MA 573
HW 4
Alex Shoop

Question 1)

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

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

plt.clf()

# variables given
S_0 = 100.
B0 = 1
mu = 0.05
sigma = 0.35
r = 0.05

# number of simulation runs/sample size for Monte Carlo
runs = 5000

# setting up sample size n x-axis variable for plotting
n = np.arange(0.0, runs, 1.0)

# European Call, using Black Scholes formula
K = 120.
T = 2

d1 = (log(S_0/K) + (r + (sigma**2)/2)*T) / (sigma * sqrt(T))
d2 = d1 - sigma*sqrt(T)

# black scholes formula price calculation
Price1 = S_0*norm.cdf(d1) - K*exp(-r*T)*norm.cdf(d2)

print("European Call option price (using Black Scholes formula) is:")
print(round(Price1,2))

plt.axhline(y = Price1, color = 'black', linestyle = '-', label = 'BS formula')
plt.legend()

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

# defining the stock generation function
def generateStock(S_0, r, sigma, T):
    return S_0 * exp((r - (sigma**2)/2)*T + sigma * sqrt(T) * np.random.standard_normal())

# defining payoff function, for this case it's regular European call
def payoff1(S, K):
    return max(S - K, 0)

# defining monte carlo integration formula (direct version)
def MonteCarlo_Direct(runs):
    # initialize array that will have payoffs of option
    payoffs = np.zeros(0)

    # looping through
    for i in xrange(runs):
```

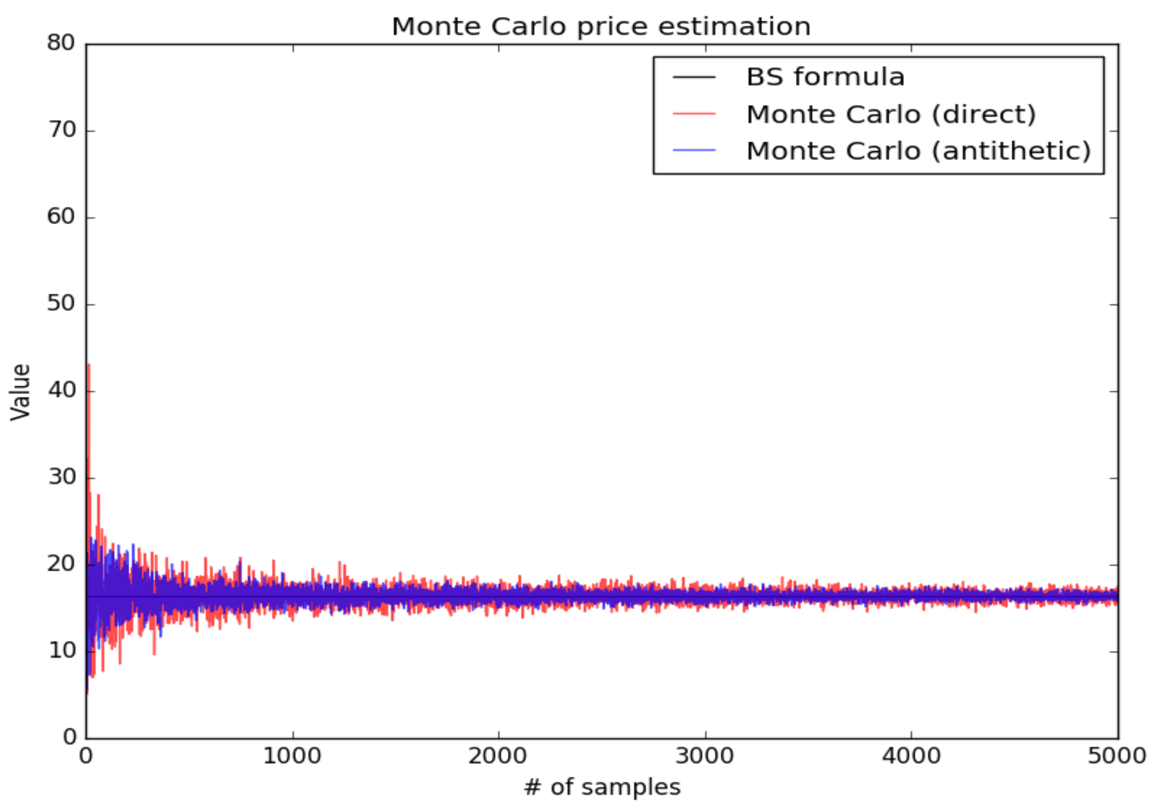
```
        # generate future stock
        S_T = generateStock(S_0, r, sigma, T)

