

Question 1)

```
1. # importing the necessary packages
2. import numpy as np
3. from numpy import sqrt, exp, log
4. from scipy.stats import norm
5. import matplotlib.pyplot as plt
6.
7. #####
8.
9. plt.clf()
10.
11. # variables given
12. S_0 = 100.
13. B0 = 1
14. mu = 0.05
15. sigma = 0.35
16. r = 0.05
17.
18. # number of simulation runs/sample size for Monte Carlo
19. runs = 5000
20.
21. # setting up sample size n x-axis variable for plotting
22. n = np.arange(0.0, runs, 1.0)
23.
24. # European Call, using Black Scholes formula
25. K = 120.
26. T = 2
27.
28. d1 = (log(S_0/K) + (r + (sigma**2)/2)*T) / (sigma * sqrt(T))
29. d2 = d1 - sigma*sqrt(T)
30.
31. # black scholes formula price calculation
32. Price1 = S_0*norm.cdf(d1) - K*exp(-r*T)*norm.cdf(d2)
33.
34. print("European Call option price (using Black Scholes formula) is:")
35. print(round(Price1,2))
36.
37. plt.axhline(y = Price1, color = 'black', linestyle = '-', label = 'BS formula')
38. plt.legend()
39.
40. #####
41.
42. # defining the stock generation function
43. def generateStock(S_0, r, sigma, T):
44.     return S_0 * exp((r - (sigma**2)/2)*T + sigma * sqrt(T) * np.random.standard_normal
45.         ())
46.
47. # defining payoff function, for this case it's regular European call
48. def payoff1(S, K):
49.     return max(S - K, 0)
50.
51. # defining monte carlo integration formula (direct version)
52. def MonteCarlo_Direct(runs):
53.     # initialize array that will have payoffs of option
54.     payoffs = np.zeros(0)
55.     # looping through
56.     for i in xrange(runs):
```

```
57.         # generate future stock
58.         S_T = generateStock(S_0, r, sigma, T)
59.
60.         # append to the payoffs list whatever the payoff is
61.         payoffs = np.append(payoffs,
62.                             exp(-r*T)*payoff1(S_T, K))
63.
64.     return sum(payoffs)/runs
65.
66. # array holding Monte Carlo direct approximation values
67. # this is also initial point, n = 0
68. MC1 = np.zeros(0)
69.
70. # counter for loop
71. num = 1
72.
73. # loop for rest of Monte Carlo approximation values, until sample size 5000
74. while num <= runs:
75.     MC1 = np.append(MC1, MonteCarlo_Direct(num))
76.     num += 1
77.
78. plt.plot(n, MC1, color = 'red', alpha = 0.7, label = 'Monte Carlo (direct)')
79. plt.legend()
80.
81. print("European Call option price (using direct Monte Carlo) is:")
82. print(round(MC1[MC1.size - 1],2))
83.
84. #####
85.
86. # defining monte carlo integration formula WITH antithetic method
87. def MonteCarlo_Anti(runs):
88.     # initialize array that will have payoffs of option
89.     payoffs1 = np.zeros(0)
90.     payoffs2 = np.zeros(0)
91.
92.     # looping through
93.     for i in xrange(runs):
94.         # generate future stock
95.         S_T = generateStock(S_0, r, sigma, T)
96.         S_T_neg = -(generateStock(S_0, r, sigma, T))
97.
98.         # append to the payoffs list whatever the payoff is
99.         payoffs1 = np.append(payoffs1, exp(-r*T)*payoff1(S_T, K))
100.        payoffs2 = np.append(payoffs2, exp(-r*T)*payoff1(np.abs(S_T_neg), K))
101.
102.        sum_payoffs = (payoffs1 + payoffs2) / 2
103.        return np.mean(sum_payoffs)
104.
105.     # array holding Monte Carlo antithetic approximation
106.     MC2 = np.zeros(0)
107.
108.     # counter for loop
109.     num = 1
110.
111.     # loop for rest of Monte Carlo approximation values, until sample size 5000
112.     while num <= runs:
113.         MC2 = np.append(MC2, MonteCarlo_Anti(num))
114.         num += 1
115.
116.     plt.plot(n, MC2, color = 'blue', alpha = 0.7, label = 'Monte Carlo (antithetic)'
117. )
118.     plt.legend()
```

```
118.  
119.     print("European Call option price (using Monte Carlo & Antithetic method) is:")  
  
120.     print(round(MC2[MC2.size - 1],2))  
121.  
122.  
123.  
124.     plt.title("Monte Carlo price estimation")  
125.     plt.xlabel("# of samples")  
126.     plt.ylabel("Value")  
127.  
128.     plt.show()
```

