Financial Risk Management&Regulation (M5320)
Fall 2016
https://courseworks.columbia.edu

Name : Duanhong(Gloria) Gao
UNI : DG2896

Homework 4

Due: 1:00 pm Thursday 6 Oct 2016

1. **Parametric VaR, 1 stock**

1. Parametric VaR, 1 stock The mean and standard deviation of the daily log returns of a stock are S 0.03% and 1.40%, respectively. Suppose the stock follows a geometric Brownian motion with parameters μ and σ, so that

$$dSt = \mu\text{Stdt} + \sigma\text{StdWt}$$

where Wt is a standard Brownian motion.
(a) What are good estimates for the drift and volatility, μ and σ, respectively?
(b) What is the 1 day 99% VaR of a position of 10,000 shares in the stock if the current stock price is $55 a share?

Solution:
(a)Denote the mean and the standard deviation of the daily log returns by μ and σ. Let Δt be the length of the return period. The relationship between the log return statistics and the GBM parameters is given by:

$$\sigma^* = \sigma\Delta t$$
$$\mu^* = (\mu - \sigma^2/2)\Delta t$$

Plugging in $\mu = 0.0003$, $\sigma = 0.014$, and $\Delta t = 1/252$, we get:

$$\sigma^* = \sigma\Delta t = \sigma\sqrt{252} = 0.2222$$
$$\mu^* = (\mu - \sigma^2/2)\Delta t = \mu * 252 + \sigma^2/2 = 0.1003$$

Therefore, $\sigma = 22.22\%$ and $\mu = 10.03\%$.

(b)The formula for the VaR of a stock is:

$$VaR(S, T, p) = S_0 - S_0 e^X$$
, where $X = \sigma\sqrt{T}\Phi^{-1}(1-p) + (\mu - \sigma^2/2)T$.

In our case, $S_0 = \$55, T = 1/252, p = 0.99, \sigma = 22.22\%, \mu = 10.03\%$.
The exponent is given by:

$$X = \sigma\sqrt{T}\Phi^{-1}(1-p) + (\mu - \sigma^2/2)T = \text{-0.0323}$$

The VaR is then

$$10,000 \times \$55 \times (1 - e^{-0.0323}) = \$17,481.16$$

1. **Parametric VaR 2 stocks**
   We have 550 shares of stock 1 and 300 shares of stock 2. Stock 1?s current price is $50 per share and stock 2?s current price is $100 per share. Stock 1 and 2 follow GBM with drift 4% and 2% and volatility 35% and 30%, respectively, each driven by Brownian motions with 20% correlation. What is the two week 98% VaR of the portfolio? Calculate it parametrically assuming that the portfolio is normally distributed.

Solution:
See Problem2. ipynotebook
We compute the mean and variance of the portfolio at time t = 10/252 using the formulas derived in class. Assuming the portfolio at time t is normal with the same mean and variance, we then move down from the mean by the appropriate number of standard deviations, yielding:
V0 = 57500
E-Vt = 57567.5044
E-Vt-square = 3322318448.5667
var-Vt= 8300882.2677
sd-Vt = 2881.1252
ss.norm.ppf(1-p) = -2.0537
VaR-Vt = 5849.6033

```
In [6]: ###### HW4 Problem 2
        import numpy as np
        import scipy.stats as ss
        import time
        import math

        def VaRCalculate(a, b, t, S1_0, S2_0, mu1, mu2, sigma1, sigma2, rho, p):
            V0 = a * S1_0 + b * S2_0
            E_Vt = a * S1_0 * math.exp(mu1 * t) + b * S2_0 * math.exp(mu2 * t)
            E_Vt_square = (a**2) * (S1_0**2) * math.exp( (2*mu1 + (sigma1**2)) * t) + (b**2) * (S2_0**2) * math.exp( (2*mu
            var_Vt = E_Vt_square - (E_Vt ** 2)
            sd_Vt = math.sqrt(var_Vt)
            VaR_Vt = V0 - (E_Vt - (-ss.norm.ppf(1-p) * sd_Vt) )
            return V0, E_Vt, E_Vt_square, var_Vt, sd_Vt,ss.norm.ppf(1-p), VaR_Vt

        a = 550
        b = 300
        t = 10/252
        S1_0 = 50
        S2_0 = 100
        mu1 = 0.04
        mu2 = 0.02
        sigma1 = 0.35
        sigma2 = 0.3
        rho = 0.2
        p = 0.98

        VaRCalculate(a, b, t, S1_0, S2_0, mu1, mu2, sigma1, sigma2, rho, p)

Out[6]: (57500,
         57567.504430008266,
         3322318448.5666924,
         8300882.267671108,
         2881.125173898404,
         -2.0537489106318225,
         5849.6032572795011)
```

