MA 573
HW 8
Alex Shoop

Question 3)
Code:

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

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

# 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 = 143.    # retrieved from APPL stock
sigma_init = 3.0   # initial guess for sigma

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

# day form
T = 36/365.
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, 110, 120, 130, 135, 140, 145, 150, 155, 160, 165, 170, 175, 180])

#CK_thetaPrices = np.zeros(0)
CK_actualPrices = np.array([mean([86.40,87.10]),
                            mean([33.50,34.25]),
                            mean([24.15,24.45]),
                            mean([14.50,14.65]),
                            mean([10.15,10.25]),
                            mean([6.50,6.55]),
                            mean([3.75,3.80]),
                            mean([1.89,1.89]),
                            mean([0.89,0.91]),
                            mean([0.42,0.44]),
                            mean([0.21,0.22]),
                            mean([0.10,0.11]),
                            mean([0.05,0.06]),
                            mean([0.02,0.03])])


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)

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

# big MAIN FUNCTION that has all parameters for optimization
# and outputs the estimated model prices
# xdata is the K value
def PricingUsingModelEC(K, sigma_init):
    num = 0
    totalPayoffs = np.zeros_like(CK)
    # monte carlo simulation loop
    while num < runs:
        payoffs = np.zeros(0)
        W1 = SimBMStep(T, 500.)
        temp = 0
        St = np.zeros(0)
        St = np.append(St, s)
        # St simulation loop
        while temp < 500:
            St = np.append(St, St[temp]
                                 + r*St[temp]*(T/500.)
                                 + sigma_init*St[temp]*(W1[temp+1] - W1[temp]))
            temp += 1
        # retrieving the terminal ST
        ST = St[St.size - 1]
        payoffs = ST - K
        # this makes every payoff value >= 0
        payoffs = [0 if i<0 else i for i in payoffs]
        totalPayoffs = np.sum([payoffs, totalPayoffs], axis = 0)
        num += 1
    AveragePayoffs = [i/runs for i in totalPayoffs]
    EstimatedPrices = [exp(-r*T)*i for i in AveragePayoffs]
    return EstimatedPrices

# regular BS pricing formula
def PricingUsingClosedForm(K, sigma_init):
    d1 = (log(s/K) + (r + (sigma_init**2)/2)*T) / (sigma_init * sqrt(T))
    d2 = d1 - sigma_init*sqrt(T)
    # black scholes formula price calculation
    PriceBS = s*norm.cdf(d1) - K*exp(-r*T)*norm.cdf(d2)
    return PriceBS


    # finally, we do curve_fit optimization, Monte Carlo integration version
    popt, pcov = optimize.curve_fit(PricingUsingModelEC, CK, CK_actualPrices)
    print("MC: If initial sigma guess = " + str(sigma_init) + ", then optimal sigma = " + str(round(popt,2)))

    # BS formula version
    popt, pcov = optimize.curve_fit(PricingUsingClosedForm, CK, CK_actualPrices)
    print("BS: If initial sigma guess = " + str(sigma_init) + ", then optimal sigma = " + str(round(popt,2)))

```

MA 573
HW 8
Alex Shoop

a)
Output:
```
K strike values for Call options (these have volume traded > 10):
[ 50 110 120 130 135 140 145 150 155 160 165 170 175 180]
Call option actual prices (ordered with their respective K's):
[  8.67500000e+01   3.38750000e+01   2.43000000e+01   1.45750000e+01
   1.02000000e+01   6.52500000e+00   3.77500000e+00   1.89000000e+00
   9.00000000e-01   4.30000000e-01   2.15000000e-01   1.05000000e-01
   5.50000000e-02   2.50000000e-02]

MC: If initial sigma guess = 0.2, then optimal sigma = 1.0
MC: If initial sigma guess = 0.3, then optimal sigma = 1.0
MC: If initial sigma guess = 0.02, then optimal sigma = 1.0
MC: If initial sigma guess = 3.0, then optimal sigma = 1.0
```

b)
Output:
```
BS: If initial sigma guess = 0.2, then optimal sigma = 0.26
BS: If initial sigma guess = 0.3, then optimal sigma = 0.26
BS: If initial sigma guess = 0.02, then optimal sigma = 0.26
BS: If initial sigma guess = 3.0, then optimal sigma = 0.26
```

c)
As clearly seen, the Monte Carlo model calibration for sigma resulted in 1.0 while the closed form Black Scholes formula model calibration for sigma resulted in 0.26.
Naturally 0.26 seems like the reasonable result.

I believe because the Monte Carlo routine uses a large size of data for the simulation process, this creates discrepancies. In addition, curve_fit optimization method uses the Levenberg-Marquardt algorithm, which expects smooth and “well-behaved” statistical world. However the Monte Carlo simulation process is not a simple world, therefore it is difficult for the curve_fit optimization method to work properly with Monte Carlo integration.