**Output:**

European Call option price (using Black Scholes formula) is:

16.37

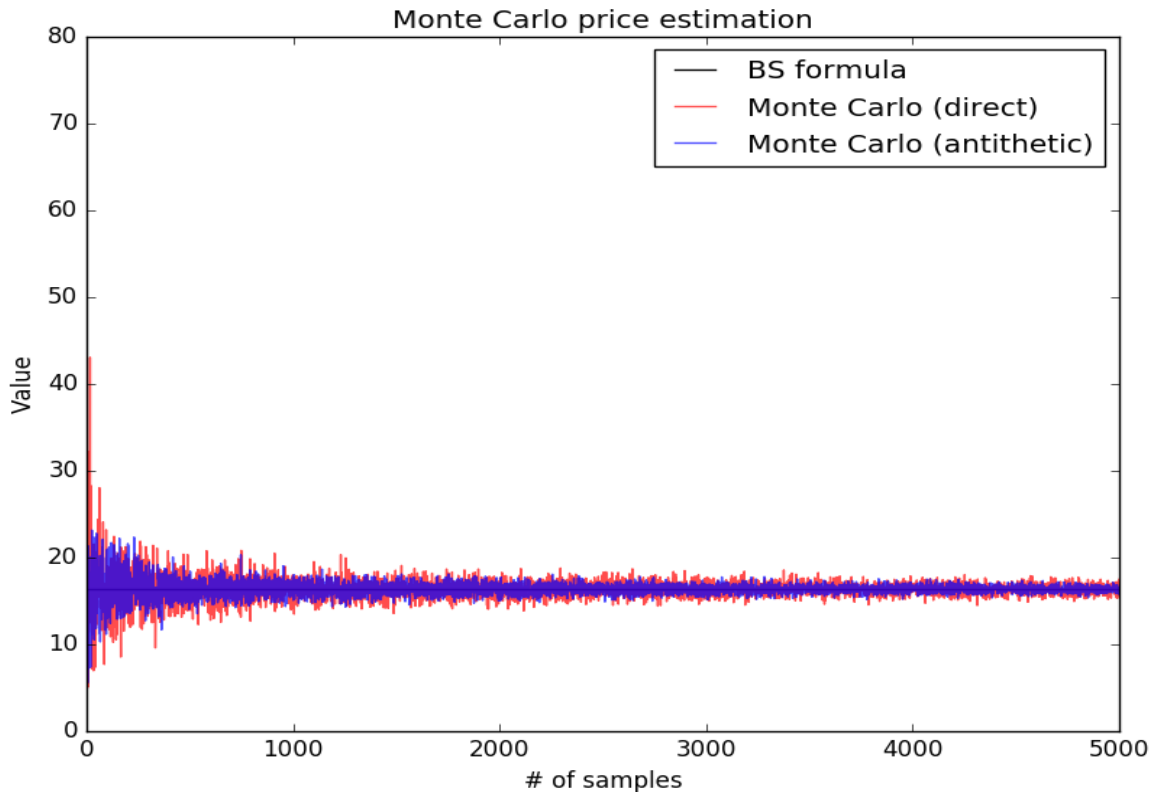
European Call option price (using direct Monte Carlo) is:

17.48

European Call option price (using Monte Carlo & Antithetic method) is:

16.24

**Figure:**



Question 2)

```
1. # importing the necessary packages
2. import numpy as np
3. from numpy import sqrt, exp, log
4. import matplotlib.pyplot as plt
5.
6. #####
7.
8. plt.clf()
9.
10. # variables given
11. S_0 = 100.
12. B0 = 1
13. mu = 0.1
14. sigma = 0.2
15. r = 0.02
16.
17. # number of simulation runs/sample size for Monte Carlo
18. runs = 5000
19.
20. # initialize our Z array used for S_T calculation
21. Z1 = np.random.standard_normal(runs)
22.
23. #####
24. # part a)
25.
26. plt.subplot(2,3,1)
27.
28. T = 0.5
29. K = 95.
30. N = 500
31. #dt = T/runs
32.
33. # calculating S_T array
34. S_T = S_0 * exp((r - (sigma**2)/2)*T + sigma * sqrt(T) * Z1)
35.
36. # calculation of European put payoffs
37. temp = K - S_T
38. for x in np.nditer(temp, op_flags = ['readwrite']):
39.     x[...] = max(x, 0) # in order to make negative values into 0
40. Euro_put_payoffs1 = temp
41.
42. # calculation of correlation
43. Correlation_a = np.corrcoef(S_T, Euro_put_payoffs1)
44. print("The correlation between underlying stock and a European put option is: ")
45. print(Correlation_a[0,1])
46.
47. # plotting
48. plt.scatter(S_T, Euro_put_payoffs1)
49. plt.title("Underlying & Euro Put")
50. plt.xlabel("Underlying Stock")
51. plt.ylabel("Payoffs")
52.
53. #####
54. # part b)
55.
56. plt.subplot(2,3,2)
57.
58. # use same T and K
59.
60. # calculating S_T array
```

```
61. S_T = S_0 * exp((r - (sigma**2)/2)*T + sigma * sqrt(T) * Z1)
62.
63. # calculation of European call payoffs
64. temp = S_T - K
65. for x in np.nditer(temp, op_flags = ['readwrite']):
66.     x[...] = max(x, 0) # in order to make negative values into 0
67. Euro_call_payoffs = temp
68.
69. # calculation of correlation
70. Correlation_b = np.corrcoef(S_T, Euro_call_payoffs)
71. print("The correlation between underlying stock and a European call option is: ")
72. print(Correlation_b[0,1])
73.
74. # plotting
75. plt.scatter(S_T, Euro_call_payoffs)
76. plt.title("Underlying & Euro Call")
77. plt.xlabel("Underlying Stock")
78. plt.ylabel("Payoffs")
79.
80. #####
81. # part c)
82.
83. plt.subplot(2,3,3)
84.
85. # use same T and K
86.
87. # initialize array to hold average values
88. Asian_average_arith1 = np.zeros(0)
89.
90. # array to hold S(T) terminal values, for plotting purposes
91. ST_array1 = np.zeros(0)
92.
93. # loop for getting St arithmetic average values. Always use (1/N) for
94. # average calculation
95.
96. # USING 180 DAYS AS HALF A YEAR (T = 0.5)
97.
98. num = 0
99. while num < runs:
100.     BM_sum = 0
101.     S_t = np.zeros(0)
102.
103.     # consider cumulative sum if time permits
104.
105.     # calculating St array
106.     i = 0
107.     while i < 180:
108.         # have to make new Brownian Motion for each increment step
109.         Z2 = np.random.standard_normal()
110.         BM_increment = sqrt(1/180.) * Z2
111.         BM_sum += BM_increment
112.
113.         S_t = np.append(S_t, S_0 * exp((r - (sigma**2)/2)*(i+1)*(1/180.) + sigma
            * BM_sum))
114.         i += 1
115.
116.     # keep the S_T terminal value for plotting
117.     ST_array1 = np.append(ST_array1, S_t[S_t.size - 1])
118.
119.     # calculate average value
120.     Asian_average_arith1 = np.append(Asian_average_arith1, sum(S_t)/S_t.size)
121.     num += 1
```

```
122.
123.     # Asian arithmetic put payoffs
124.     temp = K - Asian_average_arith1
125.     for x in np.nditer(temp, op_flags = ['readwrite']):
126.         x[...] = max(x, 0) # in order to make negative values into 0
127.     Asian_put_payoffs_arith1 = temp
128.
129.
130.     # calculation of correlation
131.     Correlation_c = np.corrcoef(ST_array1, Asian_put_payoffs_arith1)
132.     print("The correlation between underlying stock and arithmetic average Asian put
is: ")
133.     print(Correlation_c[0,1])
134.
135.     # plotting
136.     plt.scatter(ST_array1, Asian_put_payoffs_arith1)
137.     plt.title("Underlying & arithmetic average Asian Put")
138.     plt.xlabel("Underlying Stock")
139.     plt.ylabel("Payoffs")
140.
141.
142.     #####
143.     # part d)
144.
145.     plt.subplot(2,3,4)
146.
147.     # use same T and K
148.
149.     # initialize array to hold average values
150.     Asian_average_arith2 = np.zeros(0)
151.
152.     # array to hold S(T) terminal values, for Euro put payoff calculations
153.     ST_array2 = np.zeros(0)
154.
155.     # loop for getting St arithmetic average values.
156.     num = 0
157.     while num < runs:
158.         BM_sum = 0
159.         S_t = np.zeros(0)
160.
161.         # consider cumulative sum if time permits
162.
163.         # calculating St array
164.         i = 0
165.         while i < 180:
166.             # have to make new Brownian Motion for each increment step
167.             Z2 = np.random.standard_normal()
168.             BM_increment = sqrt(1/180.) * Z2
169.             BM_sum += BM_increment
170.
171.             S_t = np.append(S_t, S_0 * exp((r - (sigma**2)/2)*(i+1)*(1/180.) + sigma
* BM_sum))
