PHY 410
Homework Assignment 2

Han Wen

Person No. 50096432

September 10, 2014

Abstract

The goal of this assignment was to use numerical method to analyse some earth quake data and those regarding global warming as well.

Contents

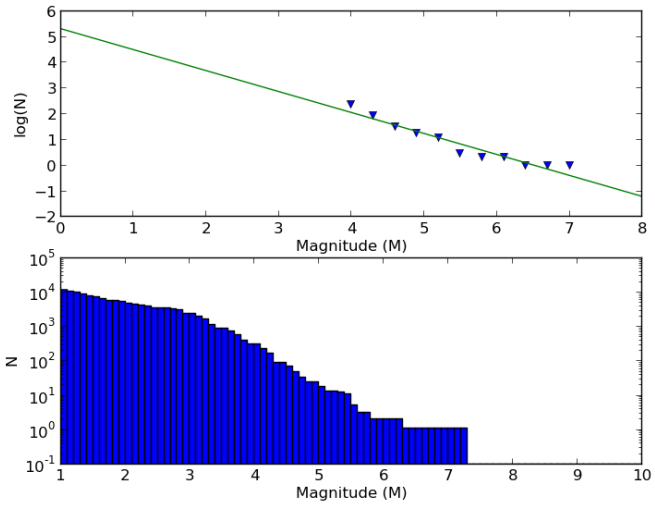

Figure 1: The earthquake using CA data.

1 Problem 1

1.1 Description

Download the NEIC Global data set of earthquakes with magnitudes greater than 1.0 and use a quake.cpp code to fit the Gutenberg Richter Law. The raw data do not provide a reasonable fit. Explain why and fix the code to obtain a more reliable estimate of the slope constant . Check your answer with values given in the references from class[1]

1.2 Numerical Analysis

The Richter scale was developed by Richter in the 1930s, relates the local magnitude scale M_L defined by the amount of amplitude variation on a seismograph [3]. In the 1970's, the moment magnitude scale was introduced given by $M_0 = muSD$. Additionally, the Gutenberg-Richter Model gives the relationship of the frequency(N) of the earthquakes and the corresponding magnitude of them. The frequency defined as the number of events with magnitude greater or equal to M, namely the Empirical model: $logN = a - bM$ Therefore we are going to do a linear fits with $logN$ and M and obtain the slope.

1.3 Results

Noticing the original code didn't choose the data homogeneously and apparently those with smaller magnitude and bigger magnitude has different slope, considering the bigger the earthquake is the more concern we have about it, we are going to only use those have magnitudes bigger or equal to 4 and choose points homogeneously in that interval, then plot and calculate the slope. Here is the plot 1 , the slope will be $b = -0.814 \pm 0.076$, compared to the value about -1 in the article [1], it's a little smaller and considering we only use the data from last 3 years and in specifically California area, the discrepancy is accepted.

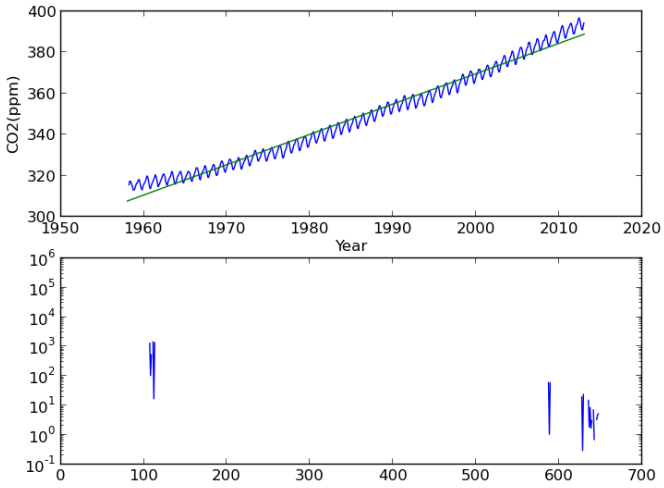

Figure 2: the CO2 with average.

2 Problem 2

2.1 Description

Download the NOAA Mauna Loa CO data set (ALSO included in the lecture 6 example code, if the link is not working) and modify the co2.cpp or co2.py code to generate graphs using other columns in the data. Does the average rate of increase depend on which colums are used in a straight line fit? Explore linear models with 3 terms to decide whether the linear rate of increase is accelerating or decelerating. If the current rate of increase continues indefinitely, when will the concentration become toxic (50,000 ppm)?