2. **Calibration**

For this and the remaining problems, let A be AMD stock (ticker AMD), and I be Intel stock (ticker INTC). Their historical values are in the spreadsheets AMD yahoo.csv and INTC yahoo.csv, respectively, which were downloaded using the script getYahoo.sh. Let $\mu(S,t,l)$ and $\sigma(S,t,l)$ be the estimated drift rate (mean) and relative volatility parameters for a GBM process S computed on date t using the last l years worth of observations (i.e. assume dS $=\mu$Sdt $+ \sigma$SdW and that μ and σ are constant over that set of observations). Tabulate and graph μ(S,t,l) and σ(S,t,l) for S = A and S = I for t ranging over the last 20 years, and l being 2, 5, and 10 years, using unweighted fitting. Use adjusted closing prices.

Why are the computed parameters over time comparable even though the value invested varies? How stable do the results look?

Solution:

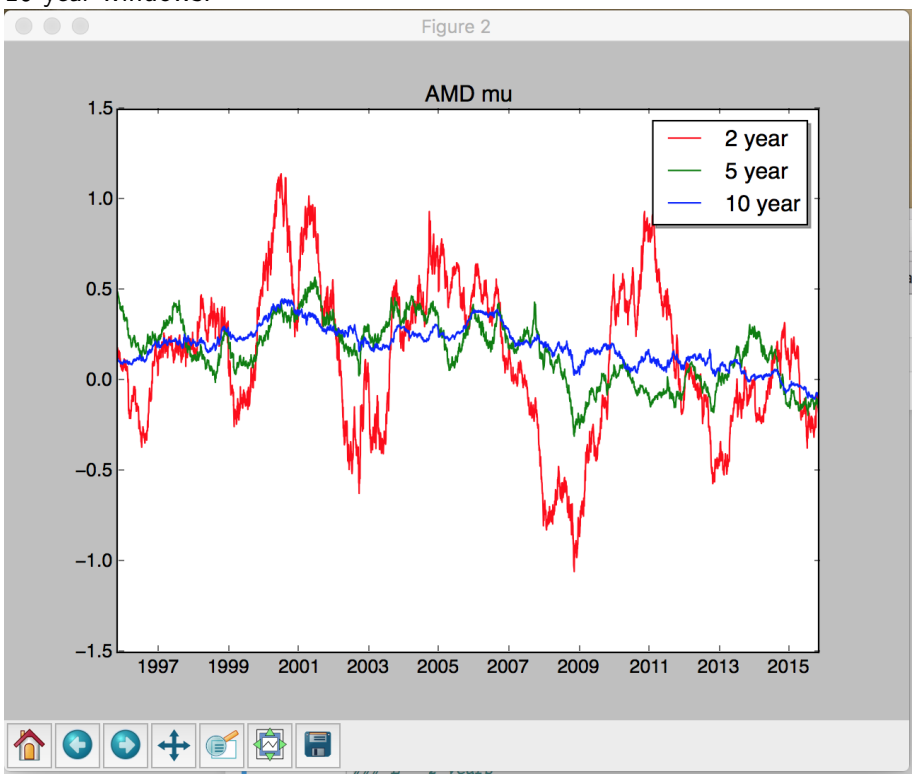
(Solution for this problem and the next one (to be split up).

See Problem3.ipynotebook.

This and the next problem can be done with python

Steps for means and averages:

1. Read the data from the data spreadsheet
2. Set up columns with log returns and log returns squared
3. For each window, the GBM vol (σ) is the STDEV() of the log returns times $\sqrt{252}$.
4. For each window, the GBM drift (μ) is the average of the log returns times 252 $+ \sigma^2/2$.
5. Set up a column with these formulas on the next 252×2 log returns to the GBM parameters for 2 year windows
6. Do the same for 252×5, and 252×10 to get the GBM parameters for the 5 year and 10 year windows.



