PHY 410
Homework Assignment 3

Han Wen

Person No. 50096432

September 19, 2014

Abstract

The goal of this assignment was to develop a thorough understanding of Fourier Transform and FFT.

Contents

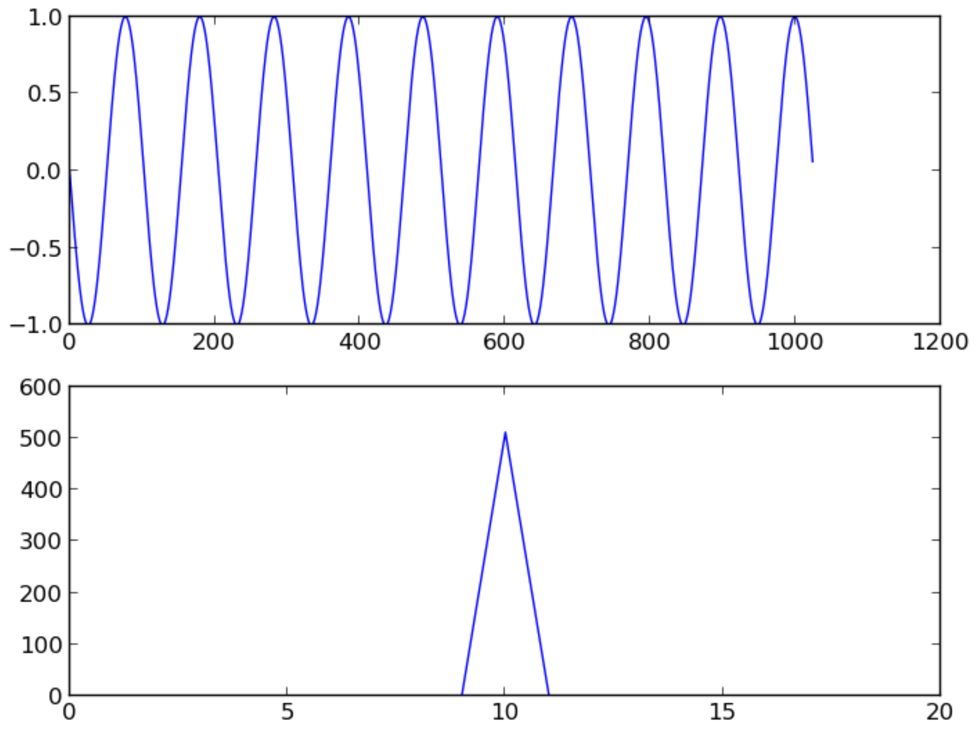

Figure 1: Problem 1.

1 Problem 1

1.1 Description

Run the Lecture 6 fft test code and interpret the power spectrum. What is the period of the oscillations? What N does this correspond to? How do you convert?

1.2 Solution

Here is the result plotting 1 . We can see from the original code or from the power we can see the frequency is 10. And the function of the plot is $y = sin(-\frac{2\pi f x}{N})$.

Therefore period $T = \frac{N}{f} = 102.4$

2 Problem 2

2.1 Description

We analyzed the sunspot number and total solar irradiance data sets in Lecture 8. Repeat this exercise with the C02 data from Lecture 6. "Clean up" the jitters by taking the FFT, performing a manipulation, and inverse FFT, and plot the cleaned spectrum in the "time" domain again. Plot the power spectra of the data sets.