        # append to the payoffs list whatever the payoff is
        payoffs = np.append(payoffs,
                            exp(-r*T)*payoff1(S_T, K))

    return sum(payoffs)/runs

# array holding Monte Carlo direct approximation values
# this is also initial point, n = 0
MC1 = np.zeros(0)

# counter for loop
num = 1

# loop for rest of Monte Carlo approximation values, until sample size 5000
while num <= runs:
    MC1 = np.append(MC1, MonteCarlo_Direct(num))
    num += 1

plt.plot(n, MC1, color = 'red', alpha = 0.7, label = 'Monte Carlo (direct)')
plt.legend()

print("European Call option price (using direct Monte Carlo) is:")
print(round(MC1[MC1.size - 1],2))

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

# defining monte carlo integration formula WITH antithetic method
def MonteCarlo_Anti(runs):
    # initialize array that will have payoffs of option
    payoffs1 = np.zeros(0)
    payoffs2 = np.zeros(0)

    # looping through
    for i in xrange(runs):
        # generate future stock
        S_T = generateStock(S_0, r, sigma, T)
        S_T_neg = -(generateStock(S_0, r, sigma, T))

        # append to the payoffs list whatever the payoff is
        payoffs1 = np.append(payoffs1, exp(-r*T)*payoff1(S_T, K))
              payoffs2 = np.append(payoffs2, exp(-r*T)*payoff1(np.abs(S_T_neg), K))

          sum_payoffs = (payoffs1 + payoffs2) / 2
          return np.mean(sum_payoffs)

      # array holding Monte Carlo antithetic approximation
      MC2 = np.zeros(0)

      # counter for loop
      num = 1

      # loop for rest of Monte Carlo approximation values, until sample size 5000
      while num <= runs:
          MC2 = np.append(MC2, MonteCarlo_Anti(num))
          num += 1

      plt.plot(n, MC2, color = 'blue', alpha = 0.7, label = 'Monte Carlo (antithetic)'
)
      plt.legend()
```

```

        print("European Call option price (using Monte Carlo & Antithetic method) is:")
        print(round(MC2[MC2.size - 1],2))



        plt.title("Monte Carlo price estimation")
        plt.xlabel("# of samples")
        plt.ylabel("Value")

        plt.show()
```

Output:

European Call option price (using Black Scholes formula) is:
16.37
European Call option price (using direct Monte Carlo) is:
17.48
European Call option price (using Monte Carlo & Antithetic method) is:
16.24

Figure:



Question 2)

```
# importing the necessary packages
import numpy as np
from numpy import sqrt, exp, log
import matplotlib.pyplot as plt

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

plt.clf()

# variables given
S_0 = 100.
B0 = 1
mu = 0.1
sigma = 0.2
r = 0.02

# number of simulation runs/sample size for Monte Carlo
runs = 5000

# initalize our Z array used for S_T calculation
Z1 = np.random.standard_normal(runs)

###########################################
# part a)

plt.subplot(2,3,1)

T = 0.5
K = 95.
N = 500
#dt = T/runs

# calculating S_T array
S_T = S_0 * exp((r - (sigma**2)/2)*T + sigma * sqrt(T) * Z1)

# calculation of European put payoffs
temp = K - S_T
for x in np.nditer(temp, op_flags = ['readwrite']):
    x[...] = max(x, 0) # in order to make negative values into 0
Euro_put_payoffs1 = temp

# calculation of correlation
Correlation_a = np.corrcoef(S_T, Euro_put_payoffs1)
print("The correlation between underlying stock and a European put option is: ")
print(Correlation_a[0,1])

# plotting
plt.scatter(S_T, Euro_put_payoffs1)
plt.title("Underlying & Euro Put")
plt.xlabel("Underlying Stock")
plt.ylabel("Payoffs")

###########################################
# part b)

plt.subplot(2,3,2)

# use same T and K

# calculating S_T array
```

```python
S_T = S_0 * exp((r - (sigma**2)/2)*T + sigma * sqrt(T) * Z1)

# calculation of European call payoffs
temp = S_T - K
for x in np.nditer(temp, op_flags = ['readwrite']):
    x[...] = max(x, 0) # in order to make negative values into 0
Euro_call_payoffs = temp

# calculation of correlation
Correlation_b = np.corrcoef(S_T, Euro_call_payoffs)
print("The correlation between underlying stock and a European call option is: ")
print(Correlation_b[0,1])

# plotting
plt.scatter(S_T, Euro_call_payoffs)
plt.title("Underlying & Euro Call")
plt.xlabel("Underlying Stock")
plt.ylabel("Payoffs")

#############################################
# part c)

plt.subplot(2,3,3)