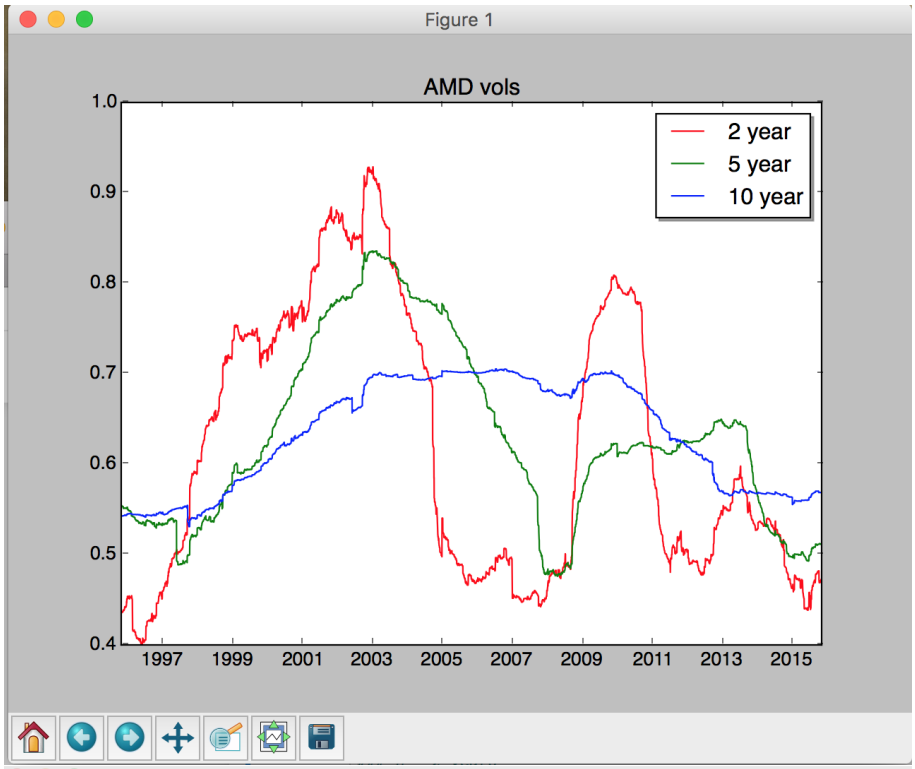


Figure 1
AMD vols
2 year
5 year
10 year
1.0
0.9
0.8
0.7
0.6
0.5
0.4
1997
1999
2001
2003
2005
2007
2009
2011
2013
2015