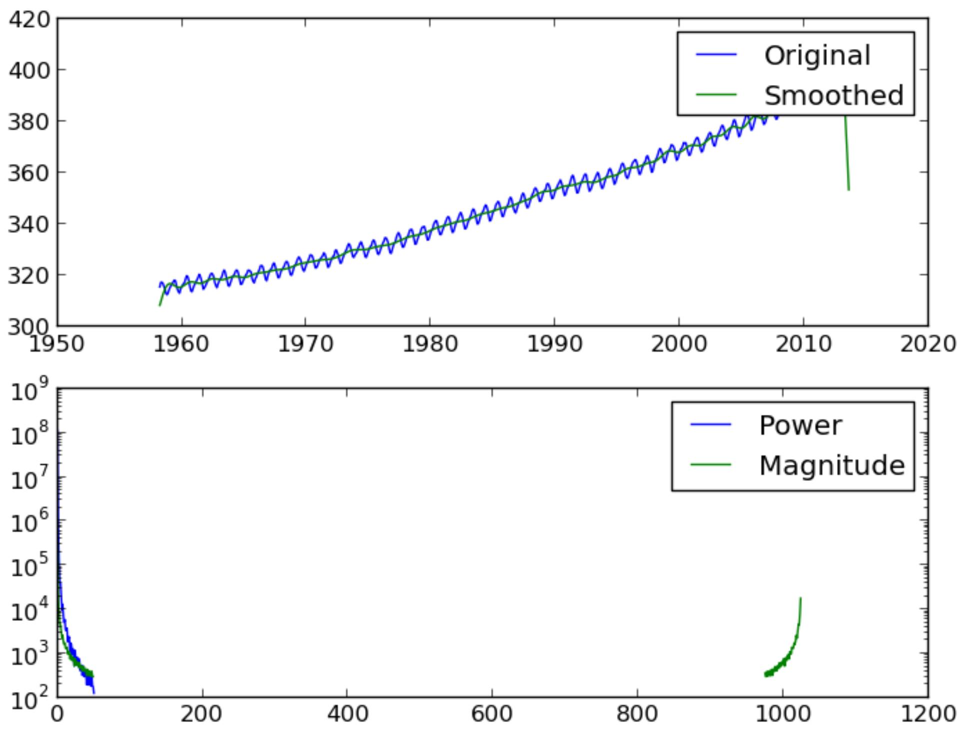

Figure 2: Problem 2 with paddle 300 and maxfreq 50

2.2 Result

By setting the paddle number to be 300 and make the maxfreq=50, I manage to have the result picture: 2.

3 Problem 3

3.1 Description

Subtract a linear fit of the Mauna Loa CO data from Lecture 6, and present the resulting Fourier transform. Further "clean up" the jitters by applying the "zeroing" method from class, and then plot the cleaned spectrum in the "time" domain, adding back the linear trend you subtracted earlier.

3.2 Result

By doing the required steps with paddle value 0 and max frequency 50, I managed to get the result plot: 3.

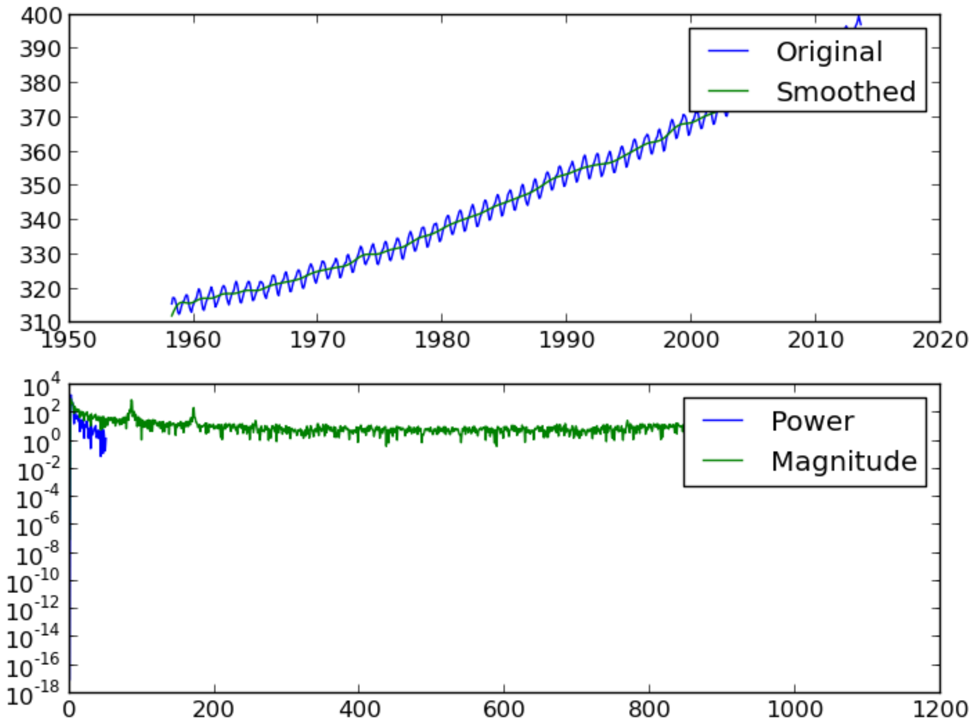

Figure 3: Problem 3 combined with linear fits

Acknowledgements

I discussed this assignment with my classmates and used material from the cited references, but this writeup is my own.

