MA 573
HW 6
Alex Shoop

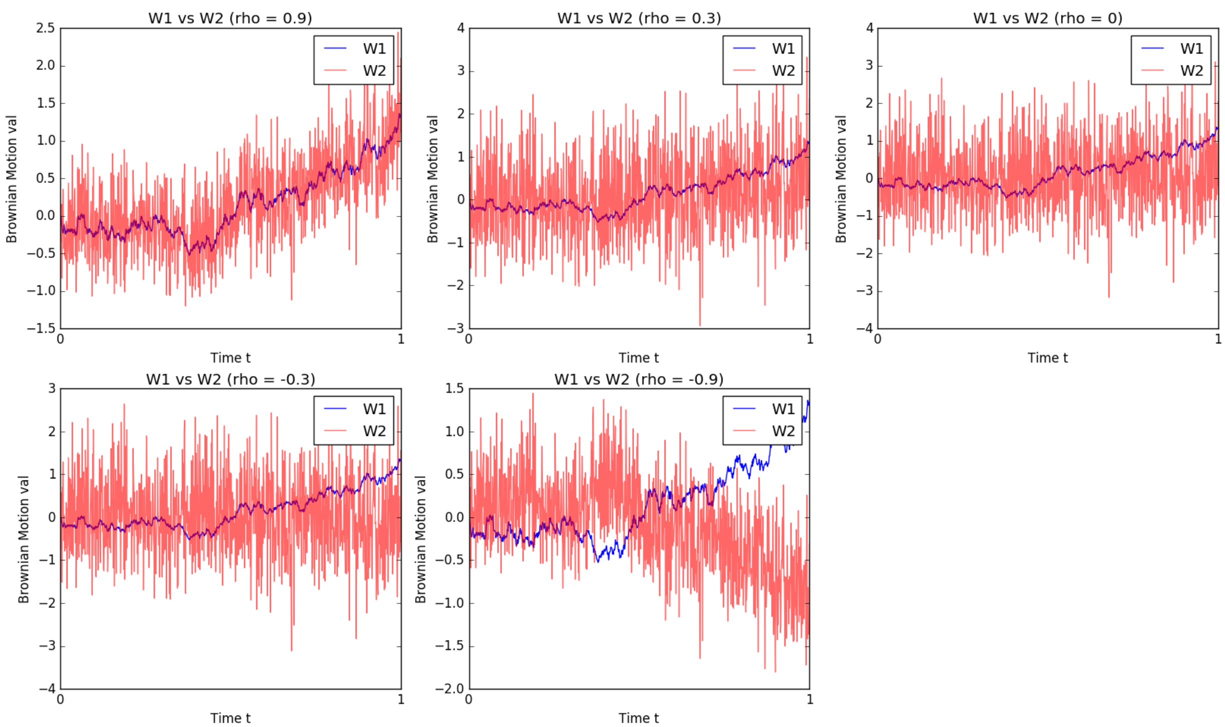
Question 1) c)

```python
# importing the necessary packages
import numpy as np
from numpy import sqrt
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
rho1 = 0.9
rho2 = 0.3
rho3 = 0
rho4 = -0.3
rho5 = -0.9
# setting up t x-axis variable
t = np.arange(0.0, 1001.0, 1.0)

W1 = SimBMStep(1, 1000.)
Z = np.random.standard_normal(1001)

plt.clf()

###################################
plt.subplot(2,3,1)

# W2 part i
W2_i = rho1*W1 + sqrt(1 - rho1**2)*Z

plt.plot(t,W1, color = 'blue', label = 'W1')
plt.plot(t,W2_i, color = 'red', alpha = 0.6, label = 'W2')
plt.title("W1 vs W2 (rho = 0.9)")

# this is to label the x-axis as 0 to 1 time t
ticks = np.arange(t.min(), t.max() + 1, 1000)
labels = range(ticks.size)
plt.xticks(ticks, labels)
plt.xlabel("Time t")
plt.ylabel("Brownian Motion val")
plt.legend()

###################################
plt.subplot(2,3,2)