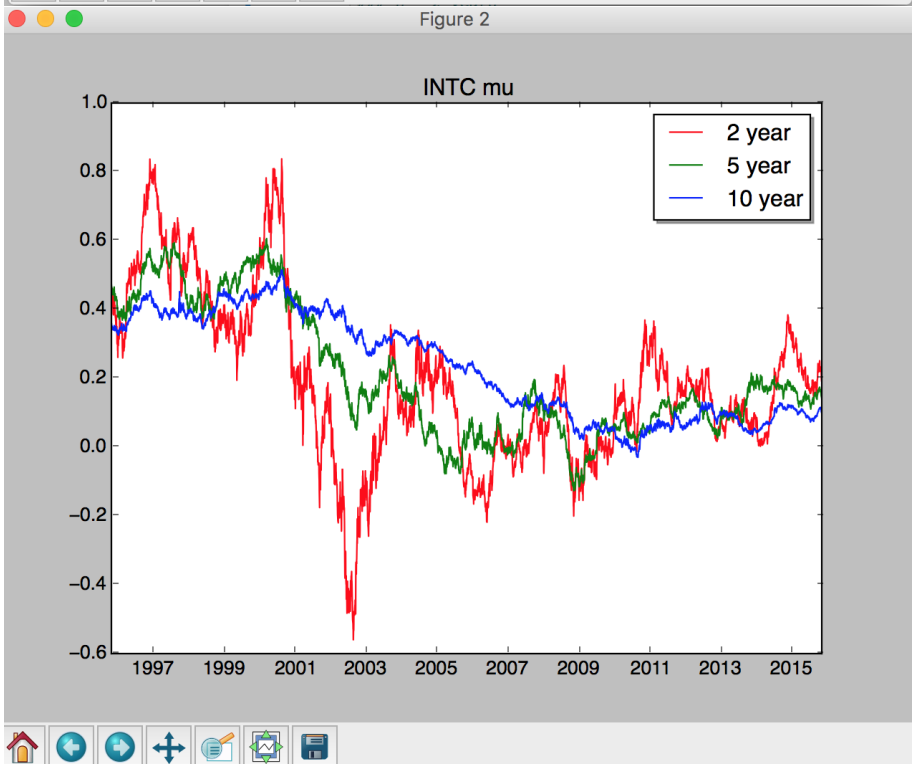
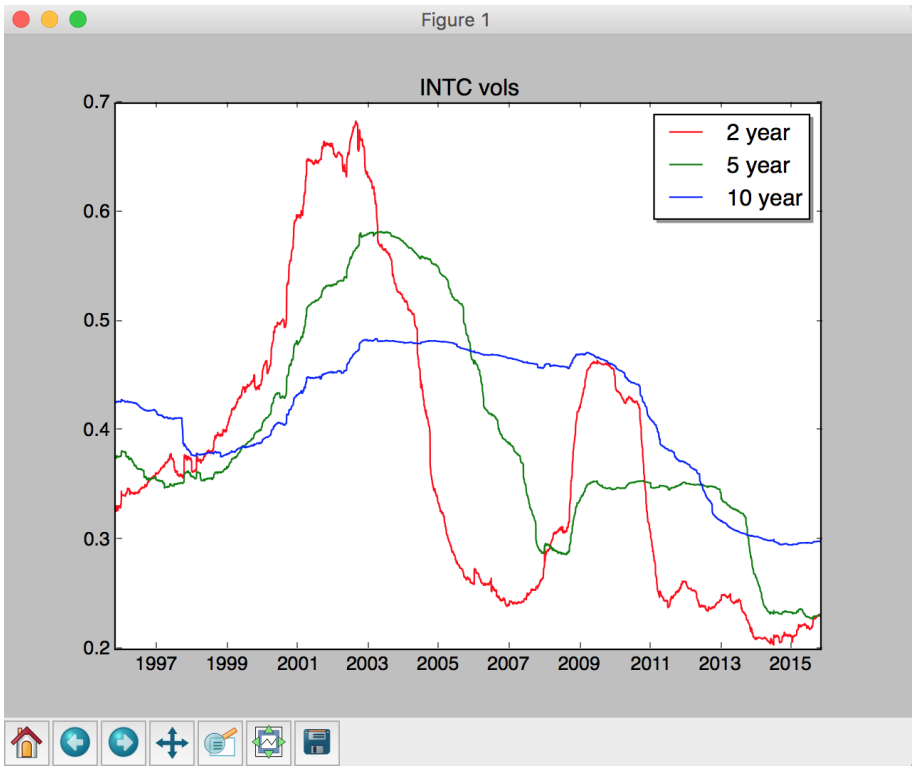


