Principles of Scientific Computing Homework 5

Michal Jander, Yevginiy Krasnitskiy, Jordan Palamos, and Andrea Yocom

May 1, 2014

Contents

1 Instructions

1. Get the data file
2. Average the 3 months and differentiate this curve in 5 year intervals. Plot the resulting slope vectors. Use a finite element approach or another way.
3. Use a numerical integration technique to compute the total area of the curve from 1870 to 1950 - compare that to the area under the curve from 1950 to 2013
4. Smooth the waveform in three ways, plot the three waveforms on the same graph and comment on differences in smoothing. Use the following methods:

- boxcar of width 5 years
- gaussian kernel of width 5 years
- exponential smoothing with greatest weight given to last 20 years

5. Plot histogram of yearly ratios of July and Sept sea ice extent, comment on patterns

6. Window the data and baseline (polynomial fit ok) to extract two cooling periods prior to 1950 that allow sea extent to remain larger than average. Determine the total area of each event, compare to average area from period 1870 to 1950 to determine overall amplitude of cooling.

7. When is meltdown? Use two sets of data: a) entire data set, determine smooth functional form that best fits data, extrapolate to zero. Do not use polynomial fit in this case. Graph fitted line to the data, extrapolated to zero. b) use only September satellite data. Fit linear regression to the data, also some power or exponential law. Plot two fits on same graph.

2 Plotting, averaging, and finding the rate of change of Arctic Sea ice extent

2.1 General trends

Figure 1, the 1870-2013 Arctic Sea ice extent averaged over the summer months (July, August, and September), illustrates that the arctic sea ice has dropped precipitously in the last 20 years, as compared to the approximately stable extent for the one hundred years preceding this. If we look at the months separately (as in Fig. 2), we see that the general trend is for July extent to be about one square kilometer above the average, the August extent to be about average, and the September extent about one square kilometer below the average.

2.2 Sea ice extent change from 1870 to 1950 and from 1950 to 2013

Averaging the summer month ice extent in five year bins as in Fig. 3 smooths things out a bit. Differentiating the curve shown in Fig. 3 in five year intervals yields a rate of change profile, shown in Fig. 4. While pre-1950 ice extent changes seem to be roughly evenly distributed about zero, it is clear that after 1950 the ice extent changes begin trending steadily toward ice loss, with the most dramatic loss from 2000-2005. Integrating under the curve shown in Fig. 4 gives us a value for the total change in extent. From 1870 to 1950, average ice extent increased by $0.166km^2$. From 1950 to 2013, average ice extent decreased by $3.055km^2$.

We can plot this slope in a slightly different way. In Fig. 5, the slope is not plotted as a flat line. Numerically integrating under this curve using Simpson's rule yields different results. Here, from 1870 to 1950, average ice extent increased by $0.03km^2$. From 1950 to 2013, average ice extent decreased by $0.264km^2$. The ratio of post-1950 loss to pre-1950 gain is 9.08. This is about a ten-fold reduction. It seems that we are losing ten times

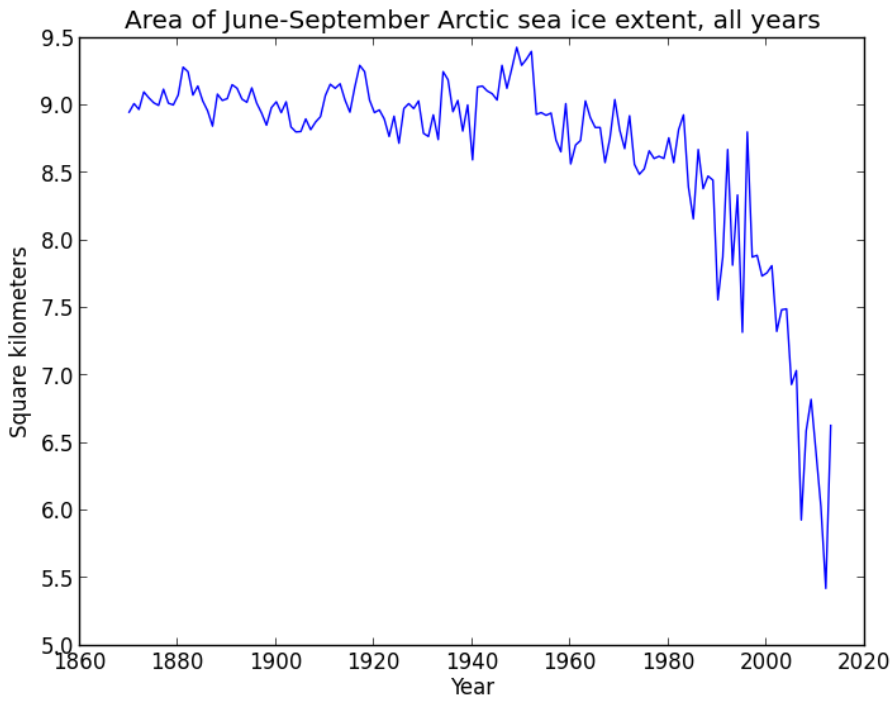


Figure 1: Here you see a precipitous arctic sea ice extent drop starting in about 1980