```

```
# W2 part ii
W2_ii = rho2*W1 + sqrt(1 - rho2**2)*Z

plt.plot(t,W1, color = 'blue', label = 'W1')
plt.plot(t,W2_ii, color = 'red', alpha = 0.6, label = 'W2')
plt.title("W1 vs W2 (rho = 0.3)")

# this is to label the x-axis as 0 to 1 time t
ticks = np.arange(t.min(), t.max() + 1, 1000)
labels = range(ticks.size)
plt.xticks(ticks, labels)
plt.xlabel("Time t")
plt.ylabel("Brownian Motion val")
plt.legend()

####################################
plt.subplot(2,3,3)

# W2 part iii
W2_iii = rho3*W1 + sqrt(1 - rho3**2)*Z

plt.plot(t,W1, color = 'blue', label = 'W1')
plt.plot(t,W2_iii, color = 'red', alpha = 0.6, label = 'W2')
plt.title("W1 vs W2 (rho = 0)")

# this is to label the x-axis as 0 to 1 time t
ticks = np.arange(t.min(), t.max() + 1, 1000)
labels = range(ticks.size)
plt.xticks(ticks, labels)
plt.xlabel("Time t")
plt.ylabel("Brownian Motion val")
plt.legend()

####################################
plt.subplot(2,3,4)

# W2 part iv
W2_iv = rho4*W1 + sqrt(1 - rho4**2)*Z

plt.plot(t,W1, color = 'blue', label = 'W1')
plt.plot(t,W2_iv, color = 'red', alpha = 0.6, label = 'W2')
plt.title("W1 vs W2 (rho = -0.3)")

# this is to label the x-axis as 0 to 1 time t
ticks = np.arange(t.min(), t.max() + 1, 1000)
labels = range(ticks.size)
plt.xticks(ticks, labels)
plt.xlabel("Time t")
plt.ylabel("Brownian Motion val")
plt.legend()

####################################
plt.subplot(2,3,5)

# W2 part v
W2_v = rho5*W1 + sqrt(1 - rho5**2)*Z

plt.plot(t,W1, color = 'blue', label = 'W1')
plt.plot(t,W2_v, color = 'red', alpha = 0.6, label = 'W2')
plt.title("W1 vs W2 (rho = -0.9)")

# this is to label the x-axis as 0 to 1 time t
```

```
        ticks = np.arange(t.min(), t.max() + 1, 1000)
        labels = range(ticks.size)
        plt.xticks(ticks, labels)
        plt.xlabel("Time t")
        plt.ylabel("Brownian Motion val")
        plt.legend()
```

Figure:



Question 2)

```python
# importing the necessary packages
import numpy as np
from numpy import sqrt, exp
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 = 100.
y = -1
lam = 5
kappa = -1.5
rho = -0.2
xi = 0.25
r = 0.03
T = 0.3
runs = 1000

# payoff function for Lookback Put option
def payoffLP(Smax, ST):
    return max(Smax - ST, 0)

# 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 = rho*W1 + sqrt(1 - rho**2)*Wtilda

    # simulating Yt values
    temp = 0
    Y = np.zeros(0)
    # getting initial Y_0, which is little y = -1
    Y = np.append(Y, y)
    # loop for Yt simulation
    while temp < 500:
        Y = np.append(Y, Y[temp] + lam*(kappa - Y[temp])*(T/500.) + xi*(W2[temp+1] - W2[temp]))
        temp += 1

    # St simulation (MILSTEIN SCHEME)
    # derivative of exp(Yt)*St is just exp(Yt)
```

```
    temp = 0
    St = np.zeros(0)
    St = np.append(St, s)
    while temp < 500:
        St = np.append(St, St[temp]
                       + r*St[temp]*(T/500.)
                       + exp(Y[temp])*St[temp]*(W1[temp+1] - W1[temp])
                       + (0.5 * exp(Y[temp])*exp(Y[temp])*St[temp]*((W1[temp+1] - W1[temp])**2 - (T/500.))))
        temp += 1

    # getting Smax from simulated St's, and ST terminal value
    Smax = max(St)
    ST = St[St.size - 1]

    # calculating payoff
    payoffsArray = np.append(payoffsArray, payoffLP(Smax, ST))

    num += 1
# calculating price
PriceLP = exp(-r*T)*np.mean(payoffsArray)
print("The price of the lookback put option is: ")
print(round(PriceLP,2))
```