Figure 2
INTC mu
2 year
5 year
10 year
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
1997
1999
2001
2003
2005
2007
2009
2011
2013
2015




```
In [ ]: ##### HW4 Problem 3 ###### unweighted mu and vols
        import numpy as np
        import scipy.stats as ss
        import pandas as pd
        import math
        import matplotlib.pyplot as plt
        import datetime as dt
        import plotly.graph_objs as go
        import plotly.plotly as py

        path = '~/Documents/Semester3/M5320/homework/HW4/INTC-yahoo.csv'
        sto_AMD= pd.read_csv(path, header = 0)
        sto_AMD.shape
        #data1 = sto_AMD.values[1:252*20,:]
        data1 = sto_AMD.values
        #data1 = data1[::-1]
        AMD_close = list(data1[:,6])

        A_log_rtn = []
        A_log_rtn_sq = []

        for i in range(1 , len(data1)-1):
            log_return = math.log( AMD_close[i]/AMD_close[i+1] )
            A_log_rtn.append(log_return)
            A_log_rtn_sq.append(log_return**2)


        A_vol_2years = []
        A_vol_5years = []
        A_vol_10years = []

        A_mu_2years = []
        A_mu_5years = []
        A_mu_10years = []
        ### L = 2 years
        for i in range(len(data1)-252*2):
            vol_2years = np.std(A_log_rtn[i:i+252*2]) * np.sqrt(252)
            mu_2years = np.mean(A_log_rtn[i:i+252*2])*252 + (vol_2years**2)/2
            A_vol_2years.append(vol_2years)
            A_mu_2years.append(mu_2years)

        ### L = 5 years
        for i in range( len(data1)-252*5):
```

```
    vol_5years = np.std(A_log_rtn[i:i+252*5]) * np.sqrt(252)
    mu_5years = np.mean(A_log_rtn[i:i+252*5])*252 + (vol_5years**2)/2
    A_vol_5years.append(vol_5years)
    A_mu_5years.append(mu_5years)

### L = 10 years
for i in range( len(data1)-252*10):
    vol_10years = np.std(A_log_rtn[i:i+252*10]) * np.sqrt(252)
    mu_10years = np.mean(A_log_rtn[i:i+252*10])*252 + (vol_10years**2)/2
    A_vol_10years.append(vol_10years)
    A_mu_10years.append(mu_10years)

timeline = data1[1:252*20,0]
timeline = [dt.datetime.strptime(d,'%Y-%m-%d').date() for d in timeline]

A_vol_2years = A_vol_2years[1:252*20]
A_vol_5years = A_vol_5years[1:252*20]
A_vol_10years = A_vol_10years[1:252*20]

A_mu_2years = A_mu_2years[1:252*20]
A_mu_5years = A_mu_5years[1:252*20]
A_mu_10years = A_mu_10years[1:252*20]


fig, ax = plt.subplots()
ax.plot(timeline, A_vol_2years, 'r-', label='2 year')
ax.plot(timeline, A_vol_5years,'g-', label='5 year')
ax.plot(timeline, A_vol_10years, label='10 year')

legend = ax.legend(loc='upper right', shadow=True)
ax.set_title('INTC vols')


fig, ax = plt.subplots()
ax.plot(timeline, A_mu_2years, 'r-', label='2 year')
ax.plot(timeline, A_mu_5years,'g-', label='5 year')
ax.plot(timeline, A_mu_10years, label='10 year')

legend = ax.legend(loc='upper right', shadow=True)
ax.set_title('INTC mu')
plt.show()
```

3. **Exponential weighting**

   Some references say to use exponential weighting with $\lambda = 0.94$ or $\lambda = 0.97$, to cal- ibrate to the last few weeks or about a month of data. These λs correspond roughly these periods because the weight becomes $1/2$ after approximately 14 days and 22 days, respectively.

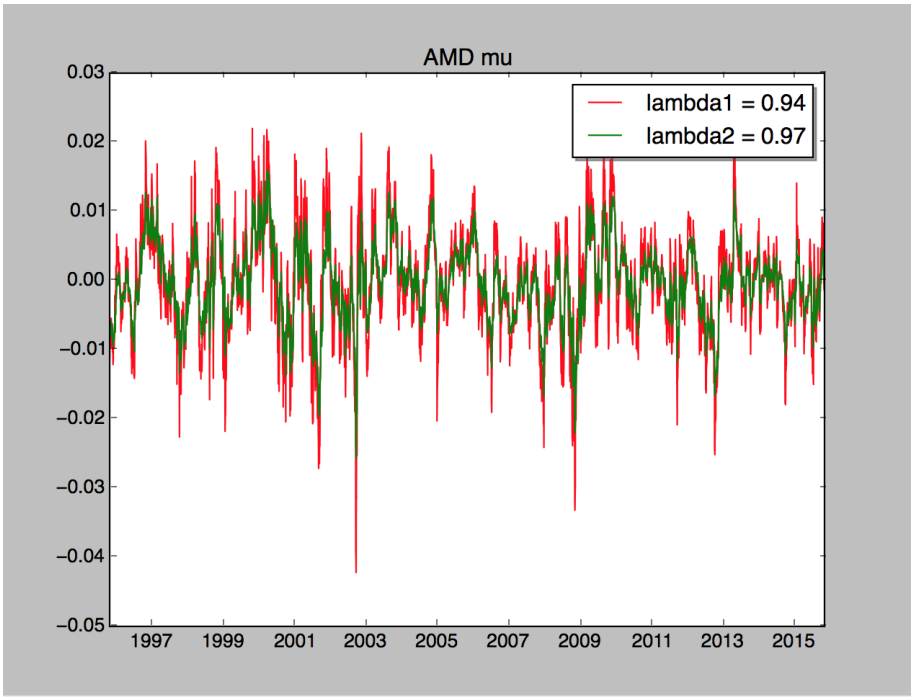
   To see the impact of exponential weighting with these weights, repeat the previous problem using exponential weighting, with $\lambda = 0.94$ and $\lambda = 0.97$.