# use same T and K

# initialize array to hold average values
Asian_average_arith1 = np.zeros(0)

# array to hold S(T) terminal values, for plotting purposes
ST_array1 = np.zeros(0)

# loop for getting St arithmetic average values. Always use (1/N) for
# average calculation

# USING 180 DAYS AS HALF A YEAR (T = 0.5)

num = 0
while num < runs:
    BM_sum = 0
    S_t = np.zeros(0)

    # consider cumulative sum if time permits

    # calculating St array
    i = 0
    while i < 180:
        # have to make new Brownian Motion for each increment step
        Z2 = np.random.standard_normal()
        BM_increment = sqrt(1/180.) * Z2
        BM_sum += BM_increment

        S_t = np.append(S_t, S_0 * exp((r - (sigma**2)/2)*(i+1)*(1/180.) + sigma * BM_sum))
        i += 1

    # keep the S_T terminal value for plotting
    ST_array1 = np.append(ST_array1, S_t[S_t.size - 1])

    # calculate average value
    Asian_average_arith1 = np.append(Asian_average_arith1, sum(S_t)/S_t.size)
    num += 1
```

```python

    # Asian arithmatic put payoffs
    temp = K - Asian_average_arith1
    for x in np.nditer(temp, op_flags = ['readwrite']):
        x[...] = max(x, 0) # in order to make negative values into 0
    Asian_put_payoffs_arith1 = temp


    # calculation of correlation
    Correlation_c = np.corrcoef(ST_array1, Asian_put_payoffs_arith1)
    print("The correlation between underlying stock and arithmetic average Asian put is:  ")
    print(Correlation_c[0,1])

    # plotting
    plt.scatter(ST_array1, Asian_put_payoffs_arith1)
    plt.title("Underlying & arithmetic average Asian Put")
    plt.xlabel("Underlying Stock")
    plt.ylabel("Payoffs")


    ###########################################
    # part d)

    plt.subplot(2,3,4)

    # use same T and K

    # initialize array to hold average values
    Asian_average_arith2 = np.zeros(0)

    # array to hold S(T) terminal values, for Euro put payoff calculations
    ST_array2 = np.zeros(0)

    # loop for getting St arithmetic average values.
    num = 0
    while num < runs:
        BM_sum = 0
        S_t = np.zeros(0)

        # consider cumulative sum if time permits

        # calculating St array
        i = 0
        while i < 180:
            # have to make new Brownian Motion for each increment step
            Z2 = np.random.standard_normal()
            BM_increment = sqrt(1/180.) * Z2
            BM_sum += BM_increment

            S_t = np.append(S_t, S_0 * exp((r - (sigma**2)/2)*(i+1)*(1/180.) + sigma * BM_sum))
            i += 1

        # keep the S_T terminal value for Euro put payoff calculation
        ST_array2 = np.append(ST_array2, S_t[S_t.size - 1])

        # calculate average value
        Asian_average_arith2 = np.append(Asian_average_arith2, sum(S_t)/S_t.size)
        num += 1