172.             i += 1
173.
174.         # keep the S_T terminal value for Euro put payoff calculation
175.         ST_array2 = np.append(ST_array2, S_t[S_t.size - 1])
176.
177.         # calculate average value
178.         Asian_average_arith2 = np.append(Asian_average_arith2, sum(S_t)/S_t.size)
179.         num += 1
180.
181.     # Asian arithmetic put payoffs
```

```
182.     temp = K - Asian_average_arith2
183.     for x in np.nditer(temp, op_flags = ['readwrite']):
184.         x[...] = max(x, 0) # in order to make negative values into 0
185.     Asian_put_payoffs_arith2 = temp
186.
187.     # calculation of European put payoffs
188.     temp = K - ST_array2
189.     for x in np.nditer(temp, op_flags = ['readwrite']):
190.         x[...] = max(x, 0) # in order to make negative values into 0
191.     Euro_put_payoffs2 = temp
192.
193.     # calculation of correlation
194.     Correlation_d = np.corrcoef(Euro_put_payoffs2, Asian_put_payoffs_arith2)
195.     print("The correlation between European put and arithmetic Asian put is: ")
196.     print(Correlation_d[0,1])
197.
198.     # plotting
199.     plt.scatter(Euro_put_payoffs2, Asian_put_payoffs_arith2)
200.     plt.title("Euro Put & arithmetic average Asian Put")
201.     plt.xlabel("Euro payoffs")
202.     plt.ylabel("Asian payoffs")
203.
204.     #####
205.     # part e)
206.
207.     plt.subplot(2,3,5)
208.
209.     # use same T and K
210.
211.     # initialize array to hold average values
212.     Asian_average_geo1 = np.zeros(0)
213.
214.     # array to hold S(T) terminal values, for Euro put payoff calculations
215.     ST_array3 = np.zeros(0)
216.
217.     # loop for getting St arithmetic average values.
218.     num = 0
219.     while num < runs:
220.         BM_sum = 0
221.         S_t = np.zeros(0)
222.
223.         # consider cumulative sum if time permits
224.
225.         # calculating St array
226.         i = 0
227.         while i < 180:
228.             # have to make new Brownian Motion for each increment step
229.             Z2 = np.random.standard_normal()
230.             BM_increment = sqrt(1/180.) * Z2
231.             BM_sum += BM_increment
232.
233.             S_t = np.append(S_t, S_0 * exp((r - (sigma**2)/2)*(i+1)*(1/180.) + sigma
* BM_sum))
234.             i += 1
235.
236.         # keep the S_T terminal value for Euro put payoff calculation
237.         ST_array3 = np.append(ST_array3, S_t[S_t.size - 1])
238.
239.         # calculate average value
240.         # slight adjustment of product and nth root calculation
241.         # first take nth root, then take product
```

```

242.         Asian_average_geo1 = np.append(Asian_average_geo1, np.product(S_t**(1/180.))
    )
243.         num += 1
244.
245.         # Asian geometric put payoffs
246.         temp = K - Asian_average_geo1
247.         for x in np.nditer(temp, op_flags = ['readwrite']):
248.             x[...] = max(x, 0) # in order to make negative values into 0
249.         Asian_put_payoffs_geo1 = temp
250.
251.         # calculation of European put payoffs
252.         temp = K - ST_array3
253.         for x in np.nditer(temp, op_flags = ['readwrite']):
254.             x[...] = max(x, 0) # in order to make negative values into 0
255.         Euro_put_payoffs3 = temp
256.
257.         # calculation of correlation
258.         Correlation_e = np.corrcoef(Euro_put_payoffs3, Asian_put_payoffs_geo1)
259.         print("The correlation between European put and geometric Asian put is: ")
260.         print(Correlation_e[0,1])
261.
262.         # plotting
263.         plt.scatter(Euro_put_payoffs3, Asian_put_payoffs_geo1)
264.         plt.title("Euro Put & geometric average Asian Put")
265.         plt.xlabel("Euro payoffs")
266.         plt.ylabel("Asian payoffs")
267.
268.         #####
269.         # part f)
270.
271.         plt.subplot(2,3,6)
272.
273.         # use same T and K
274.
275.         # initialize array to hold average values
276.         Asian_average_geo2 = np.zeros(0)
277.         Asian_average_arith3 = np.zeros(0)
278.
279.         # loop for getting St arithmetic average values.
280.         num = 0
281.         while num < runs:
282.             BM_sum = 0
283.             S_t = np.zeros(0)
284.
285.             # consider cumulative sum if time permits
286.
287.             # calculating St array
288.             i = 0
289.             while i < 180:
290.                 # have to make new Brownian Motion for each increment step
291.                 Z2 = np.random.standard_normal()
292.                 BM_increment = sqrt(1/180.) * Z2
293.                 BM_sum += BM_increment
294.
295.                 S_t = np.append(S_t, S_0 * exp((r - (sigma**2)/2)*(i+1)*(1/180.) + sigma
    * BM_sum))
296.                 i += 1
297.
298.             # keep the S_T terminal value for Euro put payoff calculation
299.             #ST_array4 = np.append(ST_array2, S_t[S_t.size - 1])
300.
301.             # calculate average value