A Appendix

A.1 python code

The following python code was used to obtain the results in this report:

```
import matplotlib.pyplot as plt

from fft import fft
from numpy import array
import math

plotfirst = False

if plotfirst == True :
    # make some fake data :

    N = 1024
    f = 10.0

    x = array([ float(i) for i in xrange(N) ] )
    y = array([ math.sin(-2*math.pi*f* xi / float(N))  for xi in x ])
    #y = array([ xi for xi in x ])
    Y = fft(y)
#   yreal=[yi.imag for yi in Y]
    Yre = [math.sqrt(Y[i].real**2 + Y[i].imag**2) for i in xrange(N)]

    s1 = plt.subplot(2, 1, 1)
    plt.plot( x, y )

    s2 = plt.subplot(2, 1, 2)
    s2.set_autoscalex_on(False)
    plt.plot( x, Yre )
    plt.xlim([0,20])

    plt.show()


else :
    # data downloaded from ftp://ftp.cmdl.noaa.gov/ccg/co2/trends/co2_mm_mlo.txt
    print ' C02 Data from Mauna Loa'
    data_file_name = 'co2_mm_mlo.txt'
    file = open(data_file_name, 'r')
    lines = file.readlines()
    file.close()
    print ' read', len(lines), 'lines from', data_file_name

    yinput = []
    xinput = []
    for line in lines :
        if line[0] != '#' :
            try:
                words = line.split()
                xval = float(words[2])
                yval = float( words[4] )
                yinput.append( yval )
                xinput.append( xval )
            except ValueError :
                print 'bad␣data:',line

    y = array( yinput[0:512] )
    x = array([ float(i) for i in xrange(len(y)) ] )
    Y = fft(y)

    Yre = [math.sqrt(Y[i].real**2+Y[i].imag**2) for i in xrange(len(Y))]


    plt.subplot(2, 1, 1)
    plt.plot( x, y )

    plt.subplot(2, 1, 2)
    plt.plot( x, Yre )
    plt.yscale('log')


    plt.show()

import matplotlib.pyplot as plt

from fft import fft, fft_power, ifft
from numpy import array, real
import math
import time


# data downloaded from ftp://ftp.cmdl.noaa.gov/ccg/co2/trends/co2_mm_mlo.txt
print '␣C02␣Data␣from␣Mauna␣Loa'
data_file_name = 'co2_mm_mlo.txt'
file = open(data_file_name, 'r')
lines = file.readlines()
file.close()
print '␣read', len(lines), 'lines␣from', data_file_name

window = False

yinput = []
```

```
xinput = []

for line in lines :
    if line[0] != '#' :
        try:
            words = line.split()
            xval = float(words[2])
            yval = float( words[4] )
            yinput.append( yval )
            xinput.append( xval )
        except ValueError :
            print 'bad␣data:',line
yb=yinput           #original data



#fft part
N = len(yinput)
log2N = math.log(N, 2)
if log2N - int(log2N) > 0.0 :
    print 'Padding␣with␣zeros!'
    pads = [300.0] * (pow(2, int(log2N)+1) - N)
    yinput = yinput + pads
    N = len(yinput)
    print 'Padded␣:␣'
    print len(yinput)
    # Apply a window to reduce ringing from the 2^n cutoff
    if window :
        for iy in xrange(len(yinput)) :
            yinput[iy] = yinput[iy] * (0.5 - 0.5 * math.cos(2*math.pi*iy/float(N-1

y = array( yinput )
x = array([ float(i) for i in xrange(len(y)) ] )
Y = fft(y)

maxfreq = 50
# Now smooth the data
for iY in range(maxfreq, len(Y)-maxfreq ) :
    Y[iY] = complex(0,0)
    #Y[iY] = Y[iY] * (0.5 - 0.5 * math.cos(2*math.pi*iY/float(N-1)))

    #for iY in range(0,N) :
    #    Y[iY] = Y[iY] * math.exp(-1.0*iY / 50.0)

powery = fft_power(Y)
powerx = array([ float(i) for i in xrange(len(powery)) ] )

Yre = [math.sqrt(Y[i].real**2+Y[i].imag**2) for i in xrange(len(Y))]
```

```
ysmoothed = ifft(Y)
ysmoothedreal = real(ysmoothed)
youtput=ysmoothedreal[0:len(xinput)]
ax1 = plt.subplot(2, 1, 1)
#p1, = plt.plot( x, y )
#p2, = plt.plot( x, ysmoothedreal )
p1, = plt.plot( xinput, yb )
p2, = plt.plot( xinput, youtput)
ax1.legend( [p1,p2], ['Original', 'Smoothed'] )

ax2 = plt.subplot(2, 1, 2)
p3, = plt.plot( powerx, powery )
p4, = plt.plot( x, Yre )