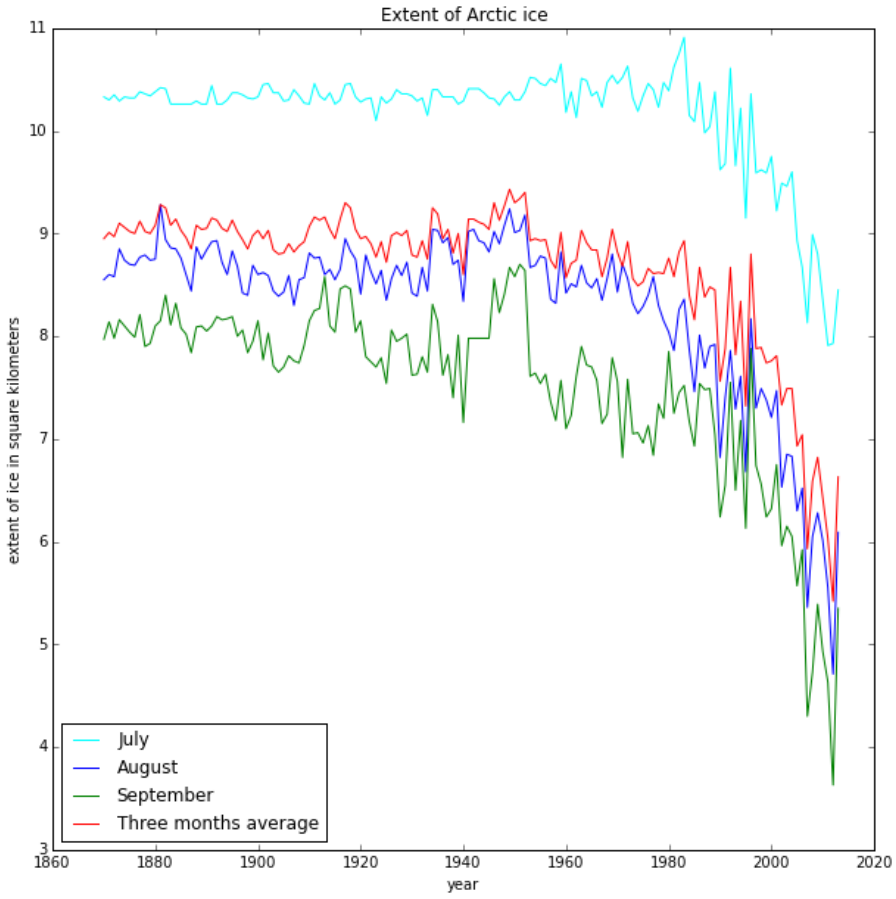


Figure 2: July above average, August is average, and September is below average

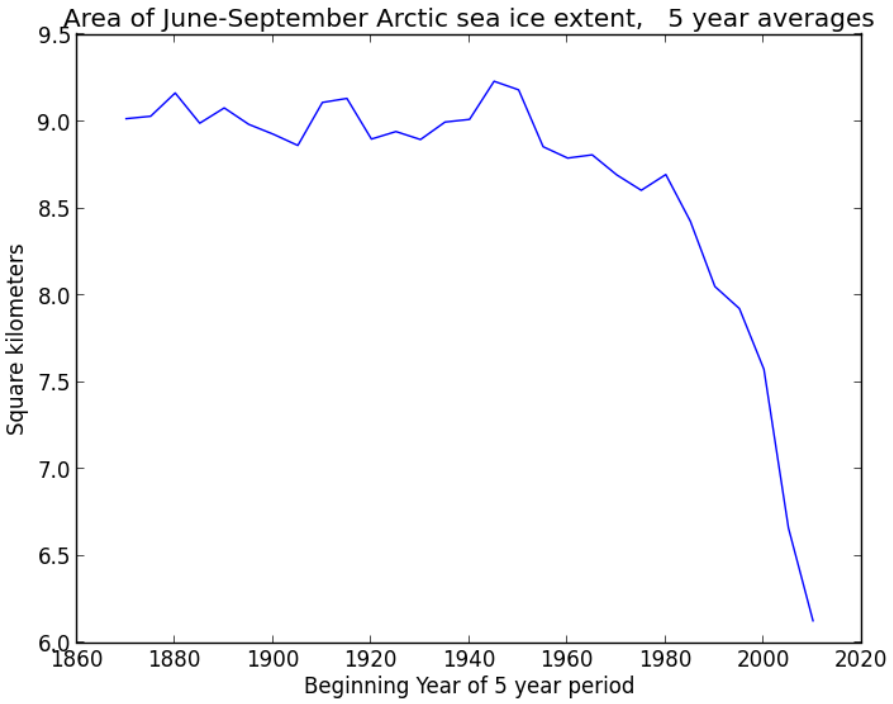


Figure 3: A smoother version of sea ice extent loss

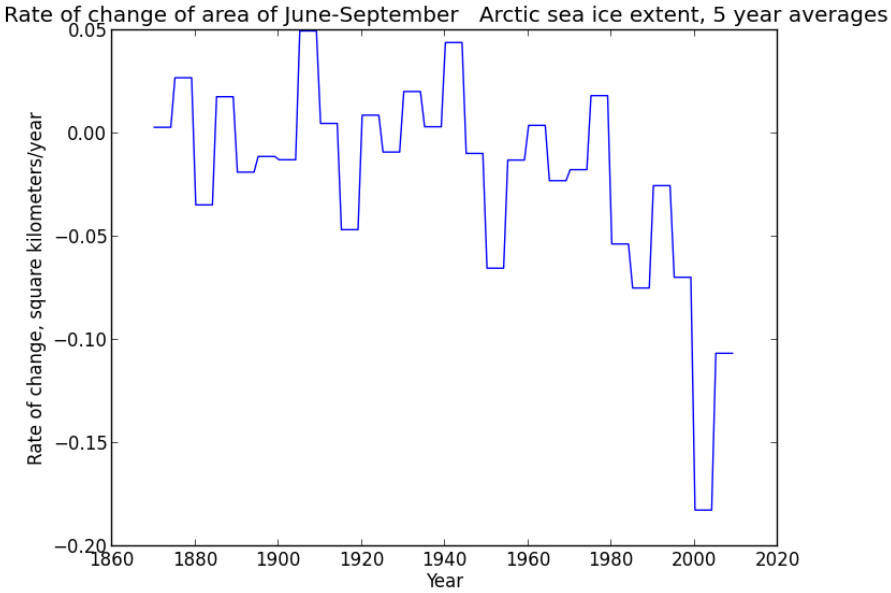


Figure 4: Each five year stretch has some average rate of change in ice extent.

more ice in the most recent 60 years than we've been able to accumulate in the eighty years prior to that.

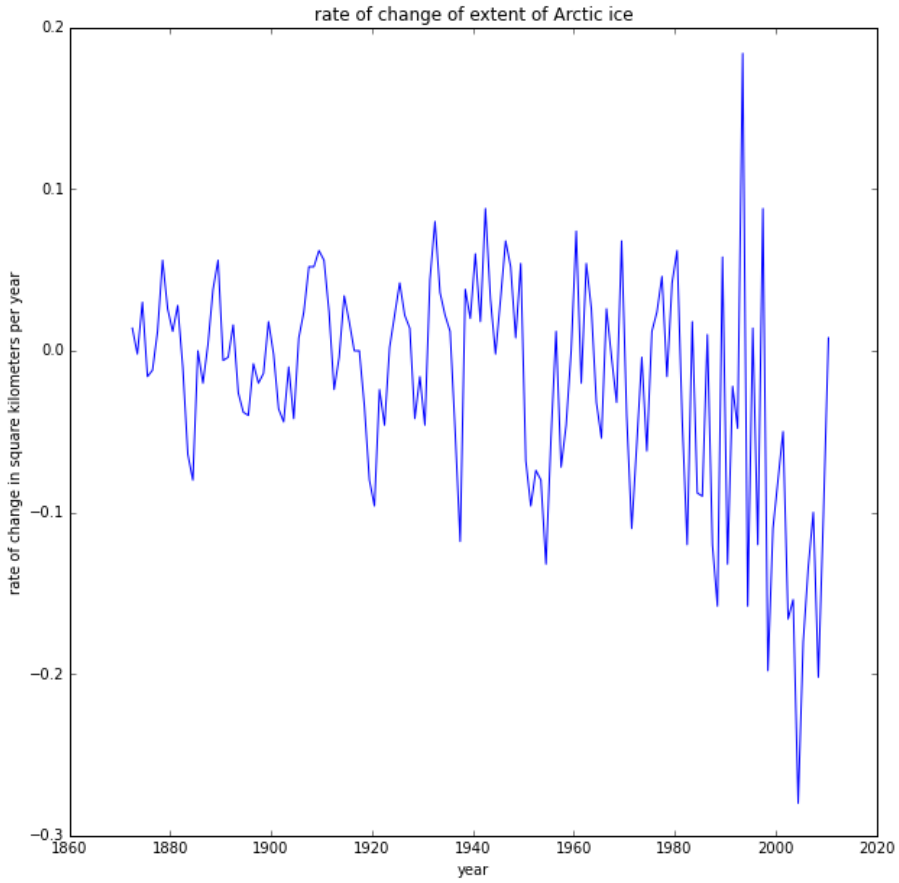


Figure 5: A different way to look at the slope.

3 Smoothed sea ice extent data

We use boxcar, Gaussian, and exponential smoothing techniques on the average ice data. The boxcar algorithm takes neighboring points and computes their average and replaces the middle point by the average. We use five points with reflection on the endpoints (i.e. use the points near the endpoint reflected about the center point.

The Gaussian algorithm replaces each point by the weighted average of the point and its neighbors, the weights being Gaussian distributed about the center point, with total of unity. We use five points, with weights of $(0.067, 0.242, 0.383, 0.242, 0.067)$.

The exponential smoothing algorithm works according to Eqn. 1, where $x(t)$ is the raw data, $s(t)$ is the smoothed data, and $s(0) = x(0)$:

$$s(t) = a * x(t-1) + (1-a) * s(t-1). \tag{1}$$

The smoothing parameter $0 < a < 1$ works as follows: Larger values of a give more weight to recent data but smoothing is worse. We use $a = 0.7$ to give greater weight to the more recent data but to keep the smoothing as good as possible.

All methods, shown in Fig. 6, suppress local variations in favor of highlighting global variations. The boxcar and Gaussian smoothing techniques tend to be similar, and both methods lag the real data temporally. The exponential smoothing, on the other hand, leads the data temporally. It follows the raw data more closely than the boxcar or Gaussian methods, particularly the recent data. This was the intention of choosing a smoothing parameter close to unity. The fit is not as smooth as Gaussian or boxcar fits, but shows local variations better with reduced noise.

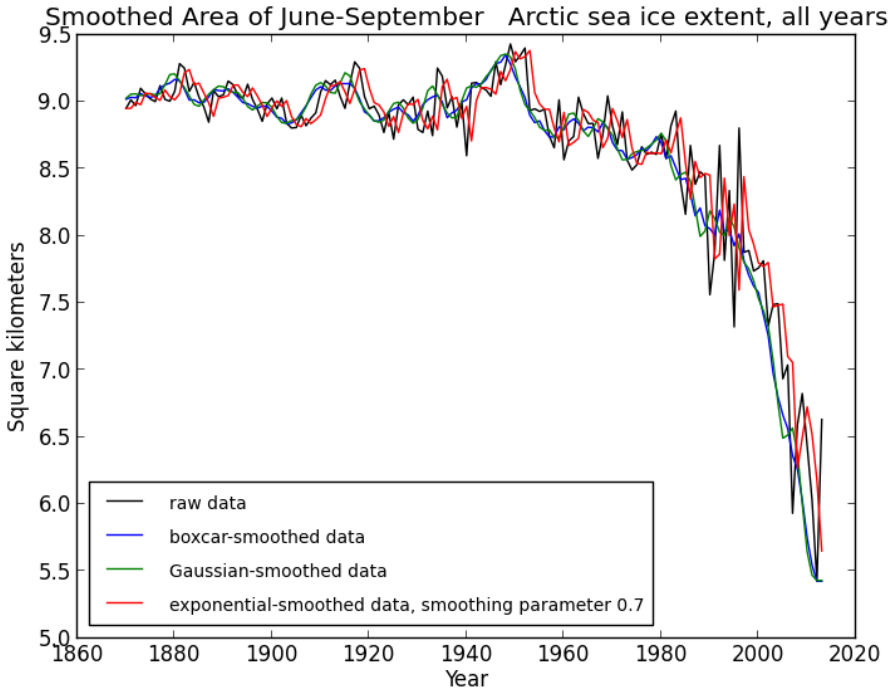


Figure 6: Boxcar, Gaussian, and Exponential Smoothing comparisons

To summarize the effects of the three smoothing methods:

- Boxcar-smoothed data follows the global behavior of the raw data, but loses the local variations in the data. It is the least noisy of the three.
- Gaussian-smoothed data shows more of local variations than does boxcar-smoothed data, but the variations are not sharp and tend to be normally-smoothed-over.
- Exponentially-smoothed data follows the local variations in the data much better, especially for the more recent data as was intended with the smoothing parameter, however the curve is not as smooth, still showing signs of noise.

4 Histogram of summer melting events

Taking the ratio of July to September ice extent gives a measure of how much ice melted over the summer. The distribution, shown in Fig. 7 looks roughly Poissonian. The long tail may indicate some random events of large mass melting. The statistical significance of the large melting events is very small, however may still fall into the distribution of melting events that is depicted in the histogram.

5 Pre-1950 cooling events

5.1 Attempt 1

Looking at the events that have occurred prior to year 1950 we can find two windows of cooling periods for which the ice extent was above normal, shown in Fig. 8. It looks like the first cooling period begins in 1933 and ends in 1936, and the second cooling period begins in 1945 and ends in 1953. We can compute the area under the gradient curves over these cooling periods and find the ratio to the total change in the sea ice extent (computed up to 1950). Using the first window from 1933 to 1936 and the second window from 1945 to 1950 (cooling periods) we find:

- change in extent of sea ice in first cooling window = 0.0453333333333 square kilometers
- change in extent of sea ice in the second cooling window = 0.164666666667 square kilometers
- ratio in first cooling window to total change prior to 1950 = 0.155873925501
- ratio in second cooling window to total change prior to 1950 = 0.566189111748

5.2 Attempt 2

If we consider the window from 1910-1920 we get a cooling magnitude of: 6.94 compared to an average cooling of: 2.46 for a relative cooling of: 4.48. If we consider the window from 1940-1950 we get a cooling magnitude of: 3.250 compared to an average cooling of: 1.410 for a relative cooling of: 1.84.