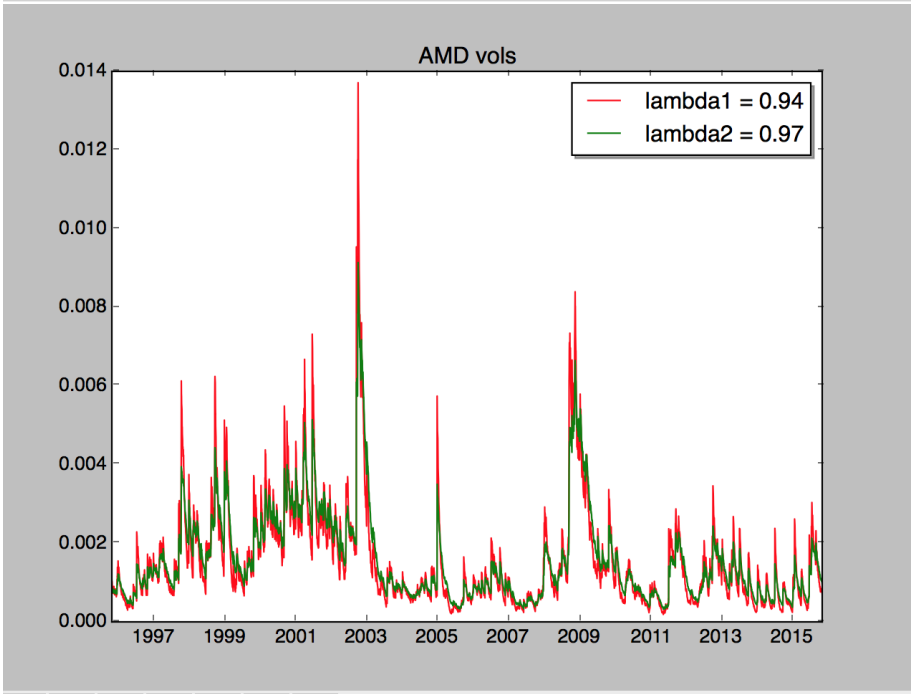
Solution:

Steps for exponentially weighted means and variances (easy, but slow method):

1. Make a column of the powers of lambda
2. Use SUM() to compute the dot product of the entire log return column starting at that date with the powers of lambda.
3. For large enough lambda, the weights drop off fast enough that we can approximate with a large window, such as with 3,000 samples.
4. Don?t forget to divide by the sum of the powers of lambda to get the weighted average
5. For the sample sigma, do the same on the square of the log returns, then subtract the square of the mean, and take the square root.
6. Then apply the formulas for converting the sample mean and average to the GBM parameter estimates.

Here are the results:


AMD mu
lambda1 = 0.94
lambda2 = 0.97
0.03
0.02
0.01
0.00
−0.01
−0.02
−0.03
−0.04
−0.05
1997
1999
2001
2003
2005
2007
2009
2011
2013
2015


AMD vols
lambda1 = 0.94
lambda2 = 0.97
0.014
0.012
0.010
0.008
0.006
0.004
0.002
0.000
1997
1999
2001
2003
2005
2007
2009
2011
2013
2015

