from alpha\_vantage.timeseries import TimeSeries  
import pickle  
  
ts = TimeSeries(key='33VUO7M2H5J562FP',output\_format='pandas', indexing\_type='date')  
  
filename = ['vaw','vcr','vdc','vde','vfh','vgt','vht','vis','vnq','vox','vpu','spy','vti']  
datafile\_count = len(filename)  
sector\_rawdata = {}  
min\_length = 10000  
  
for i, file in enumerate(filename):  
 print "Reading data for: ",filename[i]  
 rawdata, meta\_data = ts.get\_weekly(symbol=filename[i])  
 pickle.dump(rawdata, open( filename[i]+".p", "wb" ) )  
 sector\_rawdata[i] = rawdata.values  
 min\_length = min(min\_length,sector\_rawdata[i].shape[0])  
  
  
print min\_length  
  
import numpy as np  
import math  
import scipy.optimize as opt  
import matplotlib.pylab as plt  
import pickle  
  
# This optimization takes about 1 minute per iteration yielding 520 minutes (about 9 hours) for our ten years of data  
  
# Gauss Hermite computation in four dimensions  
def computeR(xti):  
 Jk = 0  
 expterm = [0,0,0,0]  
 for k1 in range(0,ghsize):  
 for k2 in range(0,ghsize):  
 for k3 in range(0,ghsize):  
 for k4 in range(0,ghsize):  
 for i in range(0, dimension):  
 qvector = [nodes[k1],nodes[k2],nodes[k3],nodes[k4]]  
 lastTerm = math.sqrt(2) \* deviation[i] \* np.sum(np.transpose(qvector)\*LowerSigma[i,:],axis = 0)  
 expterm[i] = np.exp(mu[i] - (deviation[i]\*\*2)/2 + lastTerm)  
  
 Jk += weights[k1]\*weights[k2]\*weights[k3]\*weights[k4]\*sum(np.transpose(expterm) \* xti)  
  
 return -Jk \* math.pi\*\*(-0.5\*dimension)  
  
# Optimization constraint that the sum of all inputs is equal to 1  
def apply\_sum\_constraint(inputs):  
 total = 1.0 - np.sum(inputs)  
 return total  
  
# Load financial data that has been saved earlier (11 sectors)  
filename = ['vaw','vcr','vdc','vde','vfh','vgt','vht','vis','vnq','vox','vpu']  
sector\_rawdata = []  
sector = []  
mu\_sigma\_sector = []  
min\_length = 10000  
for i in range(0,11):  
 print "Reading data for: ",filename[i]  
 sector\_rawdata.append(pickle.load(open(filename[i]+".p", "rb" )))  
 min\_length = min(min\_length,sector\_rawdata[i].shape[0])  
  
#check minimum length of time series is more than 11 years (572 weeks)  
print "Minimum Length of data series:", min\_length  
# Compute the mean (mu) and standard deviation (sigma) for each sector over varying periods (either 3, 6 or 12 months)  
duration = [39, 26, 0]  
for i in range(0,11):  
 sectorData = sector\_rawdata[i]["4. close"][-573:]  
 diff = sectorData.pct\_change()  
 sector.append(diff)  
 mu\_sigma = []  
 for j in range(1,573-51):  
 mu\_sigma.append([np.mean(diff[j+duration[2]:j+52]), np.std(diff[j+duration[2]:j+52])])  
 mu\_sigma\_sector.append(mu\_sigma)  
  
# initialize results arrays  
resultsPosition = []  
resultsSector = []  
resultsMu = []  
# Generate Gauss Hermite quadrature, number of points m  
ghsize = 10  
nodes, weights = np.polynomial.hermite.hermgauss(ghsize)  
  
# We will optimize our model parametrized function for the next 10 years (520 weeks)  
for k in range(0,520):  
 mu = []  
 # compute the mu of log(R) = mu - sigma\*sigma/2  
 for j in range(0,11):  
 mu.append(mu\_sigma\_sector[j][k][0] - (mu\_sigma\_sector[j][k][1]\*\*2)/2)  
 resultsMu.append(mu)  
 # we will limit ourselves to the four largest mu of log(R) as we limit ourselves to optimize in four dimensions  
 mu\_sort = np.argsort(mu)  
 print "Sorted sector by largest mu:", mu\_sort  
 sector1 = sector[mu\_sort[-1]][k+1+duration[2]:k+53]  
 sector2 = sector[mu\_sort[-2]][k+1+duration[2]:k+53]  
 sector3 = sector[mu\_sort[-3]][k+1+duration[2]:k+53]  
 sector4 = sector[mu\_sort[-4]][k+1+duration[2]:k+53]  
  