    # Asian arithmatic put payoffs
```

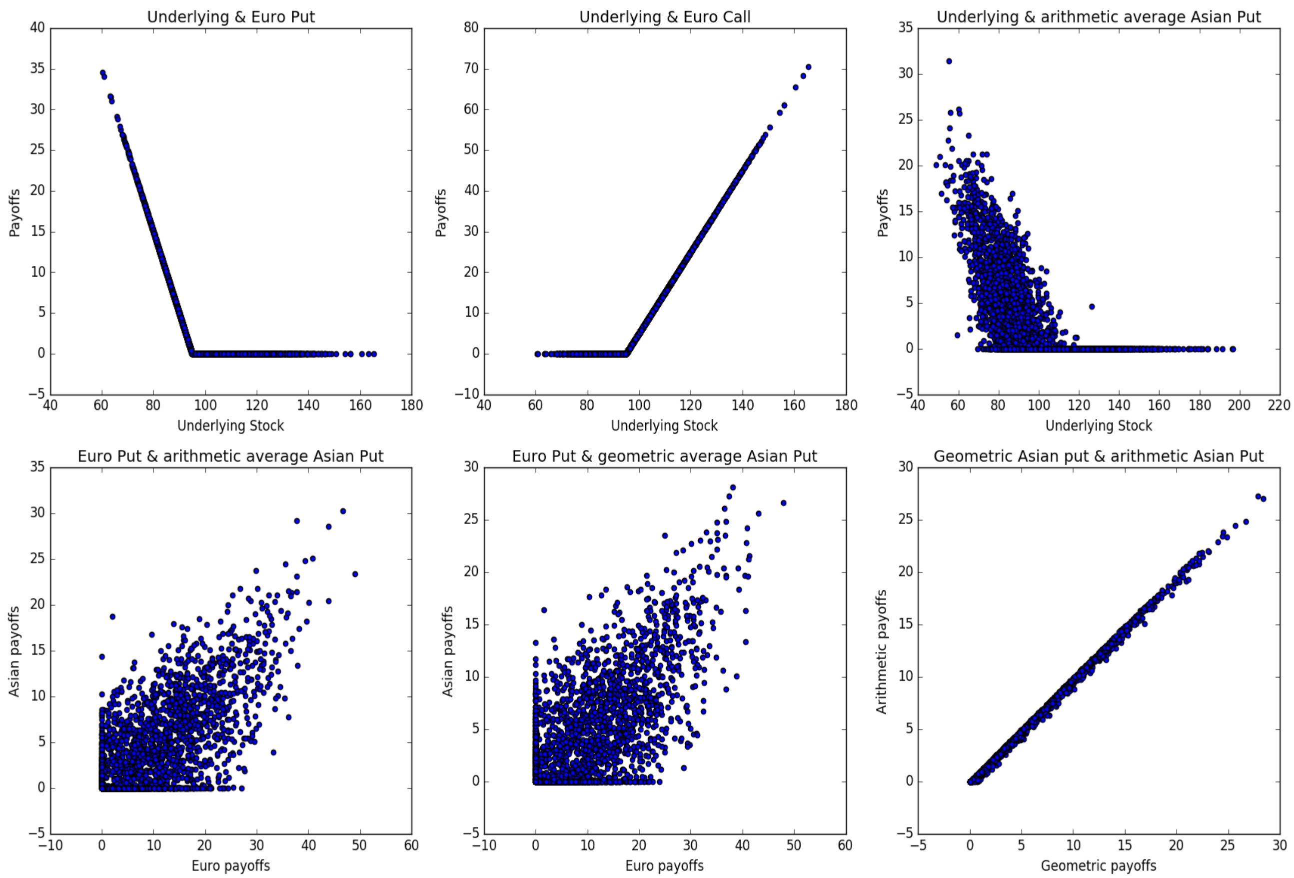
```
        temp = K - Asian_average_arith2
        for x in np.nditer(temp, op_flags = ['readwrite']):
            x[...] = max(x, 0) # in order to make negative values into 0
        Asian_put_payoffs_arith2 = temp

        # calculation of European put payoffs
        temp = K - ST_array2
        for x in np.nditer(temp, op_flags = ['readwrite']):
            x[...] = max(x, 0) # in order to make negative values into 0
        Euro_put_payoffs2 = temp

        # calculation of correlation
        Correlation_d = np.corrcoef(Euro_put_payoffs2, Asian_put_payoffs_arith2)
        print("The correlation between European put and arithmetic Asian put is: ")
        print(Correlation_d[0,1])

        # plotting
        plt.scatter(Euro_put_payoffs2, Asian_put_payoffs_arith2)
        plt.title("Euro Put & arithmetic average Asian Put")
        plt.xlabel("Euro payoffs")
        plt.ylabel("Asian payoffs")

        ###########################################
        # part e)

        plt.subplot(2,3,5)

        # use same T and K

        # initialize array to hold average values
        Asian_average_geo1 = np.zeros(0)

        # array to hold S(T) terminal values, for Euro put payoff calculations
        ST_array3 = np.zeros(0)

        # loop for getting St arithmetic average values.
        num = 0
        while num < runs:
            BM_sum = 0
            S_t = np.zeros(0)

            # consider cumulative sum if time permits

            # calculating St array
            i = 0
            while i < 180:
                # have to make new Brownian Motion for each increment step
                Z2 = np.random.standard_normal()
                BM_increment = sqrt(1/180.) * Z2
                BM_sum += BM_increment

                S_t = np.append(S_t, S_0 * exp((r - (sigma**2)/2)*(i+1)*(1/180.) + sigma
 * BM_sum))
                i += 1

            # keep the S_T terminal value for Euro put payoff calculation
            ST_array3 = np.append(ST_array3, S_t[S_t.size - 1])

            # calculate average value
            # slight adjustment of product and nth root calculation
            # first take nth root, then take product
```

```
        Asian_average_geo1 = np.append(Asian_average_geo1, np.product(S_t**(1/180.)))
        num += 1

    # Asian geometric put payoffs
    temp = K - Asian_average_geo1
    for x in np.nditer(temp, op_flags = ['readwrite']):
        x[...] = max(x, 0) # in order to make negative values into 0
    Asian_put_payoffs_geo1 = temp

    # calculation of European put payoffs
    temp = K - ST_array3
    for x in np.nditer(temp, op_flags = ['readwrite']):
        x[...] = max(x, 0) # in order to make negative values into 0
    Euro_put_payoffs3 = temp