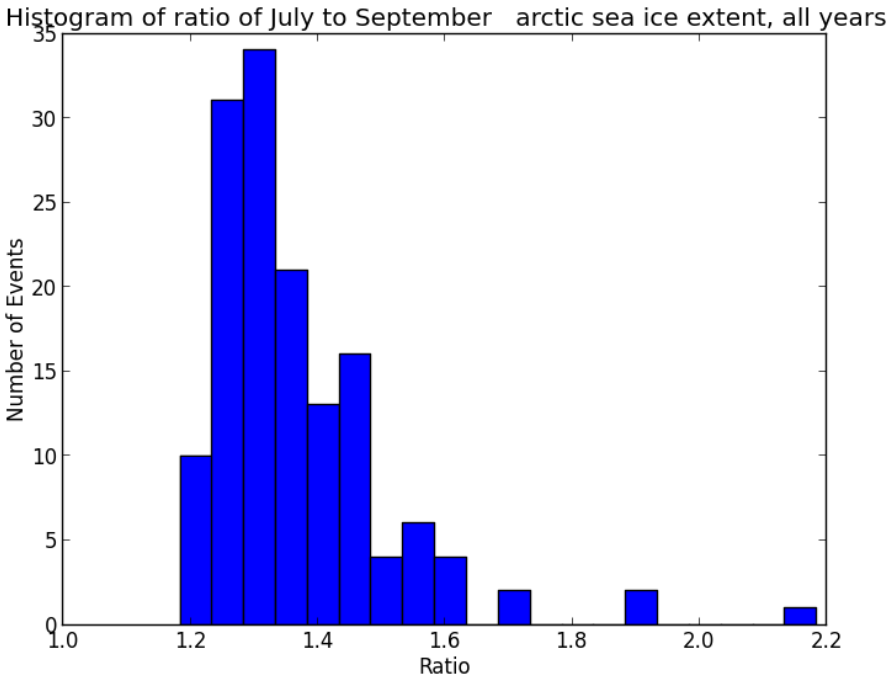


Figure 7: Histogram of summer melting events

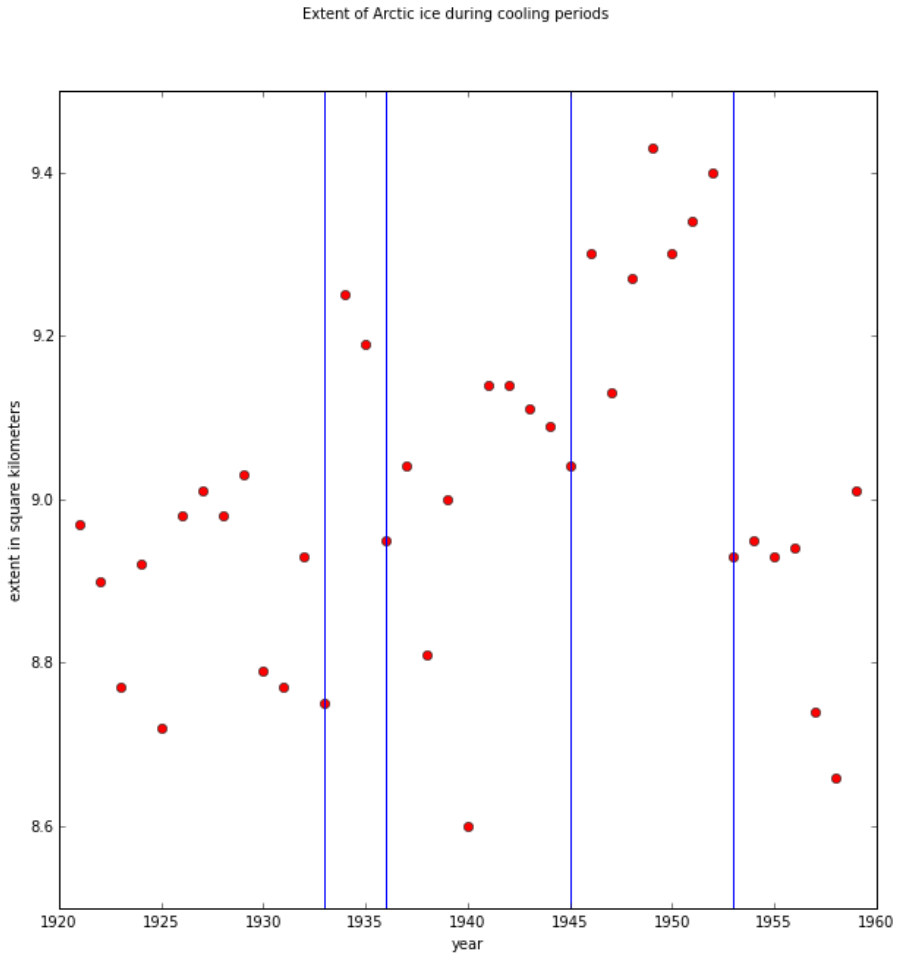


Figure 8: Cooling period windows

Figures 9, 10, and 11 have some story but I don't know what it is yet.

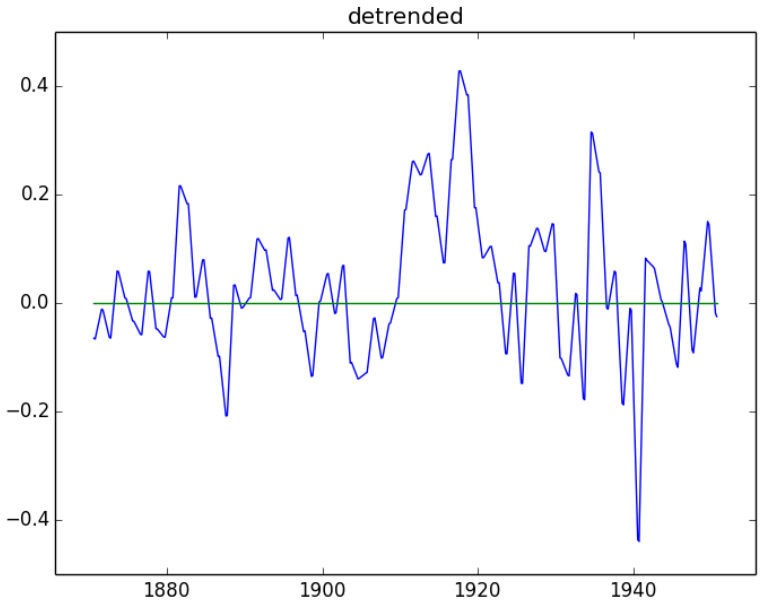


Figure 9: Detrended Data

6 Extrapolation: visiting Santa via canoe

6.1 First attempt

Attempts to fit the entire data set with an exponential function did not succeed in finding the proper parameters. We can consider fitting a polynomial function to the entire data set. This is accomplished in Fig. 12 with a polynomial of degree 6. According to this fit, Arctic ice will melt completely by the year 2029.

6.2 Fits to satellite data

We can look at the data collected for the ice extent in September starting in the year 1979 and perform polynomial fits to the data to estimate the time at which Arctic sea ice will melt. The linear (first degree polynomial) fit is shown in Fig. 13, and the third degree polynomial fit is shown in Fig. 14. Their predictions are stated in the figure captions.