 # prepare the data array to compute our means, sigmas and correlation matrix  
 dataArray = np.array([sector1, sector2, sector3, sector4])  
 Sigma = np.corrcoef(dataArray)  
 mu = np.mean(dataArray,axis=1)  
 deviation = np.std(dataArray,axis=1)  
 dimension = len(mu)  
 print "Top 4 mu and sigma: ",mu,deviation  
 print "Correlation matrix: ", Sigma  
 # compute the lower triangle correlation matrix  
 LowerSigma = np.linalg.cholesky(Sigma)  
 print "Lower triangular matrix: ", LowerSigma  
  
 # we optimize the states with values between 0 and 1 and with the constraint that the sum of the states equals one  
 bnds = ((0,1),(0,1),(0,1),(0,1))  
 constraints = ({'type': 'eq', "fun": apply\_sum\_constraint })  
  
 print computeR([1,0,0,0]), computeR([0,1,0,0]),computeR([0,0,1,0]), computeR([0,0,0,1])  
 res = opt.minimize(computeR, [0,0,0,0], bounds=bnds, tol=1e-7, constraints = constraints)  
  
 # store the results for further analysis  
 resultsPosition.append(res.x)  
 resultsSector.append(mu\_sort[::-1][:4])  
  
 print resultsPosition,resultsSector  
  
# save the final results  
pickle.dump(resultsPosition, open( "resultsPosition1.p", "wb" ) )  
pickle.dump(resultsSector, open( "resultsSector1.p", "wb" ) )  
pickle.dump(resultsMu, open( "resultsMu1.p", "wb" ) )  
  
import numpy as np  
import math  
import matplotlib.pylab as pylab  
import pickle  
import copy  
  
# Results 2: sort by mu of R on 12 months basis  
# Results 3: sort by mu of log(R) on 12 months basis  
# Results 4: sort by mu of log(R) on 6 months basis  
# Results 5: sort by mu of log(R) on 3 months basis  
  
# Build and Modify Portfolio investment each week and check returns againt SPY (S&P 500), VTI (Vanguard Total US market) and Exact Solution  
  
spy\_value = np.zeros(520)  
vti\_value = np.zeros(520)  
exact\_value = np.zeros(520)  
  
# Load financial data that has been saved earlier  
filename = ['vaw','vcr','vdc','vde','vfh','vgt','vht','vis','vnq','vox','vpu']  
sectorName = ['Materials','Consumer Discretionary','Consumer Staples','Energy','Financial','Technologies','Health Care','Industrials','Real Estate','Communications','Utilities']  
  
sector\_rawdata = []  
sector = []  
mu\_sigma\_sector = []  
min\_length = 10000  
for i in range(0,11):  
 print "Reading data for: ",sectorName[i]  
 sector\_rawdata.append(pickle.load(open(filename[i]+".p", "rb" )))  
 min\_length = min(min\_length,sector\_rawdata[i].shape[0])  
  
spy\_rawdata = pickle.load(open("spy.p", "rb" ))  
vti\_rawdata = pickle.load(open("vti.p", "rb" ))  
  
  
# These are the results from the value function (Jk) optimization  
resultsPosition = pickle.load(open( "resultsPosition5.p", "rb" ) )  
resultsSector = pickle.load(open( "resultsSector5.p", "rb" ) )  
resultsMu = pickle.load(open( "resultsMu5.p", "rb" ) )  
  
# extract only prices from the full datasets (for all sectors)  
sectorPrices = []  
for i in range(0,11):  
 sectorData = sector\_rawdata[i]["4. close"][-520:]  
 sectorPrices.append(sectorData)  
spyPrices = spy\_rawdata["4. close"][-520:]  
vtiPrices = vti\_rawdata["4. close"][-520:]  
  
  
  
# This section computes the investment results starting with $1,000,000  
# Jk is our optimized function, Jkversion2 is an approximation  
# spy\_value is the S&P500 index value  
# exact\_value is the exact solution computed by investing the full amount in the best performing sector each week  
  
# various initializations  
portf\_value = np.zeros(520)  
Jk = np.zeros(520)  
Jkversion2 = np.zeros(520)  
bestPerformance = np.zeros((520,11))  
state\_x = [0]\*11  
next\_state\_x = [0]\*11  
policy\_u = []  
  
spy\_value[0] = 1000000  
vti\_value[0] = 1000000  
exact\_value[0] = 1000000  
Jk[0] = 1000000  
Jkversion2[0] = 1000000  
  
# compute S&P500 index and Exact solution values  
for i in range(0,519):  
 spy\_share = spy\_value[i]/spyPrices[i]  
 spy\_value[i+1] = spy\_share \* spyPrices[i+1]  
 vti\_share = vti\_value[i]/vtiPrices[i]  
 vti\_value[i+1] = vti\_share \* vtiPrices[i+1]  
 for j in range(0,10):  
 bestPerformance[i][j] = (sectorPrices[j][i+1] - sectorPrices[j][i]) / sectorPrices[j][i]  
 exact\_value[i+1] = exact\_value[i] \* (np.max(bestPerformance[i]) + 1.0)  
  