Output:
The price of the lookback put option is:
12.43

Question 3) a) and b)

```
# importing the necessary packages
import numpy as np
from numpy import sqrt, exp, log
from scipy.stats import norm
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 = 100.
sigma = 0.2
r = 0.03
T = 1.
runs = 1000
K = 85
beta = 0
print("Beta is " + str(beta))

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

# defined function to get implied volatility
def ImplVol(MarketPrice, sigma_test):
    while sigma_test < 2.0:
        d1 = (log(s/K) + (r + (sigma_test**2)/2)*T) / (sigma_test * sqrt(T))
        d2 = d1 - sigma_test*sqrt(T)
        # black scholes formula price calculation
        PriceBS = s*norm.cdf(d1) - K*exp(-r*T)*norm.cdf(d2)
        # check if the two prices are close to each other
        if abs(PriceBS - MarketPrice) < 0.001:
            return sigma_test
        # else, do recursion of volatility calculation
        else:
            vega = s*sqrt(T)*norm.pdf(d1)
            sigma_test = sigma_test - (PriceBS - MarketPrice)/vega

# big loop to print out various K values
i = 0
while i < 9:

    # array holding payoffs for price calculation
    payoffsArray = np.zeros(0)

    num = 0
    while num < runs:
```

```
        # get W1 BM
        W1 = SimBMStep(T, 500.)

        # St simulation
        temp = 0
        St = np.zeros(0)
        St = np.append(St, s)
        while temp < 500:
            St = np.append(St, St[temp]
                           + r*St[temp]*(T/500.)
                           + sigma*(St[temp]**beta)*St[temp]*(W1[temp+1] - W1[temp]))
            temp += 1
        # terminal ST
        ST = St[St.size - 1]
        # calculating payoff
        payoffsArray = np.append(payoffsArray, payoffEC(ST, K))
        num += 1

    # calculating price
    PriceECmarket = exp(-r*T)*np.mean(payoffsArray)
    print("K = " + str(K) + ", beta = " + str(beta) + ", Euro Call Price = " + str(round(PriceECmarket,2)))

    # assume volatility is 1.0, in order to get correct implied volatility
    ImpliedVolatility = ImplVol(PriceECmarket, 1.0)
    print("Implied volatility = " + str(ImpliedVolatility))


    K += 5
    i += 1
```

Output:

K = 85, beta = 0, Euro Call Price = 18.78
Implied volatility = 0.184003503105
K = 90, beta = 0, Euro Call Price = 16.13
Implied volatility = 0.223356059083
K = 95, beta = 0, Euro Call Price = 12.75
Implied volatility = 0.216199749266
K = 100, beta = 0, Euro Call Price = 9.42
Implied volatility = 0.200207290968
K = 105, beta = 0, Euro Call Price = 7.28
Implied volatility = 0.203812957256
K = 110, beta = 0, Euro Call Price = 4.78
Implied volatility = 0.186659084449
K = 115, beta = 0, Euro Call Price = 4.31
Implied volatility = 0.212430179682
K = 120, beta = 0, Euro Call Price = 2.89
Implied volatility = 0.203695072607
K = 125, beta = 0, Euro Call Price = 1.68
Implied volatility = 0.189991788903

