Jacob Dennis

Physics 411

16 February 2016

Prof. Gull

Homework 4 – Code and Results

**Fourier Transform**

*Code*

import math

import numpy as np

import matplotlib.pyplot as plt

def DFT(y):

N = len(y)

c = np.zeros(N, dtype = np.complex\_)

for k in np.arange(N):

for n in np.arange(N):

c[k] += y[n] \* np.exp((-2j \* math.pi \* k \* n) / N)

return c

def IDFT(c):

N = len(c)

kRange = range(N)

y = np.zeros(N, dtype = np.complex\_)

for n in np.arange(N):

for k in kRange:

y[n] += c[k] \* np.exp(2j \* math.pi \* k \* n / N) / N

return y

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

climateData = np.fliplr(np.rot90(np.loadtxt(climateFile, usecols = (8, 10), skiprows = 2), k = 3))

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

EMNTRange = climateData[1][(-5 \* 12):] # Weather data for the last five years

allfourierCoeffs = DFT(EMNTRange)

realFourierCoeffs = allfourierCoeffs[:(len(allfourierCoeffs) // 2)]

kRange = np.arange(len(realFourierCoeffs))

#Plotting the original weather data and the original Fourier Coefficients

plt.clf()

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

plt.ylabel('Temperature (in decidegrees Centigrade)')

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 4 problem 1 plot 1.png')

plt.clf()

plt.plot(kRange, abs(realFourierCoeffs.real))

plt.ylabel('Fourier Coefficients')

plt.xlabel('k')

plt.title('Fourier Coefficients versus Frequency')

plt.savefig('Homework 4 Problem 1 plot 2.png')

#Finding the maximum k, above which all coefficients are small

kMax = 0

for k in kRange:

if abs(realFourierCoeffs[k].real) >= 200.0:

kMax = k

print 'kMax:', kMax

for k in kRange[kMax:]:

realFourierCoeffs[k] = 0.0

NewDateRange = [] #Using IDFT with half of the coefficients returns half of the data points - specifically, only the even terms.

for i in range(len(IntDateRange)):

if (i%2) == 0:

NewDateRange.append(IntDateRange[i])

newData = IDFT(realFourierCoeffs)

plt.clf()

plt.plot(NewDateRange, newData)

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

plt.ylabel('Temperature (in decidegrees Centigrade)')

plt.title('Original Data and Fourier Transformed Data \nof Extreme Minumum Monthly Temperatures')

plt.scatter(IntDateRange, EMNTRange)

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

*Results*

kmax: 16

(See attached graphs)

**Finding Roots**

*Code*

import math

import numpy as np

import matplotlib.pyplot as plt

from matplotlib import cm

def f1(x):

return x \* math.sin(x)

def f1prime(x):

return (math.sin(x) + x \* math.cos(x))

def NewtonRhapson(x0, tol, nmax, f, fprime):

error = 100.0

counter = 0

xGuesses = [x0]

while (error >= tol) & (counter <= nmax):

xGuess = xGuesses[counter] - (f(xGuesses[counter]) / fprime(xGuesses[counter]))

xGuesses.append(xGuess)

error = abs(xGuesses[counter] - xGuesses[counter + 1])

counter += 1

return xGuesses

def Bisection(x1, x2, tol, f):

xGuesses = []

while abs(x1 - x2) >= tol:

xNew = (x1 + x2) / 2

xGuesses.append(xNew)

if (np.sign(f(x1)) == np.sign(f(xNew))):

x1 = xNew

elif (np.sign(f(x2)) == np.sign(f(xNew))):

x2 = xNew

elif xNew == 0.0:

return xGuesses

return xGuesses

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

xGuesses = []

IterationDifference = abs(a - b)

while (IterationDifference >= tol):

xNew = (f(b) \* a - f(a) \* b) / (f(b) - f(a))

xGuesses.append(xNew)

if (np.sign(f(a)) == np.sign(f(xNew))):

IterationDifference = abs(a - xNew)

a = xNew

elif (np.sign(f(b)) == np.sign(f(xNew))):

IterationDifference = abs(b - xNew)

b = xNew

elif xNew == 0.0:

return xGuesses

return xGuesses

NewtonGuesses1 = NewtonRhapson(1.0, 0.5e-5, 10, f1, f1prime)

NewtonGuesses4 = NewtonRhapson(4.0, 0.5e-5, 10, f1, f1prime)

print 'It takes', len(Bisection(3.0, 4.0, 0.5e-5, f1)), 'iterations to get to the root of f(x) = x \* sin(x) between 3 and 4 using the Bisection method.'

