Jacob Dennis

Physics 411

8 February 2016

Prof. Gull

Homework 3 – Code

**Plotting Experimental Data**

import math

import numpy as np

import matplotlib.pyplot as plt

from matplotlib import cm

from mpl\_toolkits.mplot3d import Axes3D

#Part 1 - Sunspot Data

sunspotsData = [[], []]

sunspotsFile = open('sunspots.txt', 'r')

for line in sunspotsFile:

sunspotsData[0].append(np.float64(line.split())[0])

sunspotsData[1].append(np.float64(line.split())[1])

sunspotsData = np.asarray(sunspotsData)

sunspotsFile.close()

plt.clf()

plt.plot(sunspotsData[0], sunspotsData[1], label = 'Sunspots')

plt.legend()

plt.xlabel('Time since January 1748 (in months)')

plt.ylabel('Number of Sunspots')

plt.title('Sunspots per month since January 1748')

plt.savefig('Homework 3 problem 1 plot 1.png')

#Part 2 - STM data

STMData = []

STMFile = open('stm.txt', 'r')

for line in STMFile:

STMData.append(np.float64(line.split()))

STMData = np.asarray(STMData)

plt.clf()

plt.imshow(STMData, cmap = cm.gist\_earth)

plt.colorbar()

plt.title('Data from a STM')

plt.savefig('Homework 3 problem 1 plot 2.png')

**Spline Interpolation**

import math

import numpy as np

import matplotlib.pyplot as plt

from scipy import interpolate

climateFile = open('climate.txt', 'r')

climateData = []

header = climateFile.readline().split() #header line 1

junk = climateFile.readline() #header line 2

for line in climateFile:

climateData.append(np.float64([line.split()[-6], line.split()[-4]]))

climateData = np.fliplr(np.rot90(np.asarray(climateData), k = 3))

IntDateRange = np.arange(len(climateData[0][(-5 \* 12):]))

EMNTRange = climateData[1][(-5 \* 12):]

plt.clf()

plt.scatter(IntDateRange, EMNTRange, label = 'EMNT')

plt.ylabel('Temperature')

plt.xlabel('Months since 1 Jan 2010')

plt.title('Extreme Minumum Daily Temperatures \n for the dates 2010.01.01 to 2014.01.12')

plt.legend(loc = 0)

plt.savefig('Homework 3 problem 2 plot 1.png')

#Spline interpolation

spline1 = interpolate.splrep(IntDateRange, EMNTRange)

plt.clf()

plt.scatter(IntDateRange, EMNTRange, label = 'EMNT')

plt.ylabel('Temperature')

plt.xlabel('Months since 1 Jan 2010')

plt.plot(spline1[0][2:], spline1[1][:-2], label = 'spline representation')

plt.legend(loc = 0)

plt.title('Spline Representation of Extreme Minumum Daily Temperature \n for the dates 2010.01.01 to 2014.01.12')

plt.savefig('Homework 3 problem 2 plot 2.png')

**Finding Eigenvalues**

import math

import numpy as np

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d import Axes3D

from matplotlib import cm

def DiracDelta(a, b):

if a == b:

return 1.0

else:

return 0.0

def X\_nm(k):

N = np.arange(k)

M = N

xMatrix = []

for n in N:

xRow = []

for m in M:

xRow.append(math.sqrt(n + m + 1.0) / 2.0 \* DiracDelta(abs(n - m), 1.0))

xMatrix.append(np.asarray(xRow))

return np.asarray(xMatrix)

def H0\_nm(k):

N = np.arange(k)

M = N

HMatrix = []

for n in N:

HRow = []

for m in M:

HRow.append(DiracDelta(n, m) \* (n + 0.5))

HMatrix.append(np.asarray(HRow))

return np.asarray(HMatrix)

def H\_nm(k, l): #FIX ME FIX ME FIX ME FIX ME FIX ME

X = X\_nm(k)

X4 = np.dot(np.dot(np.dot(X, X), X), X)