K = 85, beta = -0.3, Euro Call Price = 17.98

Implied volatility = 0.138511841585
K = 90, beta = -0.3, Euro Call Price = 13.27
Implied volatility = 0.113685727285
K = 95, beta = -0.3, Euro Call Price = 8.25
Implied volatility = 0.0716150836257
K = 100, beta = -0.3, Euro Call Price = 3.77
Implied volatility = 0.0494287139351
K = 105, beta = -0.3, Euro Call Price = 1.28
Implied volatility = 0.0520328059972
K = 110, beta = -0.3, Euro Call Price = 0.25
Implied volatility = None
K = 115, beta = -0.3, Euro Call Price = 0.02
Implied volatility = 0.0472742516509
K = 120, beta = -0.3, Euro Call Price = 0.0
Implied volatility = 0.0503292228721
K = 125, beta = -0.3, Euro Call Price = 0.0
Implied volatility = 0.056491997428

K = 85, beta = -0.5, Euro Call Price = 18.11
Implied volatility = 0.147353340733
K = 90, beta = -0.5, Euro Call Price = 13.12
Implied volatility = 0.105224876388
K = 95, beta = -0.5, Euro Call Price = 8.06
Implied volatility = 0.0614743128677
K = 100, beta = -0.5, Euro Call Price = 3.13
Implied volatility = 0.0269013914866
K = 105, beta = -0.5, Euro Call Price = 0.17
Implied volatility = 0.0191667371791
K = 110, beta = -0.5, Euro Call Price = 0.0
Implied volatility = 0.000293248345683
K = 115, beta = -0.5, Euro Call Price = 0.0
Implied volatility = 0.0333351987452
K = 120, beta = -0.5, Euro Call Price = 0.0
Implied volatility = 0.0470901696856
K = 125, beta = -0.5, Euro Call Price = 0.0
Implied volatility = 0.056491997428

K = 85, beta = -0.7, Euro Call Price = 18.02
Implied volatility = 0.14143418072
K = 90, beta = -0.7, Euro Call Price = 13.01
Implied volatility = 0.0979190199733
K = 95, beta = -0.7, Euro Call Price = 8.08
Implied volatility = 0.0622625724996
K = 100, beta = -0.7, Euro Call Price = 3.03
Implied volatility = 0.021075922155
K = 105, beta = -0.7, Euro Call Price = 0.0

Implied volatility = 0.00842065817146
K = 110, beta = -0.7, Euro Call Price = 0.0
Implied volatility = 0.000274416855772
K = 115, beta = -0.7, Euro Call Price = 0.0
Implied volatility = 0.0333351987452
K = 120, beta = -0.7, Euro Call Price = 0.0
Implied volatility = 0.0470901696856
K = 125, beta = -0.7, Euro Call Price = 0.0
Implied volatility = 0.056491997428

K = 85, beta = -1.0, Euro Call Price = 18.06
Implied volatility = 0.143754335366
K = 90, beta = -1.0, Euro Call Price = 13.04
Implied volatility = 0.100385165424
K = 95, beta = -1.0, Euro Call Price = 8.05
Implied volatility = 0.0609180707467
K = 100, beta = -1.0, Euro Call Price = 3.05
Implied volatility = 0.0223719089738
K = 105, beta = -1.0, Euro Call Price = 0.0
Implied volatility = 0.00670529407276
K = 110, beta = -1.0, Euro Call Price = 0.0
Implied volatility = 0.000274416855772
K = 115, beta = -1.0, Euro Call Price = 0.0
Implied volatility = 0.0333351987452
K = 120, beta = -1.0, Euro Call Price = 0.0
Implied volatility = 0.0470901696856
K = 125, beta = -1.0, Euro Call Price = 0.0
Implied volatility = 0.056491997428

Role of Beta on the volatility smile:
As Beta decreases into more and more negative, the volatility smile decreases.
We can see there are lower implied volatility values for lower beta values.