ax2.legend( [p3, p4], ["Power", "Magnitude"] )
plt.yscale('log')


plt.show()

import matplotlib.pyplot as plt

from fft import fft, fft_power, ifft
from numpy import array, real
import math
import time


# data downloaded from ftp://ftp.cmdl.noaa.gov/ccg/co2/trends/co2_mm_mlo.txt
print ' C02 Data from Mauna Loa'
data_file_name = 'co2_mm_mlo.txt'
file = open(data_file_name, 'r')
lines = file.readlines()
file.close()
print ' read', len(lines), 'lines from', data_file_name

window = False

yinput = []
xinput = []

for line in lines :
    if line[0] != '#' :
        try:
            words = line.split()
            xval = float(words[2])
            yval = float( words[4] )
            yinput.append( yval )
            xinput.append( xval )
        except ValueError :
            print 'bad data:',line
yb=yinput             #original data




#least squares fit module
def least_squares_fit(x, y):
    """Perform a least-squares fit to data (x,y)

    Args :
       x : x values
       y : y values

    Returns :
       a : intercept
       b : slope
       sigma : total uncertainty (sqrt(variance/(n-2)))
       sigma_a : uncertainty on a
       sigma_b : uncertainty on b

    """
    n = len(x)
    s_x  = sum(x)
    s_y  = sum(y)
    s_xx = sum(x_i**2 for x_i in x)
    s_xy = sum(x[i]*y[i] for i in range(n))
    denom = n * s_xx - s_x**2
    if abs(denom) > 0.00001 :
        a = (s_xx * s_y - s_x * s_xy) / denom
        b = (n * s_xy - s_x * s_y) / denom
        variance = sum((y[i] - (a + b*x[i]))**2 for i in range(n))
        sigma = math.sqrt(variance/(n-2))
        sigma_a = math.sqrt(sigma**2 * s_xx / denom)
        sigma_b = math.sqrt(sigma**2 * n / denom)
        return [a, b, sigma, sigma_a, sigma_b]
    else :
        print 'error : divided by zero!'
        return None

#Then perform least square fit and modify the data then do fft
fit = least_squares_fit(xinput,yinput)
print ' least_squares fit to data1:'
print ' slope b = {0:6.3f} +- {1:6.3f}'.format( fit[1], fit[4])
print ' intercept a = {0:6.3f} +- {1:6.3f}'.format( fit[0], fit[3])
print ' log_10(N) error bar = {0:6.3f}'.format( fit[2] )
```

```
modiy = list((yinput[i]-xinput[i]*fit[1]-fit[0]) for i in range(len(yinput)))
for i in range(100):
    print modiy[i]

#fft part
N = len(modiy)
log2N = math.log(N, 2)
if log2N - int(log2N) > 0.0 :
    print 'Padding with zeros!'
    pads = [0.0] * (pow(2, int(log2N)+1) - N)   #now paddle with 0s
    modiy = modiy + pads
    N = len(modiy)
    print 'Padded : '
    print len(modiy)
    # Apply a window to reduce ringing from the 2^n cutoff
    if window :
        for iy in xrange(len(modiy)) :
            modiy[iy] = modiy[iy] * (0.5 - 0.5 * math.cos(2*math.pi*iy/float(N-1))

y = array( modiy )
x = array([ float(i) for i in xrange(len(y)) ] )
Y = fft(y)

Yre = [math.sqrt(Y[i].real**2+Y[i].imag**2) for i in xrange(len(Y))]

maxfreq = 50
# Now smooth the data
for iY in range(maxfreq, len(Y)-maxfreq ) :
    Y[iY] = complex(0,0)
    #Y[iY] = Y[iY] * (0.5 - 0.5 * math.cos(2*math.pi*iY/float(N-1)))

    #for iY in range(0,N) :
    #    Y[iY] = Y[iY] * math.exp(-1.0*iY / 50.0)

powery = fft_power(Y)
powerx = array([ float(i) for i in xrange(len(powery)) ] )

#Yre = [math.sqrt(Y[i].real**2+Y[i].imag**2) for i in xrange(len(Y))]

ysmoothed = ifft(Y)
ysmoothedreal = real(ysmoothed)
yo=ysmoothedreal[0:len(xinput)]
youtput=list(yo[i]+xinput[i]*fit[1]+fit[0] for i in range(len(xinput)))
```

```
ax1 = plt.subplot(2, 1, 1)
#p1, = plt.plot( x, y )
#p2, = plt.plot( x, ysmoothedreal )
p1, = plt.plot( xinput, yb )
p2, = plt.plot( xinput, youtput)
ax1.legend( [p1,p2], ['Original', 'Smoothed'] )

ax2 = plt.subplot(2, 1, 2)
p3, = plt.plot( powerx, powery )
p4, = plt.plot( x, Yre )
ax2.legend( [p3, p4], ["Power", "Magnitude"] )
plt.yscale('log')


plt.show()
```