, -1, -1):

c[j] = z[j] - mu[j] \* c[j + 1]

b[j] = (a[j + 1] - a[j]) / h[j] - h[j] \* (c[j + 1] + 2 \* c[j]) / 3

d[j] = (c[j + 1] - c[j]) / (3 \* h[j])

#return the b, c, and d coefficients of the cubic polynomials,

#which are of the form a + bx + cx^2 + dx^3

return(b,c,d)

Function for composite Simpson’s Rule for numeric integration:

def compositeSimpson(a, b, n):

#calculate h

h = (b - a) / n

#calculate the function points at the

xi0 = f(a) + f(b)

#variables to track summation

xi1 = 0

xi2 = 0

#iterate through and calculate the summations

for i in range(1, n):

#determine xj

x = a + (i \* h)

#if i is even, add f(x) to xi2. if i is odd, add f(x) to xi1

if (i % 2 == 0):

xi2 += f(x)

elif (i % 2 != 0):

xi1 += f(x)

#calculate the integral using the composite simpsons formula

xi = h \* (xi0 + (2 \* xi2) + (4 \* xi1)) / 3

#return the integral

return xi