print 'It takes', len(RegulaFalsi(3.0, 4.0, 0.5e-5, f1)), 'iterations to get to the root of f(x) = x \* sin(x) between 3 and 4 using the Regula Falsi method.'

xRange = np.linspace(-0.3, 4.7, 100.0)

yRange = []

for x in xRange:

yRange.append(f1(x))

yRange = np.asarray(yRange)

plt.clf()

plt.plot(xRange, yRange, label = 'f(x)', c = 'b')

plt.scatter(NewtonGuesses1, np.zeros(len(NewtonGuesses1)), label = 'x0 = 1.0', marker = 'o', c = 'r')

plt.scatter(NewtonGuesses4, np.zeros(len(NewtonGuesses4)), label = 'x0 = 4.0', marker = 's', c = 'g')

plt.title('Finding Roots via the Newton Method')

plt.xlabel('x')

plt.ylabel('f(x)')

plt.legend(loc = 0)

plt.savefig('Homework 4 Problem 2 Plot 1.png')

*Results*

It takes 18 iterations to get to the root of f(x) = x \* sin(x) between 3 and 4 using the Bisection method.

It takes 6 iterations to get to the root of f(x) = x \* sin(x) between 3 and 4 using the Regula Falsi method.

(See attached graphs)

**Fast Fourier Transform**

*Code*

import numpy as np

import matplotlib.pyplot as plt

import time

def timeMyFunction(func, inputData):

start = time.clock()

func(inputData)

end = time.clock()

return (end - start)

def DFT(Y):

N = len(Y)

c = np.zeros(N, dtype = np.complex\_)

for k in np.arange(N):

for n in np.arange(N):

c[k] += Y[n] \* np.exp((-2j \* np.pi \* k \* n) / N)

return c

def FFT(Y):

N = len(Y)

if N == 1:

C = Y

else:

Evens = FFT(Y[::2])

Odds = FFT(Y[1::2])

twiddle = np.exp(-2j \* np.arange(N) \* np.pi / N)

C = np.concatenate((Evens + twiddle[: N/2] \* Odds, Evens + twiddle[N/2 :] \* Odds)) # Since Odds is 1/2 the size of the twiddle array, need to concatenate arrays seperately.

return C

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

climateData = np.fliplr(np.rot90(np.loadtxt(climateFile, usecols = (8, 10), skiprows = 2), k = 3))

climateData1024 = climateData[1][-1024:]

dateRange1024 = climateData[0][-1024:]

fastFourierCoeffs = FFT(climateData1024)[:1024//2 + 1]

slowFourierCoeffs = DFT(climateData1024)[:1024//2 + 1]

mList = range(3, 26)

dftLimReached = False

fftLimReached = False

rfftLimReached = False

dftTimes = []

fftTimes = []

rfftTimes = []

for m in mList:

randomData = np.random.random(int(2\*\*m))

if dftLimReached == False:

t1 = timeMyFunction(DFT, randomData)

if t1 > 1.0:

dftLimReached = True

else:

dftTimes.append(np.log10(t1))

if fftLimReached == False:

t1 = timeMyFunction(FFT, randomData)

if t1 > 1.0:

fftLimReached = True

else:

fftTimes.append(np.log10(t1))

if rfftLimReached == False:

t1 = timeMyFunction(np.fft.rfft, randomData)

if t1 > 1.0:

rfftLimReached = True

else:

rfftTimes.append(np.log10(t1))

plt.clf()

plt.plot(range(513), fastFourierCoeffs.real, label = 'Real FFT coefficients')

plt.plot(range(513), slowFourierCoeffs.real, label = 'Real DFT coefficients')

plt.plot(range(513), fastFourierCoeffs.imag, label = 'Imaginary FFT coefficients')

plt.plot(range(513), slowFourierCoeffs.imag, label = 'Imaginary DFT coefficients')

plt.xlabel('k')

plt.ylabel('Coefficient')

plt.title('Real and imaginary coefficients for the FFT and DFT methods')

plt.legend(loc = 0)

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

plt.clf()

plt.plot(range(len(dftTimes)), label = 'DFT')

plt.plot(range(len(fftTimes)), fftTimes, label = 'FFT')

plt.plot(range(len(rfftTimes)), rfftTimes, label = 'rFFT')

plt.ylabel('log time (in log10(s))')

plt.xlabel('m')

plt.title('Time for each algorithm to process 2\*\*m random data points')

plt.legend(loc = 0)

plt.savefig('Homework 4 Problem 3 plot 2.png')

*Results*

(See attached graphs)