MA 573
HW 7
Alex Shoop

Question 6)
(this is my old Q6 code from last week. My new code will be submitted in HW8)

```python
# importing the necessary packages
import numpy as np
from numpy import sqrt, exp, log, mean
from scipy.stats import norm
from scipy import optimize
import matplotlib.pyplot as plt

###############################################################################

# increment step method for Brownian Motion
def SimBMStep(T, N):
    # initialize W brownian motion array
    W = np.zeros(int(N))
    W = np.append(W, 0)    # initial 0 value

    # loop
    num = 0
    while num < N:
        W[num + 1] = W[num] + sqrt(T/N)*np.random.standard_normal()
        num += 1

    return W

# initial vars
s = 139.    # retrieved from APPL stock
#s = 100.
y = 0.08
lam = 3
kappa = 0.1
rho = -0.8
xi = 0.25

# initial vars array
x0 = np.array([lam, kappa, xi, y, rho])

# number retrieved from 30yr treasury bond yield curve rate
r = 0.0320

# day form
T = 60/365.
runs = 1000

# this is all XDATA
# these are the K strike CALL options that have trading volume of more than 10
CK = np.array([50, 130, 135, 140, 145, 150, 155, 160, 165, 170, 180])
# these are the K strike PUT options that have trading volume of more than 10
PK = np.array([70, 75, 80, 100, 105, 110, 115, 120, 125, 130, 135, 140, 145, 150])

#CK_thetaPrices = np.zeros(0)
CK_actualPrices = np.array([mean([96.85,97.60]),
                            mean([10.55,10.60]),
                            mean([6.80,6.90]),
                            mean([3.90,4.00]),
                            mean([2.04,2.06]),
                            mean([0.95,0.97]),
                            mean([0.43,0.44]),
                            mean([0.20,0.21]),
                            mean([0.09,0.10]),
                            mean([0.04,0.05]),
                            mean([0.04,0.05])])

#PK_thetaPrices = np.zeros(0)
PK_actualPrices = np.array([mean([0, 0.03]),
                            mean([0.02,0.04]),
                            mean([0.01,0.05]),
                            mean([0.06,0.07]),
                            mean([0.11,0.12]),
                            mean([0.16,0.17]),
                            mean([0.27,0.28]),
                            mean([0.45,0.47]),
                            mean([0.81,0.84]),
                            mean([1.55,1.57]),
                            mean([2.89,2.92]),
                            mean([5.10,5.15]),
                            mean([8.20,8.30]),
                            mean([11.90,12.25])])

print("K strike values for Call options (these have volume traded > 10):")
print(CK)
print("Call option actual prices (ordered with their respective K's): ")
print(CK_actualPrices)

print("K strike values for Put options:")
print(PK)
print("Put option actual prices are: ")
print(PK_actualPrices)

# payoff function for European Call option
def payoffEC(ST, K):
    return max(ST - K, 0)

# payoff function for European Put option
def payoffEP(ST, K):
    return max(K - ST, 0)

# big MAIN FUNCTION that has all parameters for optimization
# and outputs the estimated model prices
# xdata is the K value
def PricingUsingModelEC(K, Plam, Pkappa, Pxi, Py, Prho):
        #lam, kappa, xi, y, rho, = params
        #errVal = abs(ydata - CThetha)

        # K loop?
        CK_estimatedPrices = np.zeros(0)
        i = 0
        while i < K.size:
            # array holding payoffs for price calculation
            payoffsArray = np.zeros(0)
            num = 0
            while num < runs:
                # get W1 and W~ Brownian Motions, in order to get the W2 BM
                # note: need to use rho correlation equation
                W1 = SimBMStep(T, 500.)
                Wtilda = SimBMStep(T, 500.)
                W2 = Prho*W1 + sqrt(1 - Prho**2)*Wtilda

                # simulating Yt values and St values
                temp = 0
                Y = np.zeros(0)
```

```python
            # getting initial Y_0, which is little y = -1
            Y = np.append(Y, Py)
            St = np.zeros(0)
            St = np.append(St, s)
            # loop for Yt and St simulation
            # NOTE, Yt follows CIR process form
            while temp < 500:
                Y = np.append(Y, max(Y[temp]
                            + Plam*(Pkappa - Y[temp])*(T/500.)
                            + Pxi*sqrt(Y[temp])*(W2[temp+1] - W2[temp]), 0))
                St = np.append(St, St[temp]
                        + r*St[temp]*(T/500.)
                        + sqrt(Y[temp])*St[temp]*(W1[temp+1] - W1[temp])
                        + (0.5 * sqrt(Y[temp])*sqrt(Y[temp])*St[temp]*((W1[temp+1] - W1[temp])**2 - (T/500.))))

                temp += 1
#
            ST = St[St.size - 1]

            # calculating payoff
            #print("K size is: ")
            #print(K.size)
            payment = payoffEC(ST, K[i])
            payoffsArray = np.append(payoffsArray, payment)

            num += 1
        # calculating price
        EstimatedPrice = exp(-r*T)*np.mean(payoffsArray)
        print("estimated price is: " + str(EstimatedPrice))
        CK_estimatedPrices = np.append(CK_estimatedPrices, EstimatedPrice)
        i += 1
    return CK_estimatedPrices


# finally, we do curve_fit optimization
popt, pcov = optimize.curve_fit(PricingUsingModelEC, CK, CK_actualPrices, x0)
print(popt)
```