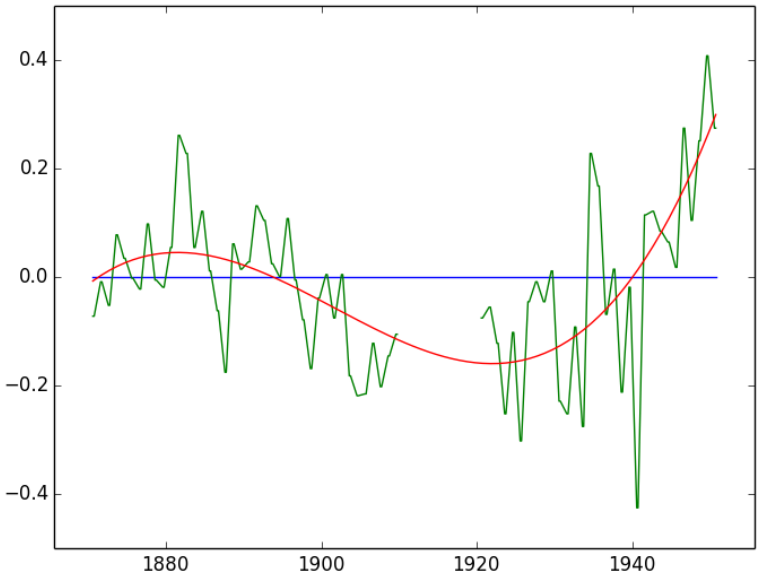


Figure 10: A baseline fit

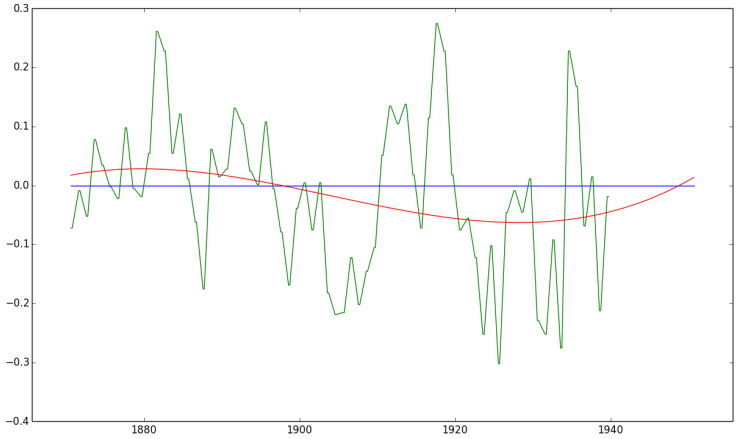


Figure 11: 1940 removed

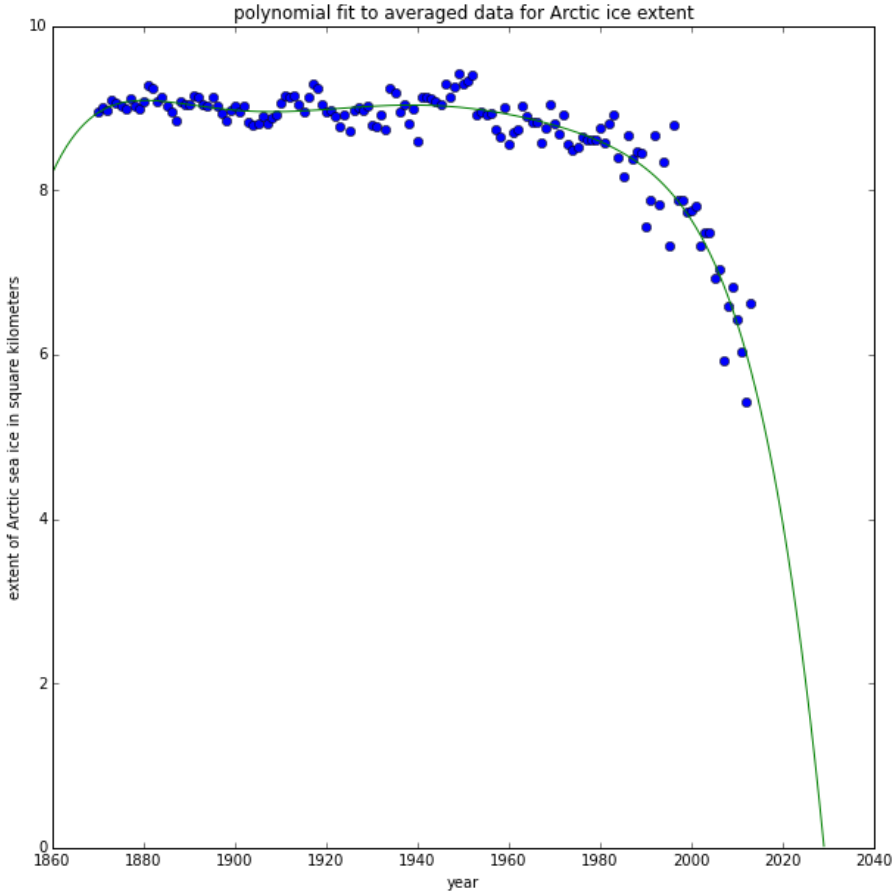


Figure 12: Visit Santa by canoe in 2029

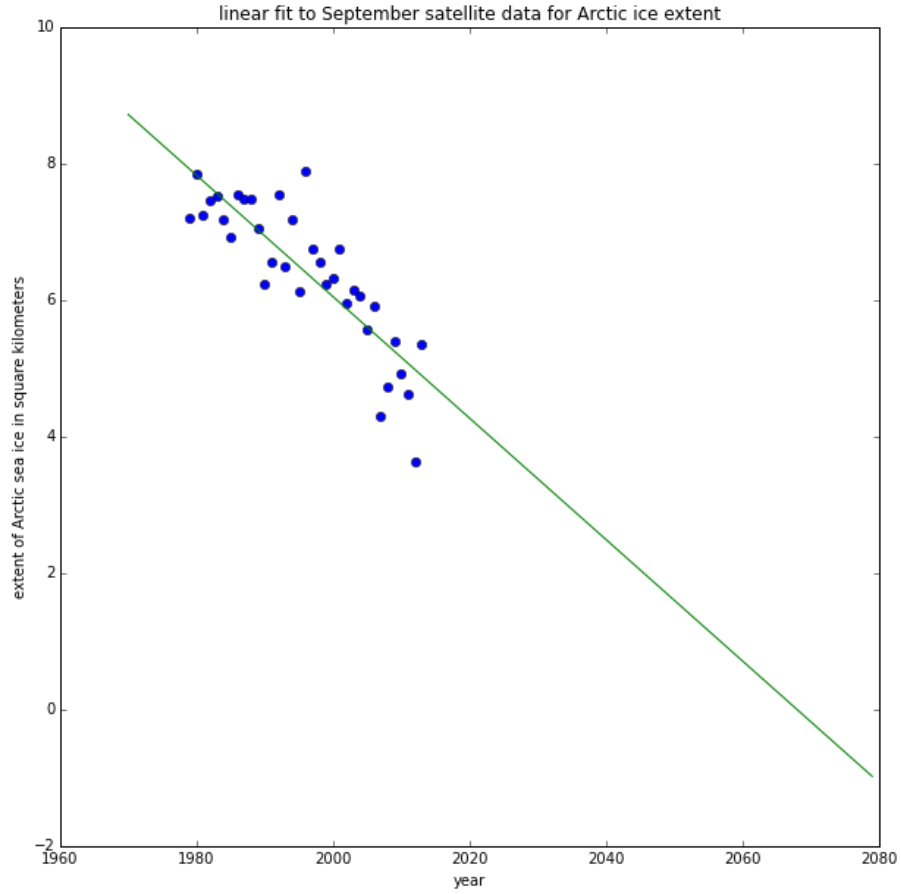


Figure 13: according to linear fit to satellite data Arctic ice will melt in the year 2067

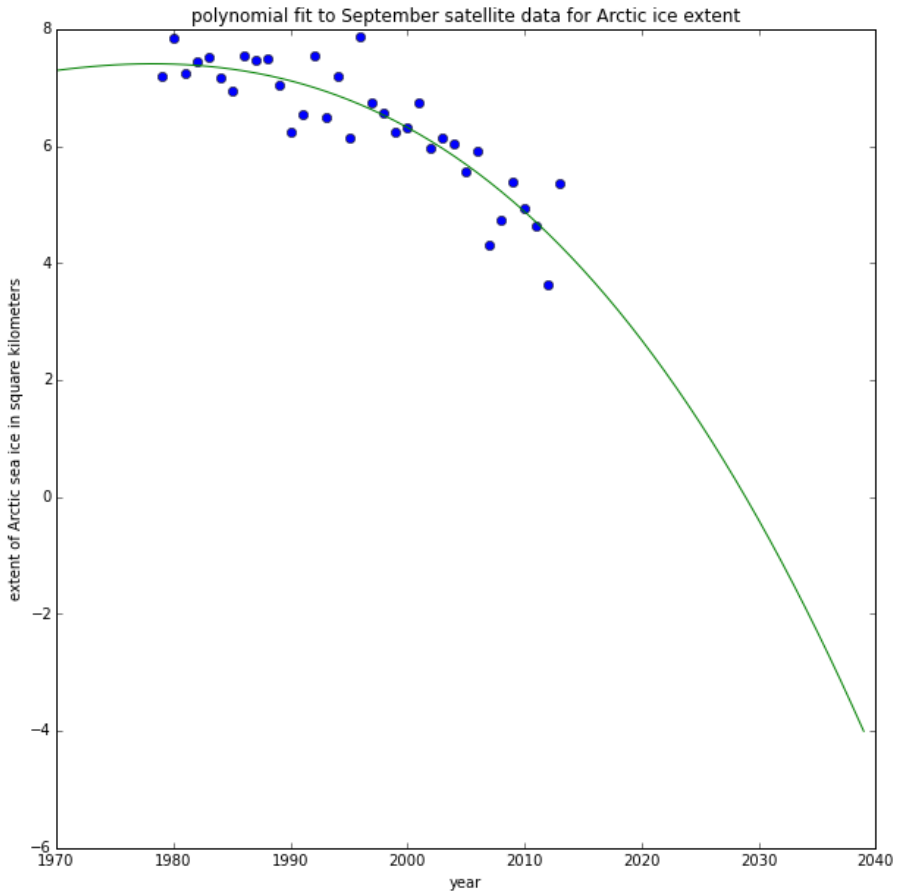
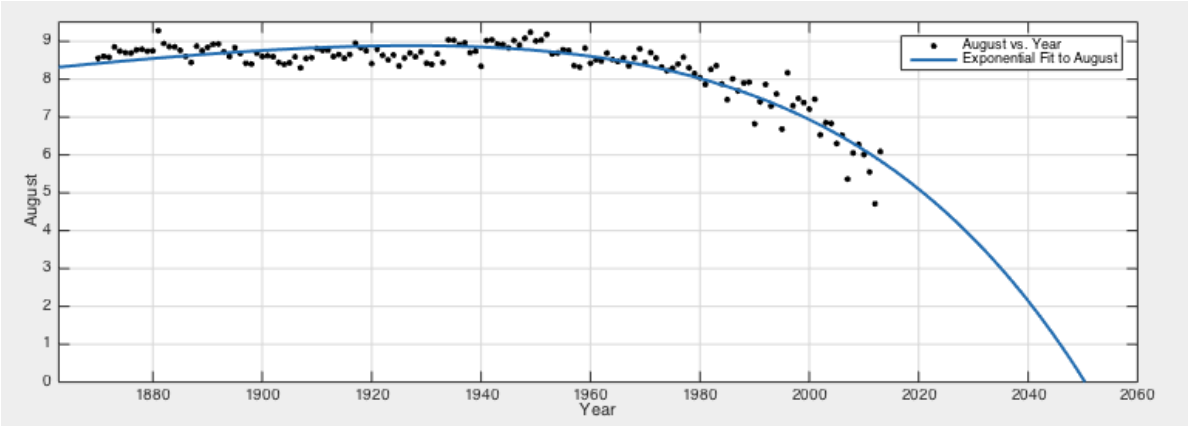


Figure 14: according to polynomial fit to satellite data Arctic ice will melt in the year 2028

6.3 Another approach, using Matlab

Exponential Fit of August Data:

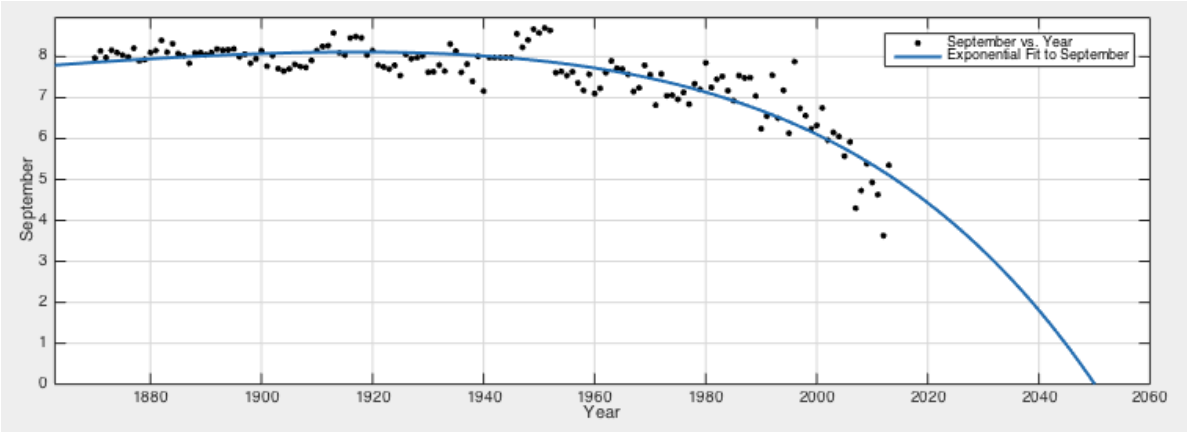


Suggested Cruiser Year: 2051

General model Exp2:
f(x) = a*exp(b*x) + c*exp(d*x)
Coefficients (with 95% confidence bounds):
a = 0.08908 (-0.2407, 0.4189)
b = 0.002459 (0.0004295, 0.004489)
c = -1.238e-16 (-2.861e-15, 2.613e-15)
d = 0.01914 (0.008502, 0.02979)

Goodness of fit:
SSE: 13.77
R-square: 0.8589
Adjusted R-square: 0.8559
RMSE: 0.3136

Exponential Fit of September Data:

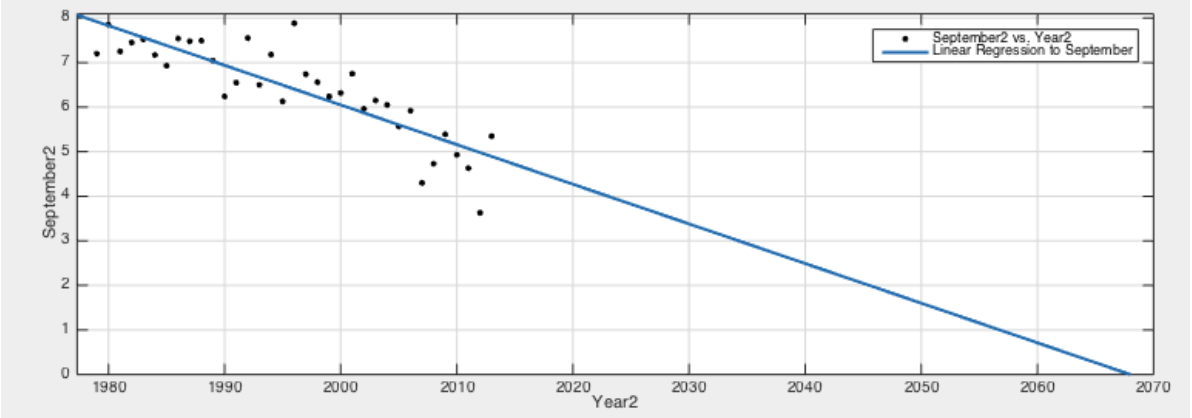


Suggested Cruiser Year: 2050

General model Exp2:
f(x) = a*exp(b*x) + c*exp(d*x)
Coefficients (with 95% confidence bounds):
a = 0.185 (-0.8124, 1.182)
b = 0.002032 (-0.0009242, 0.004989)
c = -2.511e-16 (-8.349e-15, 7.847e-15)
d = 0.01873 (0.003203, 0.03426)

Goodness of fit:
SSE: 24.22
R-square: 0.7878
Adjusted R-square: 0.7833
RMSE: 0.4159

Linear Regression of Satellite Data for September:

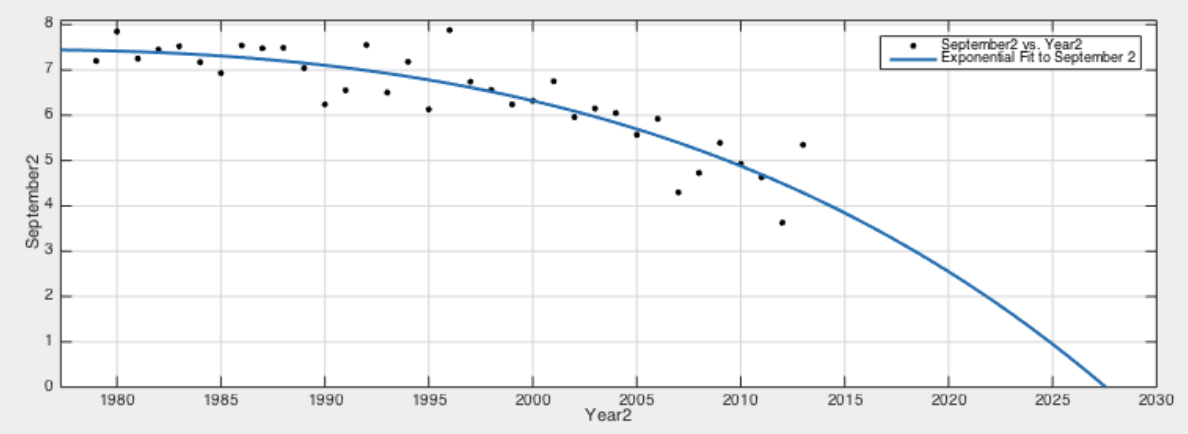


Suggested Cruiser Year: 2069

Linear model Poly1:
f(x) = p1*x + p2
Coefficients (with 95% confidence bounds):
p1 = -0.08896 (-0.1083, -0.06963)
p2 = 184 (145.4, 222.6)

Goodness of fit:
SSE: 10.64
R-square: 0.7265
Adjusted R-square: 0.7182
RMSE: 0.5678

Exponential Fit of Satellite Data for September:



Suggested Cruiser Year: 2027

General model Exp2:
f(x) = a*exp(b*x) + c*exp(d*x)
Coefficients (with 95% confidence bounds):
a = -4.554e-13 (-6.947e-05, 6.947e-05)
b = 0.01928 (-684.9, 684.9)
c = 4.639e-13 (-6.947e-05, 6.947e-05)
d = 0.01927 (-684.7, 684.7)

Goodness of fit:
SSE: 8.134
R-square: 0.7909
Adjusted R-square: 0.7706
RMSE: 0.5122

7 Code

It can't really be argued that we truly collaborated here, given how all of us have our own code and own outputs. But we tried.