```



```

302.         Asian_average_geo2 = np.append(Asian_average_geo2,
303.                                         np.product(S_t**(1/180.)))
304.         Asian_average_arith3 = np.append(Asian_average_arith3, sum(S_t)/S_t.size)
305.         num += 1
306.
307.         # Asian geometric put payoffs
308.         temp = K - Asian_average_geo2
309.         for x in np.nditer(temp, op_flags = ['readwrite']):
310.             x[...] = max(x, 0) # in order to make negative values into 0
311.         Asian_put_payoffs_geo2 = temp
312.
313.         # Asian arithmetic put payoffs
314.         temp = K - Asian_average_arith3
315.         for x in np.nditer(temp, op_flags = ['readwrite']):
316.             x[...] = max(x, 0) # in order to make negative values into 0
317.         Asian_put_payoffs_arith3 = temp
318.
319.         # calculation of correlation
320.         Correlation_f = np.corrcoef(Asian_put_payoffs_geo2, Asian_put_payoffs_arith3)
321.         print("The correlation between European put and geometric Asian put is: ")
322.         print(Correlation_f[0,1])
323.
324.         # plotting
325.         plt.scatter(Asian_put_payoffs_geo2, Asian_put_payoffs_arith3)
326.         plt.title("Geometric Asian put & arithmetic Asian Put")
327.         plt.xlabel("Geometric payoffs")
328.         plt.ylabel("Arithmetic payoffs")
329.
330.         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

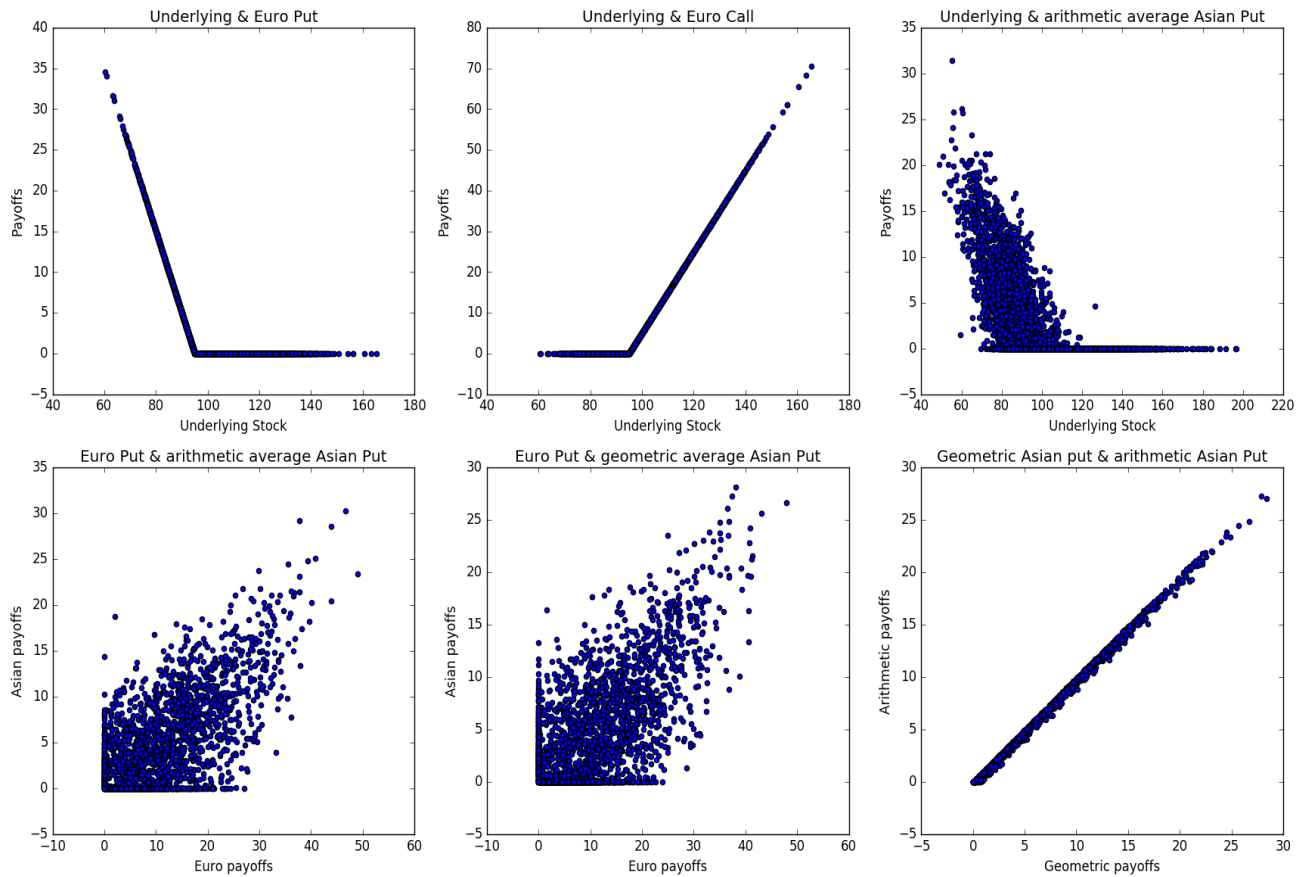
The correlation between European put and geometric Asian put is:

0.785733209395

The correlation between European put and geometric Asian put is:

0.999505638744

Figure:



Question 3)