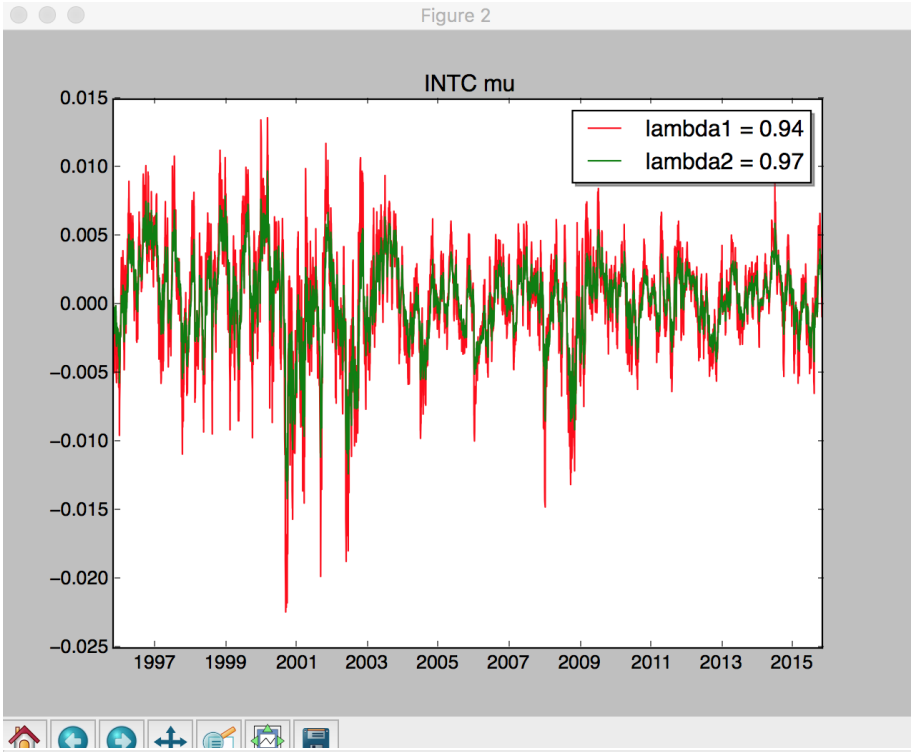


Figure 2
INTC mu
lambda1 = 0.94
lambda2 = 0.97
0.015
0.010
0.005
0.000
−0.005
−0.010
−0.015
−0.020
−0.025
1997
1999
2001
2003
2005
2007
2009
2011
2013
2015

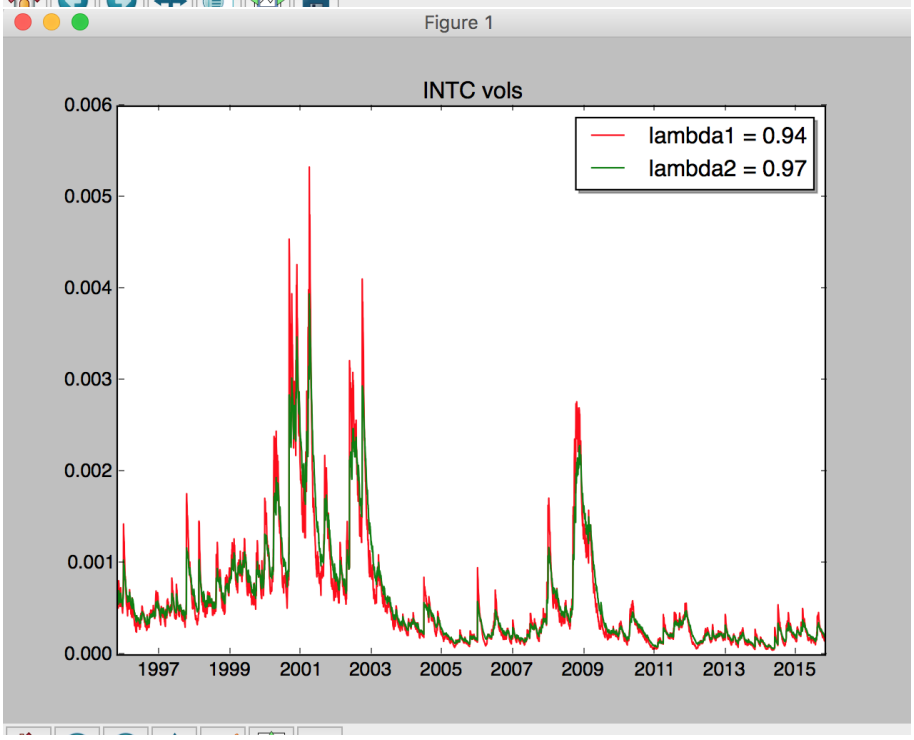


Figure 1
INTC vols
lambda1 = 0.94
lambda2 = 0.97
0.006
0.005
0.004
0.003
0.002
0.001
0.000
1997
1999
2001
2003
2005
2007
2009
2011
2013
2015