2.2 Result

Using the average value, we have the slope $b = 1.474 \pm 0.009$, with the plot 2

Using the interpolated value, we have the slope $b = 1.470 \pm 0.009$, with the plot 3

Using the trend value, we have the slope $b = 1.472 \pm 0.007$, with the plot 4.

So we can see the average rate of increase basically remains the same. With the data of average value, I did a linear fits using three different interval of data, namely the first part 1958-1976 middle part 1976-1995 and last part 1995-2014, and the average rates of increase are 0.95 1.53 1.95; for average one it's 0.95,1.51,1.94; for trend one it's 0.95,1.52,1.95. So the rate of increase is increasing. With this trend, after 25411 years it will become toxic.

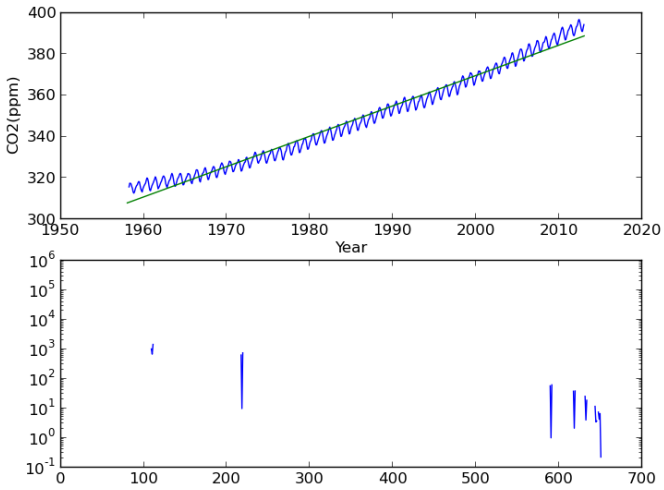

Figure 3: the CO2 with interpolated.

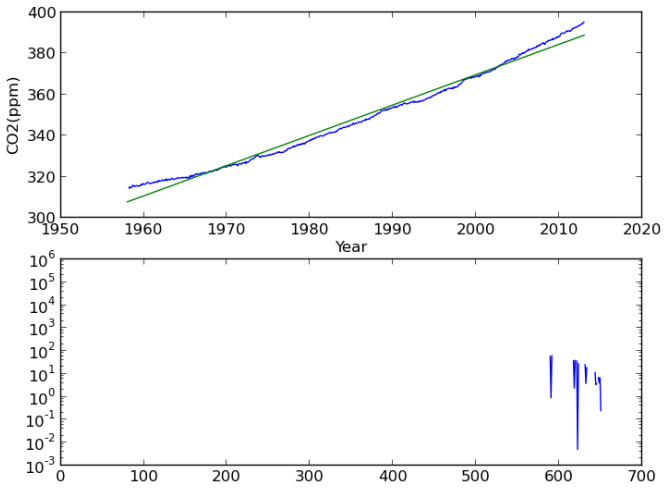

Figure 4: the CO2 with trend.

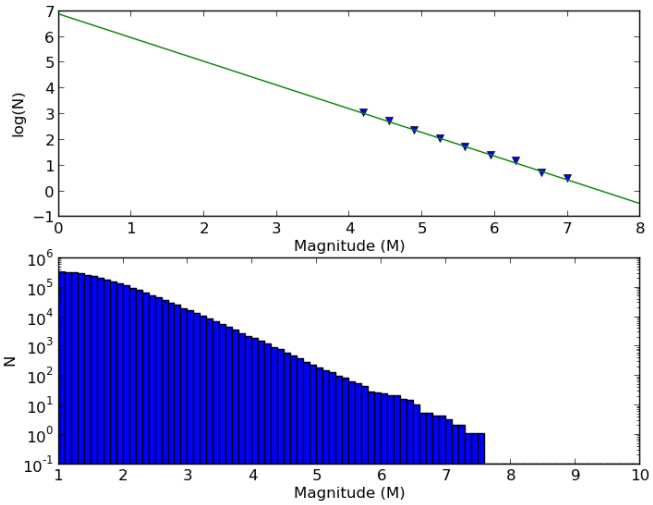

Figure 5: Earthquake 1932-1999

3 Problem 3

3.1 Description

Earthquake Problem: Find a different event data set (e.g. from the Southern California Earthquake Center http://www.scec.org/resources/data/, or any other source you can find on the Web), perform a linear fit and compare your results with the NEIC global set.