```
1. # importing the necessary packages
2. import numpy as np
3. from numpy import sqrt, exp, log
4. from scipy.stats import norm
5. import matplotlib.pyplot as plt
6.
7. #####
8.
9. # variables given
10. S_0 = 100.
11. B0 = 1
12. mu = 0.3
13. sigma = 0.3
14. r = 0.03
15.
16. # Asian Call
17. K = 120.
18. T = 5
19.
20. # number of simulation runs/sample size for Monte Carlo
21. runs = 10000
22.
23. # number of average time for Asian payoff calculation
24. N = 1260.
25.
26. dt = T/N
27.
28. #####
29. # part c)
30.
31. # Using the Geometric Asian Average as the Control Variate
32.
33. # Need actual value of Geometric Asian call option
34. sigma_geo = sigma * sqrt(1/3.)
35. r_geo = 0.5 * (r - (1/6.) * sigma**2)
36. d1 = (log(S_0/K) + 0.5 * (r + (1/6.)*sigma**2) * T) / (sigma_geo * sqrt(T))
37. d2 = d1 - (sigma_geo * sqrt(T))
38. Asian_geo_price = exp(-(r*T)) * (S_0*exp(r_geo*T)*norm.cdf(d1) - K*norm.cdf(d2))
39.
40. # initialize array to hold average values
41. Asian_avg_geo = np.zeros(0)
42. Asian_avg_arith = np.zeros(0)
43.
44. # loop for getting St arithmetic average values.
45. num = 0
46. while num < runs:
47.     BM_sum = 0
48.     S_t = np.zeros(0)
49.
50.     # calculating St array
51.     i = 0
52.     while i < int(N):
53.         # have to make new Brownian Motion for each increment step
54.         Z = np.random.standard_normal()
55.         BM_increment = sqrt(dt) * Z
56.         BM_sum += BM_increment
57.
58.         S_t = np.append(S_t, S_0 * exp((r - (sigma**2)/2)*(i+1)*(dt) + sigma * BM_sum))
59.         i += 1
```

