Ryan Herrmann

Lab 3

BMED 430

Introduction

The purpose of the lab was to use python for root finding. The goal was to find a pore membrane size that reflects 20% of a solute with a given radius. This lab made use of the σ_R equation for the reflection coefficient. This lab both utilized the Del-squared method and the built in function of root finding in python to determine the critical pore radius. The answers were expressed into four significant figures.

Numerical Methods

The first method that was used was the del-squared approach. That was using $\lambda = \frac{a}{R_{pore}}$ where $\frac{1-\sigma_r}{[2-(1-\lambda)^2]\left[1-\frac{2}{3}\lambda^2-0.163\lambda\right]} = (1-\lambda)^2$ which leads to $\lambda = 1 - \sqrt{\frac{1-\sigma}{[2-(1-\lambda)^2]\left[1-\frac{2}{3}\lambda^2-0.163\lambda^3\right]}}$ and then solving for R_{pore}. The second method was to use the built in approach and to let $g(\lambda) = \sigma_r - \left(1 - (1-\lambda)^2[2-(1-\lambda)^2]\left[1-\frac{2}{3}\lambda^2-0.163\lambda^3\right]\right)$

Pseudo-Code

- Import required packages
- Define constants and input data
    - Solute size
    - Initialize error
    - Set tolerance, epi
    - Target reflection coefficient
    - Desired sigfigs for answer
    - Initial guess for lambda = a/Rpore
    - Initial Rpore estimate
- Define function equal to zero to be evaluated
- Find root using built in function
- Find result from function using found lambda
    - Calculate $R\mathrm{pore}$, that is (solute size/lambda)
- Set-up Lists (Python) to
    - Set-up header and index
    - store initial guess and the result
    - set-up function results
    - set-up Rpore result
- Set-up dictionary for load into a DataFrame
- Create DataFrame
- Open File and Write results to a csv file
- ----- Create Plot to show the function crossing zero
- Use linspace to set-up points between fractions 0.01 and 0.99

- Format Figure
- Plot function
- Plot line at zero
- Label axes and set grid
- Save figure as a file
- --------- Del Square Method
- Set-up lists for
    - tolerance
    - Count
    - Lambda
    - Rpore
- Define functions
    - Function del square
    - Numerical derivative
- Loop through to check error against tolerance
- Form terms for del sqr method
- Find estimate for lambda
- Estimate Rpore
- Find relative error
- Check error against tolerance
- Count iteration
- Append lists for
    - Error
    - Pore radius
    - Lambda
    - Iteration count
- Set up Header List
- Create dictionary
- Load DataFrame
- Define index as desired
- Open and write scv file
- Re-use linespace
- Format Figure
- Plot function
- Plot line at zero
- Label axes and set grid
- Save figure as file

Output:

The graph for the function crossing zero is shown in Figure 1.

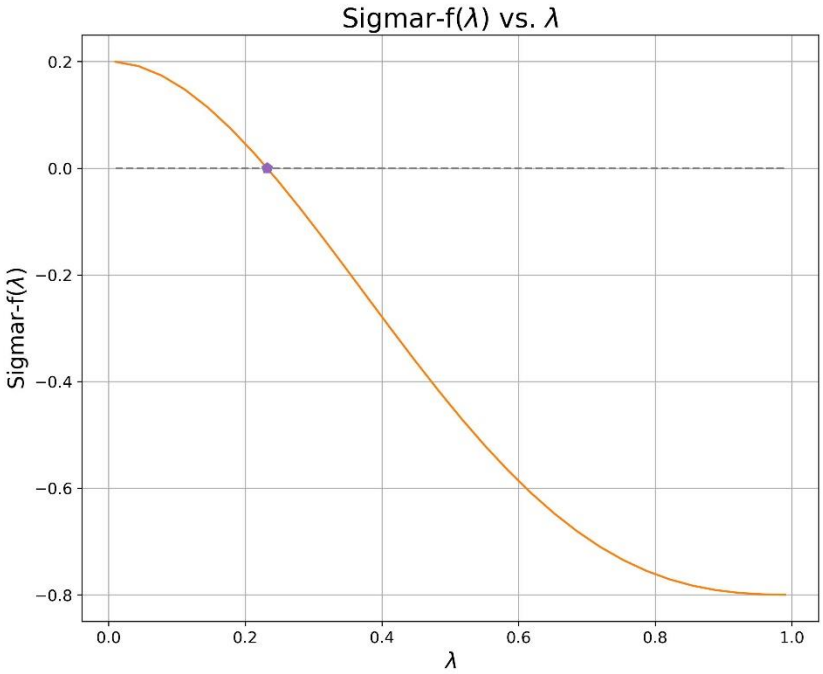

Figure 1: Del Squared, $Sigma_R$ vs Lambda: Thed graph of plotting the function and finding the zers for sigma r and the lambda value with that. The zero is plotted as the purple pentagon.