In [2]:

```
######### Problem 4 weighted mu and vols
import numpy as np
import scipy.stats as ss
import pandas as pd
import math
import matplotlib.pyplot as plt
import datetime as dt
import plotly.graph_objs as go
import plotly.plotly as py

path = '~/Documents/Semester3/M5320/homework/HW4/INTC-yahoo.csv'
sto_AMD= pd.read_csv(path, header = 0)
sto_AMD.shape
#data1 = sto_AMD.values[1:252*20,:]
data1 = sto_AMD.values
#data1 = data1[::-1]
AMD_close = list(data1[:,6])

## exponential weighting paramater lambda
lambda1 = 0.94
lambda2 = 0.97

### List the lambda values
list_lambda1 = []
list_lambda2 = []
for i in range(len(data1)):
    lambda_value = (lambda1**i)
    list_lambda1.append(lambda_value)

for i in range(len(data1)):
    lambda_value = (lambda2**i)
    list_lambda2.append(lambda_value)

wgt_A_log_rtn_1 = []
wgt_A_log_rtn_sq_1 = []

for i in range( len(data1)-1 ):
    log_return = math.log( AMD_close[i]/AMD_close[i+1] )
    wgt_log_return = log_return *  list_lambda1[i]
    wgt_A_log_rtn_1.append(wgt_log_return)

    log_return_sq = log_return ** 2
    wgt_A_log_rtn_sq_1.append( log_return_sq * list_lambda1[i] )

wgt_A_log_rtn_2 = []
wgt_A_log_rtn_sq_2 = []

for i in range( len(data1)-1 ):
    log_return = math.log( AMD_close[i]/AMD_close[i+1] )
    wgt_log_return = log_return *  list_lambda2[i]
    wgt_A_log_rtn_2.append(wgt_log_return)

    log_return_sq = log_return ** 2
    wgt_A_log_rtn_sq_2.append( log_return_sq * list_lambda2[i] )

wgt_A_vol_lambda1 = []
wgt_A_vol_lambda2 = []
wgt_A_mu_lambda1 = []
wgt_A_mu_lambda2 = []


for i in range(len(data1)-252*2):
    wgt_mu_lambda1 =  sum(wgt_A_log_rtn_1[i:i+252*2])/sum(list_lambda1[i:i+252*2])
    wgt_vol_lambda1 = sum(wgt_A_log_rtn_sq_1[i:i+252*2])/sum(list_lambda1[i:i+252*2])- wgt_mu_lambda1**2
    wgt_A_vol_lambda1.append(wgt_vol_lambda1)
    wgt_A_mu_lambda1.append(wgt_mu_lambda1)

    wgt_mu_lambda2 =  sum(wgt_A_log_rtn_2[i:i+252*2])/sum(list_lambda2[i:i+252*2])
    wgt_vol_lambda2 = sum(wgt_A_log_rtn_sq_2[i:i+252*2])/sum(list_lambda2[i:i+252*2])- wgt_mu_lambda2**2
    wgt_A_vol_lambda2.append(wgt_vol_lambda2)
    wgt_A_mu_lambda2.append(wgt_mu_lambda2)

```

```
timeline = data1[1:252*20,0]
timeline = [dt.datetime.strptime(d,'%Y-%m-%d').date() for d in timeline]

wgt_A_vol_lambda1 = wgt_A_vol_lambda1[1: 252*20]
wgt_A_vol_lambda2 = wgt_A_vol_lambda2[1: 252*20]

wgt_A_mu_lambda1 = wgt_A_mu_lambda1[1: 252*20]
wgt_A_mu_lambda2 = wgt_A_mu_lambda2[1: 252*20]

timeline = data1[1:252*20,0]
timeline = [dt.datetime.strptime(d,'%Y-%m-%d').date() for d in timeline]


fig, ax = plt.subplots()
ax.plot(timeline, wgt_A_vol_lambda1, 'r-', label='lambda1 = 0.94')
ax.plot(timeline, wgt_A_vol_lambda2, 'g-', label='lambda2 = 0.97')

legend = ax.legend(loc='upper right', shadow=True)
ax.set_title('INTC vols')


fig, ax = plt.subplots()
ax.plot(timeline, wgt_A_mu_lambda1, 'r-', label='lambda1 = 0.94')
ax.plot(timeline, wgt_A_mu_lambda2,'g-', label='lambda2 = 0.97')

legend = ax.legend(loc='upper right', shadow=True)
ax.set_title('INTC mu')
plt.show()


```