```
60.
61.     # calculate average value
62.     Asian_avg_geo = np.append(Asian_avg_geo, np.product(S_t**(1/N)))
63.     Asian_avg_arith = np.append(Asian_avg_arith, sum(S_t)/S_t.size)
64.     num += 1
65.
66. # Asian geometric call payoffs
67. temp = Asian_avg_geo - K
68. for x in np.nditer(temp, op_flags = ['readwrite']):
69.     x[...] = max(x, 0) # in order to make negative values into 0
70. Asian_call_payoffs_geo = temp
71.
72. # Asian arithmetic call payoffs
73. temp = Asian_avg_arith - K
74. for x in np.nditer(temp, op_flags = ['readwrite']):
75.     x[...] = max(x, 0) # in order to make negative values into 0
76. Asian_call_payoffs_arith = temp
77.
78. # calculate the optimal b value for the equation sampling
79. Cov_test = np.cov(Asian_call_payoffs_geo, Asian_call_payoffs_arith)
80. b_optimal = Cov_test[0,1] / Cov_test[1,1]
81.
82. # new sampling, using the control variate
83. Asian_sampling = Asian_call_payoffs_arith - b_optimal * (Asian_call_payoffs_geo - Asian
    _geo_price)
84.
85. Asian_call_price = exp(-r*T) * np.mean(Asian_sampling)
86. print("The estimated price of arithmetic average Asian Call Option (with Control Variat
    e) is:")
87. print(round(Asian_call_price,2))
88.
89. #####
90. # part d)
91.
92. Y_asian_call_control = np.zeros(0)
93. Y_asian_call_NoControl = np.zeros(0)
94.
95. # getting the respective arithmetic Asian call prices (WITH CONTROL VARIATE)
96. # Asian_sampling is from part c)
97. for i in range(runs):
98.     Y_asian_call_control = np.append(Y_asian_call_control,
99.                                     exp(-
    r*T) * (sum(Asian_sampling[: (i+1)]) / (Asian_sampling[: (i+1)].size)))
100.
101.     # getting the respective arithmetic Asian call prices based on sample size
102.     for j in range(runs):
103.         Y_asian_call_NoControl = np.append(Y_asian_call_NoControl,
104.                                             exp(-r*T) *
105.                                             (sum(Asian_call_payoffs_arith[: (j+1)])
    / (Asian_call_payoffs_arith[: (j+1)].size
    )))
106.
107.
108.
109.     plt.plot(np.arange(runs), Y_asian_call_control, label = "Monte Carlo w/ Control
    Variate")
110.     plt.plot(np.arange(runs), Y_asian_call_NoControl, label = "Direct Monte Carlo")
111.
112.     plt.title("Monte Carlo Calculation of Arithmetic Asian Call Price")
113.     plt.xlabel("Sample size")
114.     plt.ylabel("Price")
```