    # calculation of correlation
    Correlation_e = np.corrcoef(Euro_put_payoffs3, Asian_put_payoffs_geo1)
    print("The correlation between European put and geometric Asian put is: ")
    print(Correlation_e[0,1])

    # plotting
    plt.scatter(Euro_put_payoffs3, Asian_put_payoffs_geo1)
    plt.title("Euro Put & geometric average Asian Put")
    plt.xlabel("Euro payoffs")
    plt.ylabel("Asian payoffs")

    ###########################################
    # part f)

    plt.subplot(2,3,6)

    # use same T and K

    # initialize array to hold average values
    Asian_average_geo2 = np.zeros(0)
    Asian_average_arith3 = np.zeros(0)

    # loop for getting St arithmetic average values.
    num = 0
    while num < runs:
        BM_sum = 0
        S_t = np.zeros(0)

        # consider cumulative sum if time permits

        # calculating St array
        i = 0
        while i < 180:
            # have to make new Brownian Motion for each increment step
            Z2 = np.random.standard_normal()
            BM_increment = sqrt(1/180.) * Z2
            BM_sum += BM_increment

            S_t = np.append(S_t, S_0 * exp((r - (sigma**2)/2)*(i+1)*(1/180.) + sigma * BM_sum))
            i += 1

        # keep the S_T terminal value for Euro put payoff calculation
        #ST_array4 = np.append(ST_array2, S_t[S_t.size - 1])

        # calculate average value
```

```
            Asian_average_geo2 = np.append(Asian_average_geo2,
                                           np.product(S_t**(1/180.)))
            Asian_average_arith3 = np.append(Asian_average_arith3, sum(S_t)/S_t.size)
            num += 1

        # Asian geometric put payoffs
        temp = K - Asian_average_geo2
        for x in np.nditer(temp, op_flags = ['readwrite']):
            x[...] = max(x, 0) # in order to make negative values into 0
        Asian_put_payoffs_geo2 = temp

        # Asian arithmatic put payoffs
        temp = K - Asian_average_arith3
        for x in np.nditer(temp, op_flags = ['readwrite']):
            x[...] = max(x, 0) # in order to make negative values into 0
        Asian_put_payoffs_arith3 = temp

        # calculation of correlation
        Correlation_f = np.corrcoef(Asian_put_payoffs_geo2, Asian_put_payoffs_arith3)
        print("The correlation between European put and geometric Asian put is: ")
        print(Correlation_f[0,1])

        # plotting
        plt.scatter(Asian_put_payoffs_geo2, Asian_put_payoffs_arith3)
        plt.title("Geometric Asian put & arithmetic Asian Put")
        plt.xlabel("Geometric payoffs")
        plt.ylabel("Arithmetic payoffs")