Question 4)

```
# importing the necessary packages
import numpy as np
from numpy import sqrt, exp, log
from scipy.stats import norm
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 = 100.
y = 0.08
lam = 3
kappa = 0.1
rho = -0.8
xi = 0.1
r = 0.03

T = 1.0
runs = 1000
K = 85

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

# defined function to get implied volatility
def ImplVol(MarketPrice, sigma_test):
    while sigma_test < 2.0:
        d1 = (log(s/K) + (r + (sigma_test**2)/2)*T) / (sigma_test * sqrt(T))
        d2 = d1 - sigma_test*sqrt(T)
        # black scholes formula price calculation
        PriceBS = s*norm.cdf(d1) - K*exp(-r*T)*norm.cdf(d2)
        # check if the two prices are close to each other
        if abs(PriceBS - MarketPrice) < 0.001:
            return sigma_test
        # else, do recursion of volatility calculation
        else:
            vega = s*sqrt(T)*norm.pdf(d1)
            sigma_test = sigma_test - (PriceBS - MarketPrice)/vega

# big loop to get difference price values for different K values
i = 0
while i < 9:
    # 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 = rho*W1 + sqrt(1 - rho**2)*Wtilda

        # simulating Yt values
        temp = 0
        Y = np.zeros(0)
        # getting initial Y_0, which is little y = -1
        Y = np.append(Y, y)
        # loop for Yt simulation
        # NOTE, Yt follows CIR process form
        while temp < 500:
            Y = np.append(Y, max(Y[temp]
                                 + lam*(kappa - Y[temp])*(T/500.)
                                 + xi*sqrt(Y[temp])*(W2[temp+1] - W2[temp]), 0))
            temp += 1

        # St simulation (MILSTEIN SCHEME)
        temp = 0
        St = np.zeros(0)
        St = np.append(St, s)
        while temp < 500:
            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
        payoffsArray = np.append(payoffsArray, payoffEC(ST, K))

        num += 1
    # calculating price
    PriceECmarket = exp(-r*T)*np.mean(payoffsArray)
           print("K = " + str(K) + ", European Call price = " + str(round(PriceECmarket,2)))
           # assume volatility is 1.0, in order to get correct implied volatility
           ImpliedVolatility = ImplVol(PriceECmarket, 1.0)
           print("Implied volatility = " + str(ImpliedVolatility))

           K += 5
           i += 1
```

Output:

K = 85, European Call price = 22.08

Implied volatility = 0.310180904529

K = 90, European Call price = 18.9

Implied volatility = 0.30869756875

K = 95, European Call price = 17.48

Implied volatility = 0.346619156161

K = 100, European Call price = 13.78

Implied volatility = 0.312728538256
K = 105, European Call price = 11.65
Implied volatility = 0.313429144511
K = 110, European Call price = 8.65
Implied volatility = 0.285122524159
K = 115, European Call price = 7.23
Implied volatility = 0.289347967669
K = 120, European Call price = 6.28
Implied volatility = 0.299679538249
K = 125, European Call price = 5.18
Implied volatility = 0.300697706649

When changing rho, the volatility smile seems to get flatter if there's a large positive/negative correlation.
When changing xi, if xi is large then the volatility smile decreases gradually.
But if xi is small then the volatility isn't really a "smile," and it remains mostly around 0.3.

Question 5) c)

```
import numpy as np
from numpy import sqrt, exp, log
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
y = 0.08
lam = 3.
kappa = 0.1
xi = 0.25

T = 1
N = 100.

t = np.arange(0, 1.01, 0.01)

i = 0
while i < 10:
    W1 = SimBMStep(T, N)

    Y = np.zeros_like(W1)
    X = np.zeros_like(W1)
    Y[0] = y
    X[0] = sqrt(y)
    temp = 0
    while temp < X.size-1:
        X[temp + 1] = (xi*(W1[temp+1] - W1[temp])
                        + sqrt(xi**2 * (W1[temp+1] - W1[temp])**2
                            + 4*(1+lam*(T/N))*(X[temp]**2 + (T/N)*(lam*kappa - (xi**2/2)))))/(2*(1+lam*(T/N)))
        Y[temp + 1] = X[temp+1]**2
        temp += 1
    plt.plot(t, Y)
    i += 1

plt.title("Simulated CIR Yt paths")
plt.xlabel("Time t")
plt.ylabel("Process value")
plt.show()
```

Figure:
(I assumed initial values for y, xi, kappa, and lambda are same as previous question)