7.1 Andrea's Code

```
import re
import math
import numpy as np
import matplotlib
matplotlib.use('PS')
import matplotlib.pyplot as plt

"""Data Import"""
#arctic ice sea extent in km^2
#split data file into rows (year,July,August,September)
ice = []
with open('../data/icedata.txt','r') as f:
    for line in f:
        s = re.split("\t", line)
        ice.append([int(s[0]),float(s[1]),float(s[2]),float(s[3])])
ice = np.array(ice)

# sanity check for averaging routine
# for i in range(3):
#   print np.mean(ice[0:5,i+1])
# print np.mean(ice[0:5,1:]) #avg of ice in all 3 months over 5 years

"""Functions - averages, slopes, integration"""

def get_average(data,step):
    """Finds average ice extent of 3 months in dataset
    given step size"""
    average = []
    i = 0
    while i < len(ice):
        year = ice[i,0]
        area = np.mean(ice[i:i+step,1:])
        average.append([year,area])
        i += step
    average = np.array(average)
    return average

def get_slope(averaged_data):
    """Finds slope of averaged data."""
    slope =  []
    for i in range(len(averaged_data)-1):
```

```
        slope.append([averaged_data[i,0], \
          (averaged_data[i+1,1]-averaged_data[i,1])/5])
        slope.append([averaged_data[i,0]+1, \
          (averaged_data[i+1,1]-averaged_data[i,1])/5])
        slope.append([averaged_data[i,0]+2, \
          (averaged_data[i+1,1]-averaged_data[i,1])/5])
        slope.append([averaged_data[i,0]+3, \
          (averaged_data[i+1,1]-averaged_data[i,1])/5])
        slope.append([averaged_data[i,0]+4, \
          (averaged_data[i+1,1]-averaged_data[i,1])/5])
    slope = np.array(slope)
    return slope

def integrate(start,stop):
    """Integrates under flat slope curve,
    starting and stopping on specified years"""
    #value of slope
    i = start - 1870
    integral = 0
    while i < stop - 1870:
        integral += 5*slope[i,1]
        i += 5
    return integral

"""Functions: Smoothing algorithms borrowed from Gene"""
# The following algorithm smoothes data using n-point sliding box-car
# smoothing. http://users.aims.ac.za/~mike/python/env_modelling.html
# this is a better algorithm because it takes care of the
# "residuals" by reflecting the data around the endpoints

def smootherBox(x, n):
     x_smooth = []
     L = len(x)
     store = np.zeros((n,1), float)  # array of zeroes for each group
     for u in range(0, L-n):       # used to produce smoothed value
          for v in range(0, n):  #  add  values and divide by number
               store[v] = x[u+v]
          av = float(sum(store)) / n # repeat for each value at center
          x_smooth.append(av)
     for u in range(L-n,L):  # takes care of the "residual" at the end
          for v in range(0, L-u-1):  # average and find smoothed value
               store[v] = x[u+v]
          av = float(sum(store)) / n
          x_smooth.append(av)
     return x_smooth

def smootherGauss(x, n):
     coeff = [0.067, 0.242, 0.383, 0.242, 0.067]
```

```
# these are coefficients corresponding to Gaussian kernel of width 5
# from  homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm
     x_smooth = []
     L = len(x)
     store = np.zeros((n,1), float)  # array of zeroes for each group
     for u in range(0, L-n):       # used to produce smoothed value
          for v in range(0, n):    #  mult gauss coeffs by values
               store[v] = x[u+v]
          av = 0
          for i in range(0, len(coeff)):
               av += float(coeff[i]) * float(store[i])
          x_smooth.append(av)
     for u in range(L-n,L):  # takes care of "residuals" at the end
          for v in range(0, L-u-1):  # average and find smoothed value
               store[v] = x[u+v]
          av = 0
          for i in range(0, len(coeff)):
               av += float(coeff[i]*store[i])
          x_smooth.append(av)
     return x_smooth

## in exponential smoothing each value is defined as
## s[0] = x[0] and s[t] = a x[t-1] + (1-a) s[t-1]
## where 0<a<1 is smoothing parameter.  Larger values of a
## give more weight to recent data but smoothing is worse
def smootherExp(x, a):
    x_smooth = []
    L = len(x)
    x_smooth.append(x[0])
    for u in range(1, L):
        store = a * float(x[u-1]) + (1-a) * float(x_smooth[u-1])
        x_smooth.append(store)
    return x_smooth

"""Calculations and data preparation for plots"""
#one year average
annual_average = get_average(ice,1)

#five year average - excludes 2010-2013 data
ice_average = get_average(ice,5)

#flat line slope, five year average - excludes 2010-2013 data
slope = get_slope(ice_average)

# change of sea ice extent
early = integrate(1870,1950) # 0.166
late = integrate(1950,2009) # 03.055
# print early
```

```
# print late

#smoothed ice extent data
box_smooth_list = smootherBox(annual_average[:,1],5)
box_smooth = np.array([[annual_average[i,0],box_smooth_list[i]] \
  for i in range(len(box_smooth_list))])

gauss_smooth_list = smootherGauss(annual_average[:,1],5)
gauss_smooth = np.array([[annual_average[i,0],gauss_smooth_list[i]] \
  for i in range(len(gauss_smooth_list))])

exp_smooth_all = []
for i in np.arange(0,1,0.1):
    exp_smooth_list = smootherExp(annual_average[:,1],i)
    exp_smooth = np.array([[annual_average[i,0],exp_smooth_list[i]] \
      for i in range(len(exp_smooth_list))])
    exp_smooth_all.append(exp_smooth)

"""Plots"""
plt.figure(0)
plt.plot(ice_average[:,0],ice_average[:,1])
plt.xlabel('Beginning Year of 5 year period')
plt.ylabel('Square kilometers')
plt.title('Area of June-September Arctic sea ice extent, \
  5 year averages')
plt.savefig('../output/avg_area_five_year')

plt.figure(1)
n, bins, patches = plt.hist(ice[:,1]/ice[:,3],20)
plt.xlabel('Ratio')
plt.ylabel('Number of Events')
plt.title('Histogram of ratio of July to September \
  arctic sea ice extent, all years')
plt.savefig('../output/annual_ratio_hist')

plt.figure(2)
plt.plot(ice[:,0],ice[:,1]/ice[:,3])
plt.xlabel('Year')
plt.ylabel('Ratio')
plt.title('Ratio of July to September arctic sea ice extent, \
  all years')
plt.savefig('../output/annual_ratio')

plt.figure(3)
plt.plot(slope[:,0],slope[:,1])
plt.xlabel('Year')
plt.ylabel('Rate of change, square kilometers/year')
plt.title('Rate of change of area of June-September \
```

```
  Arctic sea ice extent, 5 year averages')
plt.savefig('../output/slope')

plt.figure(4)
plt.plot(annual_average[:,0],annual_average[:,1])
plt.xlabel('Year')
plt.ylabel('Square kilometers')
plt.title('Area of June-September Arctic sea ice extent, all years')
plt.savefig('../output/avg_area_annual')

plt.figure(5)
plt.plot(annual_average[:,0],annual_average[:,1],\
  label=r'raw data', color='black')
plt.plot(box_smooth[:,0],box_smooth[:,1],\
  label=r'boxcar-smoothed data', color='blue')
plt.plot(gauss_smooth[:,0],gauss_smooth[:,1],\
  label=r'Gaussian-smoothed data', color='green')
plt.plot(exp_smooth_all[7][:,0], exp_smooth_all[7][:,1],\
  label=r'exponential-smoothed data, smoothing parameter 0.7',\
  color='red')
plt.xlabel('Year')
plt.ylabel('Square kilometers')
plt.title('Smoothed Area of June-September \
  Arctic sea ice extent, all years')
plt.legend(loc='lower left')
leg = plt.gca().get_legend()
ltext  = leg.get_texts()
plt.setp(ltext, fontsize='small')
plt.savefig('../output/smooth_avg_area_annual')

# generate exponentially smoothed plot commands
# for i in range(10):
#     j = np.arange(0,1,0.1)[i]
#     print 'plt.plot(exp_smooth_all['+str(i)+'][:,0], \
#       exp_smooth_all['+str(i)+'][:,1],label=r\'exponential-smoothed\
#       data, smoothing parameter '+str(j)+'\')'

plt.figure(6)
plt.plot(exp_smooth_all[1][:,0], exp_smooth_all[1][:,1],\
  label=r'exponential-smoothed data, smoothing parameter 0.1')
plt.plot(exp_smooth_all[2][:,0], exp_smooth_all[2][:,1],\
  label=r'exponential-smoothed data, smoothing parameter 0.2')
plt.plot(exp_smooth_all[3][:,0], exp_smooth_all[3][:,1],\
  label=r'exponential-smoothed data, smoothing parameter 0.3')
plt.xlabel('Year')
plt.ylabel('Square kilometers')
plt.title('Various Exponentially Smoothed Ice Extent Averages')
plt.legend(loc='lower left')
```

```
plt.savefig('../output/expsmooth123')

plt.figure(7)
plt.plot(exp_smooth_all[4][:,0], exp_smooth_all[4][:,1],\
  label=r'exponential-smoothed data, smoothing parameter 0.4')
plt.plot(exp_smooth_all[5][:,0], exp_smooth_all[5][:,1],\
  label=r'exponential-smoothed data, smoothing parameter 0.5')
plt.plot(exp_smooth_all[6][:,0], exp_smooth_all[6][:,1],\
  label=r'exponential-smoothed data, smoothing parameter 0.6')
plt.xlabel('Year')
plt.ylabel('Square kilometers')
plt.title('Various Exponentially Smoothed Ice Extent Averages')
plt.legend(loc='lower left')
plt.savefig('../output/expsmooth456')

plt.figure(8)
plt.plot(exp_smooth_all[7][:,0], exp_smooth_all[7][:,1],\
  label=r'exponential-smoothed data, smoothing parameter 0.7')
plt.plot(exp_smooth_all[8][:,0], exp_smooth_all[8][:,1],\
  label=r'exponential-smoothed data, smoothing parameter 0.8')
plt.plot(exp_smooth_all[9][:,0], exp_smooth_all[9][:,1],\
  label=r'exponential-smoothed data, smoothing parameter 0.9')
plt.xlabel('Year')
plt.ylabel('Square kilometers')
plt.title('Various Exponentially Smoothed Ice Extent Averages')
plt.legend(loc='lower left')
plt.savefig('../output/expsmooth789')
```