        plt.show()
```

Output:
The correlation between underlying stock and a European put option is:
-0.726837599772
The correlation between underlying stock and a European call option is:
0.941542516821
The correlation between underlying stock and arithmetic average Asian put is:
-0.601785081306
The correlation between European put and arithmetic Asian put is:
0.78219059056
The correlation between European put and geometric Asian put is:
0.785733209395
The correlation between European put and geometric Asian put is:
0.999505638744

Figure:


Underlying & Euro Put
Payoffs
Underlying Stock
Underlying & Euro Call
Payoffs
Underlying Stock
Underlying & arithmetic average Asian Put
Payoffs
Underlying Stock
Euro Put & arithmetic average Asian Put
Asian payoffs
Euro payoffs
Euro Put & geometric average Asian Put
Asian payoffs
Euro payoffs
Geometric Asian put & arithmetic Asian Put
Arithmetic payoffs
Geometric payoffs

Question 3)

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

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

# variables given
S_0 = 100.
B0 = 1
mu = 0.3
sigma = 0.3
r = 0.03

# Asian Call
K = 120.
T = 5

# number of simulation runs/sample size for Monte Carlo
runs = 10000

# number of average time for Asian payoff calculation
N = 1260.

dt = T/N

#####################################
# part c)

# Using the Geometric Asian Average as the Control Variate

# Need actual value of Geometric Asian call option
sigma_geo = sigma * sqrt(1/3.)
r_geo = 0.5 * (r - (1/6.) * sigma**2)
d1 = (log(S_0/K) + 0.5 * (r + (1/6.)*sigma**2) * T) / (sigma_geo * sqrt(T))
d2 = d1 - (sigma_geo * sqrt(T))
Asian_geo_price = exp(-(r*T)) * (S_0*exp(r_geo*T)*norm.cdf(d1) - K*norm.cdf(d2))

# initialize array to hold average values
Asian_avg_geo = np.zeros(0)
Asian_avg_arith = np.zeros(0)

# loop for getting St arithmetic average values.
num = 0
while num < runs:
    BM_sum = 0
    S_t = np.zeros(0)

    # calculating St array
    i = 0
    while i < int(N):
        # have to make new Brownian Motion for each increment step
        Z = np.random.standard_normal()
        BM_increment = sqrt(dt) * Z
        BM_sum += BM_increment

        S_t = np.append(S_t, S_0 * exp((r - (sigma**2)/2)*(i+1)*(dt) + sigma * BM_sum))

        i += 1
```

```
    # calculate average value
    Asian_avg_geo = np.append(Asian_avg_geo, np.product(S_t**(1/N)))
    Asian_avg_arith = np.append(Asian_avg_arith, sum(S_t)/S_t.size)
    num += 1

# Asian geometric call payoffs
temp = Asian_avg_geo - K
for x in np.nditer(temp, op_flags = ['readwrite']):
    x[...] = max(x, 0) # in order to make negative values into 0
Asian_call_payoffs_geo = temp

# Asian arithmatic call payoffs
temp = Asian_avg_arith - K
for x in np.nditer(temp, op_flags = ['readwrite']):
    x[...] = max(x, 0) # in order to make negative values into 0
Asian_call_payoffs_arith = temp

# calculate the optimal b value for the equation sampling
Cov_test = np.cov(Asian_call_payoffs_geo, Asian_call_payoffs_arith)
b_optimal = Cov_test[0,1] / Cov_test[1,1]

# new sampling, using the control variate
Asian_sampling = Asian_call_payoffs_arith - b_optimal * (Asian_call_payoffs_geo - Asian_geo_price)

Asian_call_price = exp(-r*T) * np.mean(Asian_sampling)
print("The estimated price of arithmetic average Asian Call Option (with Control Variate) is:")
print(round(Asian_call_price,2))

######################################
# part d)

Y_asian_call_control = np.zeros(0)
Y_asian_call_NoControl = np.zeros(0)

# getting the respective arithmetic Asian call prices (WITH CONTROL VARIATE)
# Asian_sampling is from part c)
for i in range(runs):
    Y_asian_call_control = np.append(Y_asian_call_control,
                                       exp(-r*T) * (sum(Asian_sampling[:(i+1)]) / (Asian_sampling[:(i+1)].size)))

    # getting the respective arithmetic Asian call prices based on sample size
    for j in range(runs):
        Y_asian_call_NoControl = np.append(Y_asian_call_NoControl,
                                            exp(-r*T) *
                                             (sum(Asian_call_payoffs_arith[:(j+1)])
                                             / (Asian_call_payoffs_arith[:(j+1)].size)))


    plt.plot(np.arange(runs), Y_asian_call_control, label = "Monte Carlo w/ Control Variate")
    plt.plot(np.arange(runs), Y_asian_call_NoControl, label = "Direct Monte Carlo")

    plt.title("Monte Carlo Calculation of Arithmetic Asian Call Price")
    plt.xlabel("Sample size")
    plt.ylabel("Price")
```