Global Warming Problem: Download a NASA Global Temperature Anomaly data set for a similar time period as the NOAA Mauna Loa data and analyze the trend. Can you conclude from your analysis that the two sets are correlated in any way?

3.2 Numerical analysis

For earthquake, I got the data from National Climatic Data Center [4], from year 1932 to 1999

For CO2, I got data from Earth System Research Laboratory [5] from 1969 to 2005

3.3 Result

For earthquake: the calculated slope is $b = -0.921 \pm 0.018$, with the plot 5, we can see it's more close to the accepted value in that article.

For CO2: With those data the calculated slope is $b = 1.655 \pm 0.011$, with the plot 6, the number is bigger than the above ones but not too much, probably because the sample is smaller and in the earlier time this data is incomplete. However, the trend is very similar, the general trend is increasing but will vibrate locally.

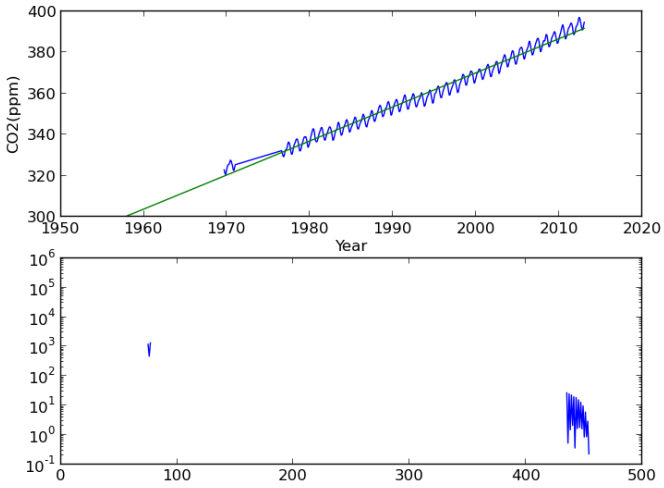

Figure 6: CO2 1969-2005

Acknowledgements

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

References

[1] H. Kanamori and E.E. Brodsky, The Physics of Earthquakes, Physics Today 54, 34-40 (2001)http://scitation.aip.org/content/aip/magazine/physicstoday/article/54/6/10.1063/1.1387590

[2] PHY 410-505 Webpage, http://www.physics.buffalo.edu/phy410-505.

[3] a brief introduction to seismographs and seismograms,http://www.colorado.edu/physics/phys2900/homepages/Marianne.Hogan/graphs.html.

[4] National Climatic Data Center data base ftp://ftp.ncdc.noaa.gov/pub/data/anomalies/anomalies/usingGHCNMv2/

[5] Mauna Loa, Hawaii, United States(MLO), monthly averages http://www.esrl.noaa.gov/gmd/dv/data/index.php?site=mlo&category=Greenhouse%2BGases&parameter_name=Carbon%2BDioxide