# compute Jk (our parametrized function)  
for i in range(0,519):  
 Jk[i+1] = 0  
 portf\_share = [0,0,0,0]  
 state\_x = copy.deepcopy(next\_state\_x)  
 policy = []  
 actionList = []  
 for j in range(0,4):  
 index = int(resultsSector[i][j])  
 if resultsPosition[i][j] > 1e-2:  
 actionList.append(index)  
 # compute new state x\_ti  
 portf\_share[j] = Jk[i]\*resultsPosition[i][j]/sectorPrices[index][i]  
 next\_state\_x[index] = portf\_share[j]  
 stateChange = next\_state\_x[index] - state\_x[index]  
 # compute policy which is to buy and sell various sectors  
 if stateChange >= 1:  
 policy.append("Buy " + str(stateChange) + " shares of " + str(sectorName[index]) + ", ")  
 if stateChange <= -1:  
 policy.append("Sell " + str(-stateChange) + " shares of " + str(sectorName[index]) + ", ")  
 Jk[i+1] += portf\_share[j] \* sectorPrices[index][i+1]  
  
 # we have to remove the sector we had invested in and which are not selected this week  
 for j in range(0,10):  
 if j not in actionList:  
 stateChange = - state\_x[j]  
 next\_state\_x[j] = 0  
 if stateChange <= -1:  
 policy.append("Sell " + str(-stateChange) + " shares of " + str(sectorName[j]) + ", ")  
 policy\_u.append(policy)  
 #print portf\_share, Jk[i+1], spy\_value[i+1], exact\_value[i+1]  
  
# compute the Jk approximation based on equal investment in the four sectors with highest mu of log(R)  
equalShare = [0.25,0.25,0.25,0.25]  
for i in range(0,519):  
 Jkversion2[i+1] = 0  
 for j in range(0,4):  
 index = int(resultsSector[i][j])  
 portf\_share[j] = Jkversion2[i]\*equalShare[j]/sectorPrices[index][i]  
 Jkversion2[i+1] += portf\_share[j] \* sectorPrices[index][i+1]  
  
  
# Print the policy and compare to exact solution for each week  
for i in range(0,519):  
 print ""  
 print "Week: ", i  
 print "Policy: ",policy\_u[i]  
 print "Max Mu: ", np.max(resultsMu[i]), " for ETF: ", sectorName[np.argmax(resultsMu[i])]  
 index = np.argmax(bestPerformance[i])  
 print ""  
 print "Best Investment was: ", sectorName[index], " with Mu of : ", resultsMu[i][index]  
 print ""  
  
# Plotting section  
  
# Plot Jk and our Jk approximation  
weeks\_axis = np.arange(2009.0+5./12.,2019.0+5./12.,1./52.)  
dollar = np.column\_stack([Jk,Jkversion2,spy\_value])  
  
pylab.plot(weeks\_axis,dollar)  
pylab.grid(True) # affiche un grille  
pylab.xlabel('Weekly over 10 years')  
pylab.ylabel("Investment Results")  
pylab.legend(('DP Optimal', 'DP Top 4 Model','S&P500'))  
pylab.title('DP Sector Investment Result vs S&P500 each week')  
  
# Plot whether our Jk beats or loses compared to our benchmark (S&P500)  
profit\_loss = np.column\_stack([Jk-spy\_value,Jkversion2-spy\_value])  
pylab.show()  
pylab.plot(weeks\_axis,profit\_loss)  
pylab.grid(True) # affiche un grille  
pylab.xlabel('Weekly over 10 years')  
pylab.ylabel("Profit/Loss")  
pylab.legend(('DP Optimal', 'DP Top 4 Model'))  
  
pylab.title("DP Profit and Loss vs S&P500 each week")  
  
pylab.show()  
  
# Plot Exact Solution vs S&P500 using a log scale as the exact solution beats the market by a wide margin  
logplot = np.column\_stack([exact\_value,spy\_value])  
pylab.plot(weeks\_axis,logplot)  
pylab.grid(True) # affiche un grille  
pylab.xlabel('Weekly over 10 years')  
pylab.ylabel("DP Exact Solution")  
pylab.yscale('log')  
pylab.title("Log plot of DP Exact Solution vs S&P500")  
  
pylab.show()  
  
# Plot best Mu of log(R) each week  
maxMu = np.max(resultsMu,axis=1)  
pylab.plot(weeks\_axis,maxMu)  
pylab.grid(True) # affiche un grille  
pylab.xlabel('Weekly over 10 years')  
pylab.ylabel("Maximum Mu")  
  
pylab.title("Mu of log(R) each week")  
  
pylab.show()  
  
# Plot Sector Prices to get an idea of the trends...  
prices = np.column\_stack(sectorPrices)  
pylab.plot(weeks\_axis,prices)  
pylab.grid(True) # affiche un grille  
pylab.xlabel('Weekly over 10 years')  
pylab.ylabel("All Sectors Performance")  
pylab.legend(sectorName,bbox\_to\_anchor=(1., 1.))  
  
pylab.title("DP Sector Prices each week")  
  
pylab.show()