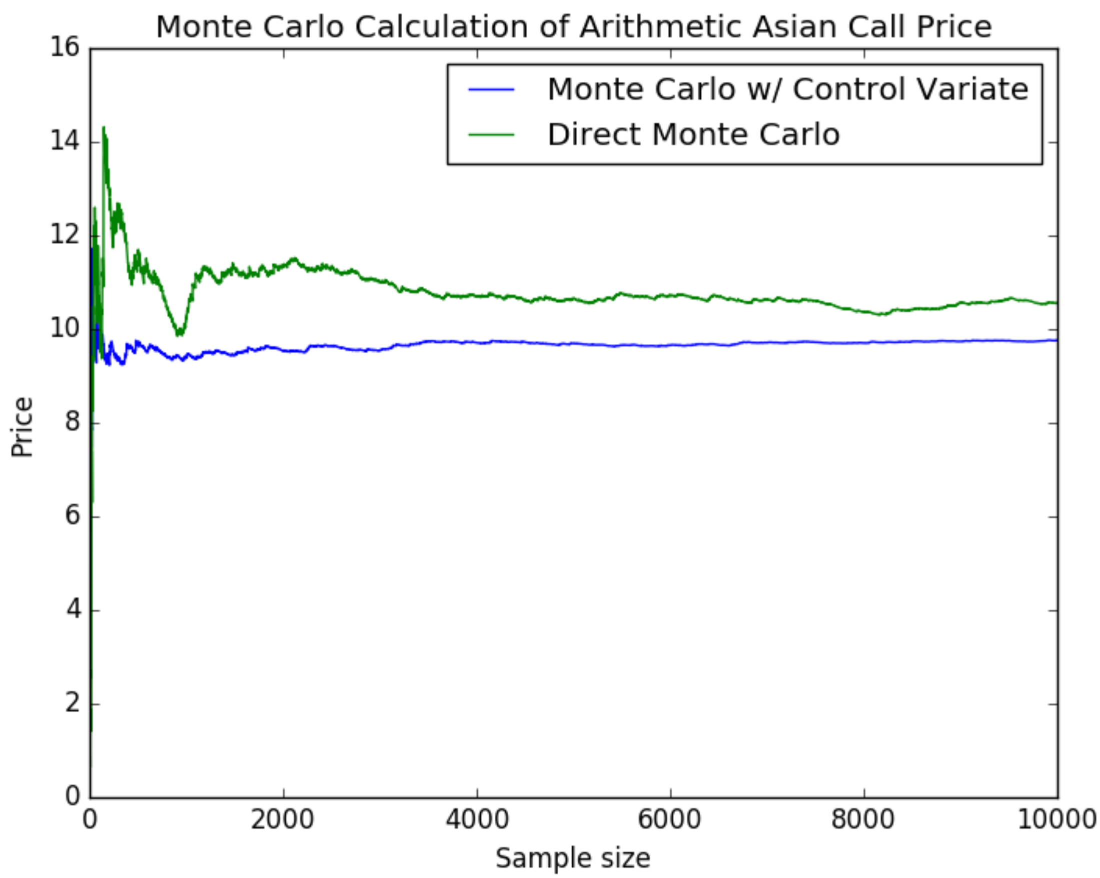
```python

        plt.legend(loc = 1)
        plt.show()

        #####################################
        # part d)
        print("---------------------------------------------------------------------
")
        print("Actual price of Asian geometric call is: ")
        print(round(Asian_geo_price,2))

        print("The estimated price of arithmetic average Asian Call Option (with Control
 Variate) is:")
        print(round(Asian_call_price,2))
        print("The estimated price of arithmetic average Asian Call (using only Direct M
onte Carlo) is: ")
        print(round(Y_asian_call_NoControl[runs - 1], 2))

        # calculations for Euro Call calculation

        Z1 = np.random.standard_normal(runs)
        # calculating S_T array
        S_T = S_0 * exp((r - (sigma**2)/2)*T + sigma * sqrt(T) * Z1)
        # calculation of European call payoffs
        temp = S_T - K
        for x in np.nditer(temp, op_flags = ['readwrite']):
            x[...] = max(x, 0) # in order to make negative values into 0
        Euro_call_payoffs = temp
        Euro_call_price = exp(-r*T) * np.mean(Euro_call_payoffs)

        print("Price of European call option is: ")
        print(round(Euro_call_price,2))
```

Output:

Actual price of Asian geometric call is:
8.83
The estimated price of arithmetic average Asian Call Option (with Control Variate) is:
9.72
The estimated price of arithmetic average Asian Call (using only Direct Monte Carlo) is:
10.71
Price of European call option is:
25.38

Therefore, we can note that the European call option price is higher than both the Asian arithmetic average call and the Asian geometric average call options.

Figure:


Monte Carlo Calculation of Arithmetic Asian Call Price
Monte Carlo w/ Control Variate
Direct Monte Carlo
Price
Sample size
0
2
4
6
8
10
12
14
16
0
2000
4000
6000
8000
10000

Question 4)

```python
# importing the necessary packages
import numpy as np
from numpy import exp, log, sqrt
import matplotlib.pyplot as plt
import math

################################################################################
#part a)

plt.clf()

# variables given
S_0 = 100.

mu = 0.1
sigma = 0.2

T = 10
N = 10000.   #10 years, and 1000 steps per year
dt = T/N

lambda_var = 2.

#simulation runs
runs = 1000.

# our large Y array which will have 1000 samples
Y_stratified = np.zeros(0)

# inverse CDF of exponential
def InverseExp(x):
    y = - 0.5 * (log(1 - x))
    return y

# Need to get stratified intervals
a_stratPoints = np.zeros(0)
for i in range(10):
    val = (i+1)/float(10)
    a_stratPoints = np.append(a_stratPoints, InverseExp(val))

U1 = np.random.random_sample(100)
Y_stratified = np.append(Y_stratified, 0 + (a_stratPoints[0] - 0)*U1 )

# loop for Yi values
for j in range(9):
    U2 = np.random.random_sample(100)

    Y_stratified = np.append(Y_stratified, a_stratPoints[j] + (a_stratPoints[j+1] - a_stratPoints[j])*U2)


# plot and info for regular density of exponential, and non-stratified sampling
x = np.random.random_sample(1000)
y = InverseExp(x)
z = np.arange(0, 3, 0.1)
w = lambda_var * np.exp(-z * lambda_var)*100

plt.plot(z, w, color = 'red', label = 'Exp density')
plt.hist(y, range = (0,3), bins = 30, color = 'blue', label = 'Non-stratified sampling')
```