7.2 Gene's Code

```
# -*- coding: utf-8 -*-
# <nbformat>2</nbformat>

# <codecell>

infile = open("iceData.txt","r") ## read the data file
content = infile.read()
infile.close()
lines = content.split('\n') ## split up lines
#print lines

# <codecell>

data = []
for each in lines:  ## split each line into separate strings
    if len(each) < 1:
        continue          ## get rid of the empty list at the end
    temp = each.split()
```

```
    data.append(temp)     ## data contains lists of data for each year
print data
## data contains year, July, August, and September ice extent in square kilometers

# <codecell>

dataNum = []  ## this will hold data converted to floating point types
for each in data:
    templine = []
    for i in range(0, len(each)):  ## convert element in each line to float
        temp = float(each[i])
        templine.append(temp)
    dataNum.append(templine)  ## write lists of floats to array dataNum
print dataNum

# <codecell>

avgExt = []   ## this array will hold year and its corresponding summer average exte
for each in dataNum:
    avg = round((each[1] + each[2] + each[3])/3., 2)  ## round to 2 decimal places
    temp = [int(each[0]), avg]
    avgExt.append(temp)
print avgExt
print len(avgExt)

# <codecell>

import matplotlib.pyplot as plt
import numpy as np

julyExt = []  ## make plots of the data for the three months and average ice
augExt = []   ## extent over the years to get an idea of what the data looks like
septExt = []

for each in dataNum:
    temp1 = [int(each[0]), each[1]]
    julyExt.append(temp1)
    temp2 = [int(each[0]), each[2]]
    augExt.append(temp2)
    temp3 = [int(each[0]), each[3]]
    septExt.append(temp3)

def plotData(monthData, month, plotVal):
    x_val = [x[0] for x in monthData]
    y_val = [x[1] for x in monthData]
    plt.plot(x_val,y_val, plotVal)
    plt.suptitle('Extent of Arctic ice in ' + month)
    plt.xlabel('year')
```

```
    plt.ylabel('extent in square kilometers')
    return plt.show()

plotData(julyExt, 'July', 'bo')
plt.savefig("Ice-extent-July.png")
plotData(augExt, 'August', 'bo')
plt.savefig("Ice-extent-August.png")
plotData(septExt, 'September', 'bo')
plt.savefig("Ice-extent-September.png")
plotData(avgExt, 'average over three Summer months', 'ro')
plt.savefig("Ice-extent-ThreeMonthsAverage.png")

# <codecell>

z = [x[0] for x in avgExt]
y1 = [x[1] for x in julyExt]
y2 = [x[1] for x in augExt]
y3 = [x[1] for x in septExt]
y4 = [x[1] for x in avgExt]

fig = plt.figure(figsize=(10, 10))
plt.xlabel('year')
plt.ylabel('extent of ice in square kilometers')
plt.title(r'Extent of Arctic ice')
plt.plot(z, y1, label=r'July', color='cyan')
plt.plot(z, y2, label=r'August', color='blue')
plt.plot(z, y3, label=r'September', color='green')
plt.plot(z, y4, label=r'Three months average', color='red')
plt.legend(loc='lower left')
plt.show()
plt.savefig("Ice-extent.png")

# <codecell>

#from numpy import gradient
dx = 5
x = [] ## holds values of years
y = [] ## holds values of corresponding extents of ice
dy = [] ## five year differences
dydx = [] ## slopes over five years
xmid = [] ## midpoint for each five year interval
for each in avgExt:
    temp = each[0]
    x.append(temp)
    temp1 = each[1]
    y.append(temp1)
#print x
#print y
```

```
for i in range(0, len(y)-dx):
    temp = y[i+dx]-y[i]
    dy.append(temp)
    temp1 = temp/dx
    dydx.append(temp1)
    temp2 = float(x[i+dx]+x[i])/2.
    xmid.append(temp2)
#print dydx
#print xmid
#print len(dydx)
#print len(xmid)

# <codecell>

z = [x for x in xmid]
y = [x for x in dydx]

fig = plt.figure(figsize=(10, 10))
plt.xlabel('year')
plt.ylabel('rate of change in square kilometers per year')
plt.title(r'rate of change of extent of Arctic ice')
plt.plot(z, y, color='blue')
plt.show()
plt.savefig("rate-of-change-of-extent.png")

# <codecell>

rate = []  ## this is the rate of change of Arctic sea ice extent
for i in range(0, len(dydx)):
    temp = [xmid[i], dydx[i]]
    rate.append(temp)

upTo1950 = []
after1950 = []
for each in rate:
    temp = each[1]
    if each[0] < 1950:
        upTo1950.append(temp)
    else:
        after1950.append(temp)
#print upTo1950
#print after1950

# <codecell>

## compute the area under above curve to find the change in extent of ice
from scipy.integrate import simps, trapz
```

```
## dx indicates the spacing of the data along the x axis.

## Compute the area using the composite trapezoidal rule.
areaUpTo1950 = trapz(upTo1950, dx=1)
#print "area up to 1950 using trapezoidal rule =", areaUpTo1950
areaAfter1950 = trapz(after1950, dx=1)
#print "area after 1950 using trapezoidal rule =", areaAfter1950
## Compute the area using the composite Simpon's rule.
areaUpTo1950 = simps(upTo1950, dx=1)
#print "area up to 1950 using Simpson's rule =", areaUpTo1950
areaAfter1950 = simps(after1950, dx=1)
ratio = areaAfter1950/areaUpTo1950
print "change in extent of sea ice up to 1950 =", round(areaUpTo1950, 2), "square kil
print "change in extent of sea ice after 1950 =", areaAfter1950, "square kilometers"
print "ratio after 1950 to that before =", round(ratio, 2)

# <codecell>

extent = []  ## this will hold all the values of average extent data
for each in avgExt:
    temp = each[1]
    extent.append(temp)
print extent

    ##  this does not do a good job at the end because of "residuals"
#def runningMean(x, N):  ## this will compute the running mean over N values
#    y = np.zeros((len(x),))
#    for ctr in range(len(x)):
#         y[ctr] = np.sum(x[ctr:(ctr+N)])
#    return y/N
#avgExtBoxSmooth = runningMean(extent, 3)

# <codecell>

#import scipy
#from scipy.signal import boxcar

## try it using a built-in Python function
#boxSmooth = boxcar(extent)
#print boxSmooth

## The following algorithm smoothes data using n-point sliding box-car
## smoothing. Stolen from http://users.aims.ac.za/~mike/python/env_modelling.html
## this is a better algorithm because it takes care of the "residuals" by
## reflecting the data around the endpoints

def smoother(x, n):
     x_smooth = []
```

```
    L = len(x)
    store = zeros((n,1), float)  ## array of zeroes for each group of
    for u in range(0, L-n):      ## used to produce smoothed value
        for v in range(0, n):  ##  add the values and divide by their number
            store[v] = x[u+v]
        av = float(sum(store)) / n  ## repeat for each value at center
        x_smooth.append(av)
    for u in range(L-n,L):  ## this takes care of the "residual" at the end
        for v in range(0, L-u-1):  ## average the values and find the smoothed val
            store[v] = x[u+v]
        av = float(sum(store)) / n
        x_smooth.append(av)
    return x_smooth

avgExtBoxSmooth = smoother(extent, 5)  ## data smoothed over interval of 5 years
print avgExtBoxSmooth
print len(avgExtBoxSmooth)

# <codecell>

year = []
avgExtBox = []
for each in avgExt:
    temp = each[0]
    year.append(temp)
for i in range(0, len(year)):
    temp = [year[i], avgExtBoxSmooth[i]]
    avgExtBox.append(temp)

# <codecell>

z = [x[0] for x in avgExtBox]
y1 = [x[1] for x in avgExt]
y2 = [x[1] for x in avgExtBox]

fig = plt.figure(figsize=(10, 10))
plt.xlabel('year')
plt.ylabel('extent of ice in square kilometers')
plt.title(r'comparing extent of Arctic ice data with data smoothed with Boxcar filter
plt.plot(z, y1, label=r'raw data', color='red')
plt.plot(z, y2, label=r'smoothed data', color='blue')
plt.legend(loc='lower left')
plt.show()
plt.savefig("Ice-extent-Boxcar-smoothed.png")

# <codecell>

def smootherGauss(x, n):
```

```
     coeff = [0.067, 0.242, 0.383, 0.242, 0.067]
## these are coefficients corresponding to Gaussian kernel of width 5
## taken from the web page homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm
     x_smooth = []
     L = len(x)
     store = zeros((n,1), float)  ## array of zeroes for each group of
     for u in range(0, L-n):      ## used to produce smoothed value
          for v in range(0, n):   ##  add the values multiplied by gaussian coeffici
               store[v] = x[u+v]
          av = 0
          for i in range(0, len(coeff)):
               av += float(coeff[i]) * float(store[i])
          x_smooth.append(av)
     for u in range(L-n,L):  ## this takes care of the "residuals" at the end
          for v in range(0, L-u-1):  ## average the values and find the smoothed val
               store[v] = x[u+v]
          av = 0
          for i in range(0, len(coeff)):
               av += float(coeff[i]*store[i])
          x_smooth.append(av)
     return x_smooth

avgExtGaussSmooth = smootherGauss(extent, 5)  ## data smoothed over interval of 5 ye
print avgExtGaussSmooth
print len(avgExtGaussSmooth)

# <codecell>

avgExtGauss = []
for i in range(0, len(year)):
    temp = [year[i], avgExtGaussSmooth[i]]
    avgExtGauss.append(temp)

z = [x[0] for x in avgExtGauss]
y1 = [x[1] for x in avgExt]
y2 = [x[1] for x in avgExtGauss]
fig = plt.figure(figsize=(10, 10))
plt.xlabel('year')
plt.ylabel('extent of ice in square kilometers')
plt.title(r'comparing extent of Arctic ice data with data smoothed with Gaussian filt
plt.plot(z, y1, label=r'raw data', color='red')
plt.plot(z, y2, label=r'smoothed data', color='blue')
plt.legend(loc='lower left')
plt.show()
plt.savefig("Ice-extent-Gaussian-smoothed.png")

# <codecell>
```

```
## in exponential smoothing each value is defined as
## s[0] = x[0] and s[t] = a x[t-1] + (1-a) s[t-1]
## where 0<a<1 is smoothing parameter.  Larger values of a
## give more weight to recent data but smoothing is worse
def smootherExp(x, a):
    x_smooth = []
    L = len(x)
    x_smooth.append(x[0])
    for u in range(1, L):
        store = a * float(x[u-1]) + (1-a) * float(x_smooth[u-1])
        x_smooth.append(store)
    return x_smooth
avgExtExpSmooth = smootherExp(extent, .7)  ## data smoothed over interval of 5 years
print avgExtExpSmooth
print len(avgExtExpSmooth)

# <codecell>

avgExtExp = []
for i in range(0, len(year)):
    temp = [year[i], avgExtExpSmooth[i]]
    avgExtExp.append(temp)

z = [x[0] for x in avgExtExp]
y1 = [x[1] for x in avgExt]
y2 = [x[1] for x in avgExtExp]
fig = plt.figure(figsize=(10, 10))
plt.xlabel('year')
plt.ylabel('extent of ice in square kilometers')
plt.title(r'comparing extent of Arctic ice data with data smoothed with exponential f
plt.plot(z, y1, label=r'raw data', color='red')
plt.plot(z, y2, label=r'smoothed data', color='blue')
plt.legend(loc='lower left')
plt.show()
plt.savefig("Ice-extent-exponential-smoothed.png")

# <codecell>

z = [x for x in year]
y = [x for x in extent]
y1 = [x for x in avgExtBoxSmooth]
y2 = [x for x in avgExtGaussSmooth]
y3 = [x for x in avgExtExpSmooth]
fig = plt.figure(figsize=(10, 10))
plt.xlabel('year')
plt.ylabel('extent of ice in square kilometers')
plt.title(r'comparing extent of Arctic ice data with data smoothed with different fil
plt.plot(z, y, label=r'raw data', color='red')
```

```
plt.plot(z, y1, label=r'boxcart smoothed ', color='blue')
plt.plot(z, y2, label=r'gaussian smoothed', color='magenta')
plt.plot(z, y3, label=r'exponential smoothed ', color='cyan')
plt.legend(loc='lower left')
plt.show()
plt.savefig("Ice-extent-smoothed-comparison.png")

# <codecell>

import pylab as P

## find the ratio between yearly July and September ice extents
july = [x[1] for x in julyExt]
aug = [x[1] for x in augExt]
sept = [x[1] for x in septExt]
ratioJulyToSept = []
for i in range(len(july)):
    temp = float(july[i]) / float(sept[i])
    ratioJulyToSept.append(temp)
#z = [x for x in year]
#y = [x for x in ratioJulyToSept]
#plt.xlabel('year')
#plt.ylabel('ratio of ice extent')
#plt.title(r'ratio of extent of Arctic ice in July to that in September ')
#plt.plot(z, y, color='red')
#plt.bar(z, y)
#plt.show()
#plt.savefig("Ice-extent-comparison-ratio-July-September.png")

plt.xlabel('ratio of ice extent')
plt.ylabel('number of occurences')
plt.title(r'histogram ratio of extent of Arctic ice in July to that in September ')
plt.hist(ratioJulyToSept)
plt.show()
plt.savefig("Ice-extent-histogram-ratio-July-September.png")

# <codecell>

from numpy import *
from matplotlib.pyplot import *

coefficients = polyfit(year, extent, 6)
polynomial = poly1d(coefficients)
xs = arange(1860, 2030, 1)
ys = polynomial(xs)
zeroes = roots(coefficients)
#print zeroes
for each in zeroes:    ## find the year at which ice extent goes to zero
```

```
    if int(each.real) > 2000:
        zero = int(each.real)
print "according to our fit Arctic ice will melt in the year", zero
fig = plt.figure(figsize=(10, 10))
plot(year, extent, 'o')
plot(xs, ys)
ylabel('extent of Arctic sea ice in square kilometers')
xlabel('year')
title(r'polynomial fit to averaged data for Arctic ice extent ')
show()
savefig("Ice-extent-polynomial-fit.png")

# <codecell>

plotData(avgExt, 'average over three Summer months', 'ro')

# <codecell>

## fit an exponential curve to noisy data to check if exponential fit works
from scipy.optimize import curve_fit

def fitFunc(t, a, b, c):
    return a*np.exp(-b*t) + c  ## x is the x-axis
x = np.linspace(0,4,50)
y = fitFunc(x, 2.5, 1.3, 0.5)
yn = y + 0.2*np.random.normal(size=len(x)) ## noisy function
fitParams, fitCovariances = curve_fit(fitFunc, x, yn) ## yn is the function to fit to
## The scipy.optimize module contains a least squares curve fit routine
## that requires as input a user-defined fitting function (in our case fitFunc ),
## the x-axis data (in our case, t) and the y-axis data (in our case, noisy).
## The curve_fit routine returns an array of fit parameters, and a matrix of
## covariance data (the square root of the diagonal values are the 1-sigma
## uncertainties on the fit parametersprovided you have a reasonable fit in
## the first place.):
print fitParams

# <codecell>

## found it impossible to find the right fitting parameters for exponential fit
x = np.array(year)
y = np.array(extent)
#print x
#print y
def fitFunc(t, a, b, c):
    return a*np.exp(b*t) + c
## try this set of parameters to find an initial guess
popt = [-.00000001, .01, 10]
plot(x,fitFunc(x,*popt))
```

```
show()

## use initial guess to get closer
fitParams, fitCovariances = curve_fit(fitFunc, x, y, [-.00000001, .01, 10])
print fitParams
## still doesn't seem to be able to optimize

# <codecell>

## now let's look at the window 1900 - 1960 to find the alleged cooling events:
cooling = []
for each in avgExt:
    if each[0] > 1920 and each[0] < 1960:
        temp = [each[0],each[1]]
        cooling.append(temp)
x_val = [x[0] for x in cooling]
y_val = [x[1] for x in cooling]
fig = plt.figure(figsize=(10, 10))
plot(x_val,y_val, 'ro')
suptitle('Extent of Arctic ice during cooling periods')
xlabel('year')
ylabel('extent in square kilometers')
## put vertical lines at the beginning and end of cooling window
axvline(x=1933)
axvline(x=1936)
axvline(x=1945)
axvline(x=1953)
show()
savefig("Ice-extent-cooling-windows.png")

# <codecell>

## it looks like first cooling period begins in 1933 and ends in 1936
## and the second cooling period begins in 1945 and ends in 1953
## since we already have the derivative function dydx, we can simply
## find the area under this curve for the time periods we are interested in:
firstWindow = []
secondWindow = []
for each in rate: ## rate holds the midpoints and corresponding derivatives
    temp = each[1]
    if each[0] > 1933 and each[0] < 1936:
        firstWindow.append(temp)
    elif each[0] > 1945 and each[0] < 1950:
        secondWindow.append(temp)
print firstWindow
print secondWindow
## in the second window going up to 1950 only to account for increase in ice coverag
```

```
# <codecell>

## Compute the area using the composite Simpon's rule.
areaFirstWindow = simps(firstWindow, dx=1)
areaSecondWindow = simps(secondWindow, dx=1)
print areaFirstWindow
print areaSecondWindow

firstRatio = areaFirstWindow/areaUpTo1950
secondRatio = areaSecondWindow/areaUpTo1950
print "change in extent of sea ice in first cooling window =", areaFirstWindow, "squa
print "change in extent of sea ice in the second cooling window =", areaSecondWindow,
print "ratio in first cooling window to total change prior to 1950 =", firstRatio
print "ratio in second cooling window to total change prior to 1950 =", secondRatio

# <codecell>

## now we use September data after 1979 and make linear fit and a polynomial
## fit to the data

septExtSat = []
for each in septExt:
    if each[0] >= 1979:
        septExtSat.append(each)
#print septExtSat
yearSat = []
extSat = []
for each in septExtSat:
    yearSat.append(each[0])
    extSat.append(each[1])
## linear fit or polynomial of degree 1
coeffLin = polyfit(yearSat, extSat, 1)
polynomial = poly1d(coeffLin)
xl = arange(1970, 2080, 1)
yl = polynomial(xl)
zeroes = roots(coeffLin)
#print zeroes
for each in zeroes:
    if int(each.real) > 2020:
        zero = int(each.real)
print "according to linear fit to satellite data Arctic ice will melt in the year", z
fig = plt.figure(figsize=(10, 10))
plot(yearSat, extSat, 'o')
plot(xl, yl)
ylabel('extent of Arctic sea ice in square kilometers')
xlabel('year')
title(r'linear fit to September satellite data for Arctic ice extent ')
show()
```

```
savefig("Ice-extent-satellite-linear-fit.png")

coeffPoly = polyfit(yearSat, extSat, 3)  ## fit polynomial of degree 3
polynomial = poly1d(coeffPoly)
xp = arange(1970, 2040, 1)
yp = polynomial(xp)
zeroes = roots(coeffPoly)
#print zeroes
for each in zeroes:              ## find the year at which ice extent goes to zero
    if int(each.real) > 2020:
        zero = int(each.real)
print "according to polynomial fit to satellite data Arctic ice will melt in the year
fig = plt.figure(figsize=(10, 10))
plot(yearSat, extSat, 'o')
plot(xp, yp)
ylabel('extent of Arctic sea ice in square kilometers')
xlabel('year')
title(r'polynomial fit to September satellite data for Arctic ice extent ')
show()
savefig("Ice-extent-satellite-polynomial-fit.png")

# <codecell>
```