A Appendix

A.1 python code

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

```
import math
import matplotlib.pyplot as plt
import numpy as np

def sine_transform(data):
    """Return Fourier sine transform of a real data vector"""
    N = len(data)
    transform = [ 0 ] * N
    for k in range(N):
        for j in range(N):
            angle = math.pi * k * j / N
            transform[k] += data[j] * math.sin(angle)
    return transform

file_name = "co2_mm_mlo.txt"
file = open(file_name, "r")
lines = file.readlines()
file.close()

dates = []
data = []
for line in lines:
    if len(line) > 4:
        try:
            year = int(line[0:4])
            if year > 1957 and year < 2013:
                words = str.split(line)
                dates.append(float(words[2]))
                ppm = float(words[4])
                if ppm > 0: data.append(ppm)
        except ValueError:
            pass

print "␣read", len(data), "values␣from", file_name


transform = sine_transform(data)

freqs = [ float(i) for i in xrange(len(transform))]

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

whn=int(len(dates)/3)
fit = least_squares_fit(dates, data)
print ' least_squares fit to data:'
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] )
slope=fit[1]
intercept=fit[0]

fit = least_squares_fit(dates[0:whn], data[0:whn])
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] )

fit = least_squares_fit(dates[whn:2*whn], data[whn:2*whn])
print ' least_squares fit to data2:'
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] )

fit = least_squares_fit(dates[2*whn:3*whn], data[2*whn:3*whn])
print ' least_squares fit to data3:'
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] )

plt.subplot(2, 1, 1)
plt.plot( dates, data )
plt.xlabel('Year')
plt.ylabel('CO2(ppm)')
t=np.arange(1958,2013,0.01)
plt.plot(t,slope*t+intercept)


ax = plt.subplot(2, 1, 2)
plt.plot( freqs, transform )
ax.set_yscale('log')

plt.show()

import math
import matplotlib.pyplot as plt
import numpy as np

# define a function to linear fit x-y data without error bars
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



"""Plot data for the Gutenberg-Richter Model.

Here, we plot the curve of the number of earthquakes
greater than magnitude M, for each M value.

So, we loop over the earthquakes, and store the
frequency of each magnitude. At the end of the loop,
we compute the cumulative distribution such that the
value at magnitude M will be the integral of the frequency
distribution for >= M. This is what the Gutenberg-Richter
Model predicts.

"""


# data downloaded from http://earthquake.usgs.gov/earthquakes/search/
print ' Earthquake data: Gutenberg-Richter Model'
data_file_name = 'california_earthquakes_2010_to_2013.csv'
file = open(data_file_name, 'r')
lines = file.readlines()
file.close()
print ' read', len(lines), 'lines from', data_file_name

# store event data in two ways for demonstration
# 1. in a python dictionary object with
#    key = magnitude starting in column 50
#    value = number of events with this magnitude
# 2. keeping a "tuple" of the results for later usage with matplotlib
histogram = dict()
magvalues = []
for line in lines:
    if line[0] != 't' :
        try:
```

```
            words = line.split(',')
            [latitude,longitude,depth,mag] = [float(s) for s in words[1:5] ]
            magvalues.append( mag )
            # For debugging :
            #print 'read : {0:s} , ({1:6.3f},{2:6.3f}), d = {3:4.1f}, m = {4:6.3f}
            #     words[0], latitude, longitude, depth, mag
            #     )
            histogram[mag] += 1
        except KeyError :
            histogram.setdefault(mag, 1)
        except ValueError:
            print 'bad data:', line

num_events = sum(histogram[M] for M in histogram.keys())
num_bins = len(histogram)
print ' stored', num_events, 'events in', num_bins, 'bins'

# x data = M values sorted in increasing order
# y data = log_10(N) where N = number of events with magnitude >= M
M_values = sorted(histogram.keys())
dN_values = [histogram[M] for M in M_values]
log10N_values = [ math.log10(sum(dN_values[i:]))
                  for i in range(len(M_values)) ]
#to obtain a backwards sequence list
mb_values=M_values[::-1]
print mb_values
#obtain the maximum and minimum
maxint=int(mb_values[0])
minint=int(M_values[0])
h=(maxint-minint)/20.0                  #step length, totally number is 20
#newm=[7,6.5,6,5.5,5,4.5,4,3.5,1]
newm = [maxint-i*h for i in range(21)]     # new selected m
#the loop to generate new x,y components
newlogn=[]                                      # corresponding new n


for k in range (len(newm)):
    for j in range(len(mb_values)):
        if newm[k]>=mb_values[j]:
            break
    newlogn.append( log10N_values[-j] )
print newlogn,newm
#The fitting must start from 4
lenm=len(newm)
for m in range(lenm):
    if newm[m] < 4.0:
        for mp in range (m,lenm):
            del newm[m]
```

```
            del newlogn[m]
        break
print newm, newlogn


#print 'log10N_values is '
#for i in xrange(len(log10N_values)) :
#    print str(M_values[i]) + '  ' + str(log10N_values[i])
# First plot our "home-grown" values with our least-squares
# fit in place.
fit = least_squares_fit(newm, newlogn)
print ' least_squares fit to data:'
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] )
plt.subplot( 2, 1, 1)
#plt.plot( M_values, log10N_values, 'v')
plt.plot( newm, newlogn, 'v')
plt.xlabel( 'Magnitude (M)' )
plt.ylabel( 'log(N)' )
t=np.arange(0,8,0.001)
plt.plot(t,fit[1]*t+fit[0])


# Next also plot matplotlib's version of the same thing.
plt.subplot( 2, 1, 2)
plt.hist( magvalues, bins=90, range=[1.0,10.0], log=True, bottom=0.1,cumulative=-1
plt.xlabel( 'Magnitude (M)' )
plt.ylabel( 'N' )

# perform a least square fit
#fit = least_squares_fit(M_values, log10N_values)


plt.show()
```