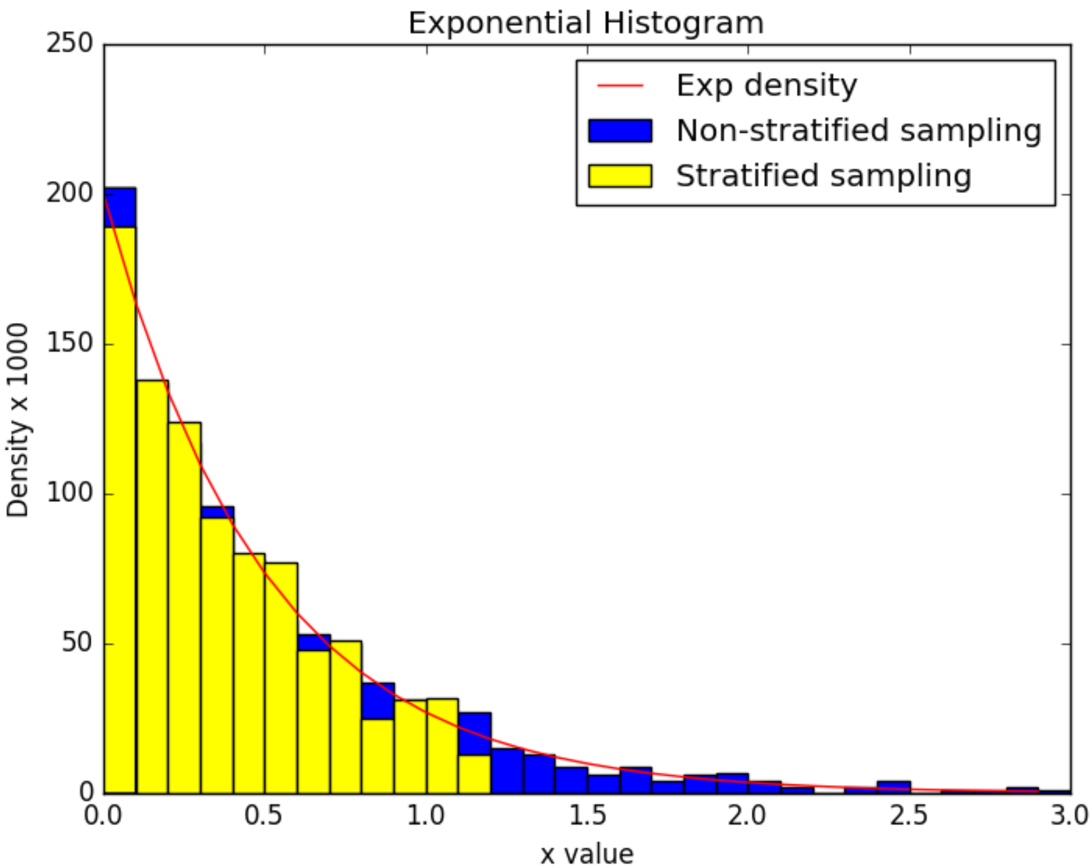
```
plt.hist(Y_stratified, range = (0,3), bins = 30, color = 'yellow', label = 'Stratified 
sampling')

plt.title ("Exponential Histogram")
plt.xlabel ("x value")
plt.ylabel ("Density x 1000")

plt.legend()
plt.show()

###############################################################################
#part b)

# array to keep ST terminal stock values, for expectation calculation
ST_array = np.zeros(0)

num = 0
while num < runs:
    S_t = np.zeros(0)
    BM_sum = 0

    # calculating St array
    k = 0
    tau = Y_stratified[num]

    # if no default occured, then set time to final terminal time (aka. T = 10)
    if tau > 1.0:
        tau = 1.0

    # St simulation loop, until time tau.
    while k < tau:
        # have to make new Brownian Motion for each increment step
        Z = np.random.standard_normal()
        BM_increment = sqrt(dt) * Z
        BM_sum += BM_increment

        S_t = np.append(S_t, S_0 * exp((mu - 0.5*(sigma**2)*(k+1)*(dt) + sigma * BM_sum
)))
        k += 0.001    #1000 steps each year

    ST_array = np.append(ST_array, S_t[S_t.size - 1])
    num += 1

    PriceAtDefault = sum(ST_array) / ST_array.size

    print("The expected price at default is: ")
    print(PriceAtDefault)
```

Output:

The expected price at default is:

111.740436214

Figure:


Exponential Histogram
Exp density
Non-stratified sampling
Stratified sampling
Density x 1000
250
200
150
100
50
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
x value