H0 = H0\_nm(k)

return H0 + X4 \* l

def getMinMaxOffDiags(M):

offM = M - np.diag(np.diag(M))

return np.amax([np.amax(offM), abs(np.amin(offM))])

def QRDiagonalize(M, tol):

m = M

checkTol = getMinMaxOffDiags(M)

while checkTol >= tol:

q, r = np.linalg.qr(m)

checkTol = getMinMaxOffDiags(q)

m = np.dot(r, q)

return m

def getSmallestEigenvalues(M, tol):

DiagM = QRDiagonalize(M, tol)

Eigens = np.diag(DiagM).tolist()

min1 = min(Eigens)

Eigens.remove(min1)

min2 = min(Eigens)

return [min1, min2]

lambdaRange = np.arange(0.0, 0.1, .001)

Eigenvalues = []

for l in lambdaRange:

H = H\_nm(4.0, l)

Eigenvalues.append(getSmallestEigenvalues(H, 1.0e-8))

Eigenvalues = np.rot90(np.asarray(Eigenvalues), k = 3)

kRange = np.arange(2.0, 10.0)

Eigenvalues2 = []

for k in kRange:

EigenvalueRow = []

for l in lambdaRange:

H = H\_nm(k, l)

EigenvalueRow.append(getSmallestEigenvalues(H, 1.0e-8))

Eigenvalues2.append(np.rot90(np.asarray(EigenvalueRow), k = 3))

plt.clf()

plt.plot(lambdaRange, Eigenvalues[0], label = 'Smallest Eigenvalues')

plt.plot(lambdaRange, Eigenvalues[1], label = 'Second Smallest Eigenvalues')

plt.legend(loc = 0)

plt.xlabel('lambda')

plt.ylabel('Eigenvalues')

plt.title('Lambda versus Eigenvalues for k = 4')

plt.savefig('Homework 3 Problem 3 plot 1.png')

plt.clf()

for i in range(len(Eigenvalues2)):

plt.plot(lambdaRange, Eigenvalues2[i][0], label = str(kRange[i]))

plt.legend(loc = 0)

plt.xlabel('lambda')

plt.ylabel('Eigenvalues')

plt.title('Lambda versus Eingenvalues for a range of ks')

plt.savefig('Homework 3 Problem 3 plot 2.png')

**Finding Maxima**

import math

import numpy as np

import matplotlib.pyplot as plt

#Constants:

h = 6.62607004e-34 #J \* s

c = 299792458.0 \* 1e9 #nm/s

kB = 1.38064852e-23 #J/K

phi = (1.0 + math.sqrt(5.0)) / 2.0 # Golden Ratio

XMax = 4.965114231

def Wiens(T):

return h \* c / (XMax \* kB \* T)

def P(T, l):

return (2.0 \* h \* c\*\*2 / l\*\*5) \* (1.0 / (math.exp(h \* c / (l \* kB \* T)) - 1.0))

def goldenAlgorithmMax(a, b, f, tol):

x1 = a

x4 = b

x3 = (x4 - x1) / phi + x1

x2 = x4 - (x4 - x1) / phi

while (abs(x4 - x1) >= tol) & (abs(x4 - x1) < 5000000.0): #Added to prevent divergence and an infinite loop

if f(x2) > f(x3):

x4 = x3

x3 = x2

x2 = x4 - (x4 - x1) / phi

else:

x1 = x2

x2 = x3

x3 = (x4 - x1) / phi + x1

if (abs(x4 - x1) <= tol):

return 0.5 \* (x2 + x3)

else:

return 'Algorithm failed; sequence diverged'

TList = [100.0, 200.0, 300.0, 400.0]

WiensList = []

GoldenList = []

for T in TList:

def PTemp(l):

return P(T, l)

WiensList.append(Wiens(T))

GoldenList.append(goldenAlgorithmMax(7000.0, 29500.0, PTemp, 1e-5))

plt.clf()

plt.scatter (TList, GoldenList, marker = 'o', color = 'g')

plt.plot(TList, WiensList)

plt.xlabel('Temperature (K)')

plt.ylabel('Wavelength')

plt.savefig('Homework 3 Problem 4 plot 1.png')