7.3 Jordan's Code

```
import numpy as np
import matplotlib.pyplot as plt
from numpy import trapz

"""
this whole first part is importing raw data.
"""
ice_data =[]
year_list = []
avg_list=[]
with open("icedata.txt", 'r') as f:

^^Ifor line in f:
^^I^^Il = line.split("^^I")
^^I^^Iif int(l[0]) <= 1950:
^^I^^I^^Iice_data.append(l[1])
^^I^^I^^Iice_data.append(l[2])
^^I^^I^^Iice_data.append(l[3])
^^I^^I^^Iavg_list.append((float(l[1])+float(l[2])+float(l[3]))/3)
^^I^^I^^Iyear_list.append(float(l[0]))

flo_ice = [float(x) for x in ice_data]
```

```
#calculate average
total_avg = []
for i in flo_ice:
^^Itotal_avg.append(np.average(flo_ice))

#make an x-axis, called p for some reason
p=[]
for year in year_list:
^^Ip.append(year+.583)
^^Ip.append(year+.666)
^^Ip.append(year+0.75)

s=[]
for i in avg_list:
^^Is.append(i)
^^Is.append(i)
^^Is.append(i)


zeroed_data = [x-total_avg[1] for x in s]


#data windows
windowed1=[]
for i in range(len(p)):
^^Iif not p[i]>=1910 or not p[i]<=1920:
^^I^^Iwindowed1.append(zeroed_data[i])
^^Ielse:
^^I^^Iwindowed1.append(np.nan)

windowed2=[]
for i in range(len(p)):
^^Iif not p[i]>=1940:
^^I^^Iwindowed2.append(zeroed_data[i])
^^Ielse:
^^I^^Iwindowed2.append(np.nan)

#mask data
masked1_y = [y for y in windowed1 if y == y]
masked1_x = [x for x in p if windowed1[p.index(x)]==windowed1[p.index(x)]]

masked2_y = [y for y in windowed2 if y == y]
masked2_x = [x for x in p if windowed2[p.index(x)]==windowed2[p.index(x)]]

#make baseline
z=np.polyfit(masked1_x,masked1_y,3)
f=np.poly1d(z)
```

```
w=np.polyfit(masked2_x,masked2_y,3)
g=np.poly1d(w)

#detrended
detrended1 = [zeroed_data[p.index(x)] - f(x) for x in p]
detrended2 = [zeroed_data[p.index(x)] - g(x) for x in p]

#total integral for each
average_integral1 = trapz(detrended1,p)
average_integral2 = trapz(detrended2,p)

#integral for 10 year window
window_integral1 = trapz(detrended1[p.index(1910.583):p.index(1919.75)])
window_integral2 = trapz(detrended2[p.index(1940.583):p.index(1949.75)])


#print results
#area under window curve compared with overall integral
#for two window choices
print window_integral1
print average_integral1

print window_integral2
print average_integral2

"""
RESULTS:

If we consider the window from 1910-1920 we get a cooling magnitude of: 6.94
compared to an average cooling of: 2.46
Relative cooling: 4.48

If we consider the window from 1940-1950 we get a cooling magnitude of: 3.250
compared to an average cooling of: 1.410
Relative cooling: 1.84
"""


"""Plots"""
#plt.plot(p,detrended1)
#plt.plot(p,detrended2)
plt.plot(p,[0]*len(p))
plt.plot(p,windowed1)
#plt.plot(p,zeroed_data)
plt.plot(p,f(p))
plt.xlim(p[0]-5,p[-1]+5)
plt.show()
```