Table of the built in function and the results with relative error is shown in Table 1;

Table 1: Python Root finder results

Sigmar	Fsolve	Del Squared	Initial Guess	Iterations	Rel Error
0.2	0.2321	0.2321	0.9	6	9.65E-12

Table 2 shows the iterations and values of the del squared iterative approach.

Table 2: Del squared method with error and iterations

Iterations	Lambda	Pore Radius (nm)	Relative Error
0	0.9	111.1	10
1	0.002326	111.1	385.9
2	0.355	111.1	0.9934
3	0.2348	111.1	0.5116
4	0.2321	111.1	0.01161
5	0.2321	111.1	1.09E-05
6	0.2321	111.1	9.65E-12

Discussion

The method in python worked, and the del squared method also worked. In the lab, there were issues with the built in method if the guess was off, and that can show that sometimes using the del squared method to do the iterations can be better than the built in. In both cases, python was able to do the math and the loops for learning the guess is really good when wanting to make the error even smaller.

Appendix

```
import numpy as np
import pandas
import math
import matplotlib.pyplot as plt
from scipy.optimize import fsolve

#import variables
a = 100 #nm
sigmar = 0.2 #fraction reflected
epi = 1e-5 #convergence criteria
error = 10.0 #initialize the error
sigfigs = 4
guess = 0.9 #Lambda = a/rpore
x0 = guess
rpore = a/guess
icount = 0 #initial count

#define lists for del^2
L_error = ['%.*g' % (sigfigs,error)]
L_lam = ['%.*g' % (sigfigs,guess)]
L_rpore = ['%.*g' % (sigfigs,rpore)]
L_icount = [icount]

#list for fsolve
L_fsolve = []

def g(x,sigr):
    kpore = (1-x)**2
    term2 = 2-kpore
    term3 = 1- (2*x**2)/3 - 0.163*x**3
    term4 = kpore*term2*term3
    f = sigr-(1-term4)
    return f

#test = g(x0,sigmar)
lam1 = fsolve(g,x0,sigmar)
L_fsolve.append('%.*g' % (sigfigs,lam1[0]))
print(L_fsolve)
```

```
#del squared functions
def g1(x,sigr):
    kpore = (1-x)**2
    term2 = 2-kpore
    term3 = 1 - (2*x**2)/3 - 0.163*x**3
    f1 = 1.0 - math.sqrt((1-sigr)/(term2*term3))
    return f1

def gprime(y0,y1,y2):
    f2 = (y2-y1)/(y1-y0)
    return f2

def g3(y1,y2,gdev):
    f3 = y2 + (gdev/(1-gdev))*(y2-y1)
    return f3

while error >= epi:
    x1 = g1(x0,sigmar)
    x2 = g1(x1,sigmar)
    derivative = gprime(x0,x1,x2)

    xdel2 = g3(x1,x2,derivative)

    error = np.abs(xdel2 -x0)/xdel2
    x0 = xdel2
    icount = icount + 1
    L_error.append('%.*g' % (sigfigs,error))
    L_icount.append(icount)
    L_lam.append('%.*g' % (sigfigs,x0))
    L_rpore.append('%.*g' % (sigfigs,rpore))

#creation of dictionary
results = {"Iterations":L_icount,"Lambda":L_lam, "Pore Radius (nm)":L_rpore,
"Relative Error":L_error}

df1 = pandas.DataFrame(results)
df1.set_index("Iterations",inplace= True)
print(df1)
print("")

given = {"Sigmar $\\sigma_r$":sigmar,"Fsolve":L_fsolve, "Del Squared":L_lam[-
1],"Initial Guess":L_lam[0],"Iterations":L_icount[-1],"Rel Error":L_error[-1]}
df2 = pandas.DataFrame(given)
print(df2)
```

```
#plotting the data
xnew = np.linspace(0.01,0.99,30)
ynew = g(xnew,sigmar)

fig = plt.figure(figsize = (10,8))
plt.plot(xnew,ynew, color = "tab:orange")
plt.plot(xnew,np.zeros(len(xnew)), "--", color = "tab:gray")
plt.plot(x0,g(x0,sigmar),marker = "p",color = "tab:purple", markersize = 8)
plt.title("Sigmar-f($\\lambda$) vs. $\\lambda$", fontsize = 20)
plt.xlabel("$\\lambda$",fontsize = 16.5)
plt.ylabel("Sigmar-f($\\lambda$)", fontsize = 16.5)
plt.xticks(fontsize = 12)
plt.yticks(fontsize = 12)
plt.grid(True)
plt.show()

fig.savefig('delSquaredGraph.jpeg',dpi = 300,bbox_inches = 'tight')

L_dfs = [df2,df1]
with open('DelsquaredCSV.csv','w',newline='') as f:
    for df in L_dfs:
        df.to_csv(f)
        f.write("\n")
```