```
115.  
116.     plt.legend(loc = 1)  
117.     plt.show()  
118.  
119.     #####  
120.     # part d)  
121.     print("-----  
")  
122.     print("Actual price of Asian geometric call is: ")  
123.     print(round(Asian_geo_price,2))  
124.  
125.     print("The estimated price of arithmetic average Asian Call Option (with Control  
Variate) is:")  
126.     print(round(Asian_call_price,2))  
127.     print("The estimated price of arithmetic average Asian Call (using only Direct M  
onte Carlo) is: ")  
128.     print(round(Y_asian_call_NoControl[runs - 1], 2))  
129.  
130.     # calculations for Euro Call calculation  
131.  
132.     Z1 = np.random.standard_normal(runs)  
133.     # calculating S_T array  
134.     S_T = S_0 * exp((r - (sigma**2)/2)*T + sigma * sqrt(T) * Z1)  
135.     # calculation of European call payoffs  
136.     temp = S_T - K  
137.     for x in np.nditer(temp, op_flags = ['readwrite']):  
138.         x[...] = max(x, 0) # in order to make negative values into 0  
139.     Euro_call_payoffs = temp  
140.     Euro_call_price = exp(-r*T) * np.mean(Euro_call_payoffs)  
141.  
142.     print("Price of European call option is: ")  
143.     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:

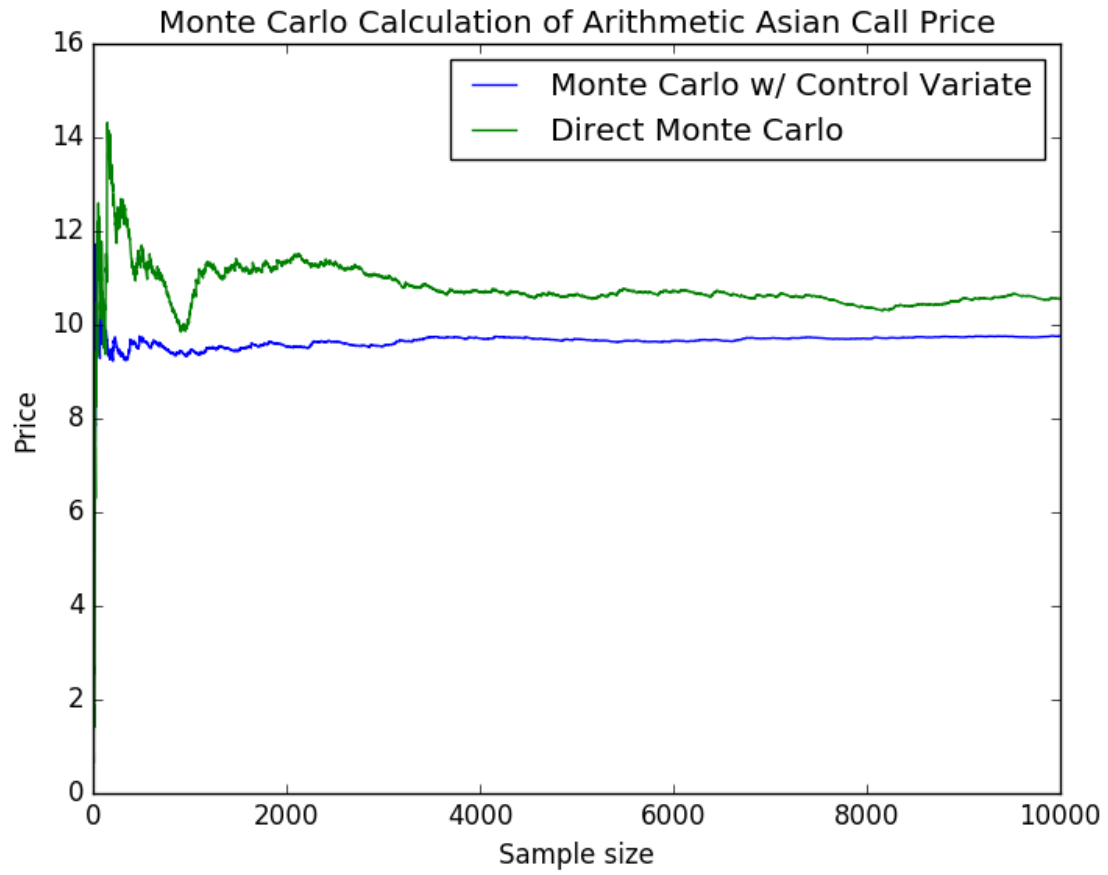
10.71

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:



Question 4)

```
1. # importing the necessary packages
2. import numpy as np
3. from numpy import exp, log, sqrt
4. import matplotlib.pyplot as plt
5. import math
6.
7. #####
8. #part a)
9.
10. plt.clf()
11.
12. # variables given
13. S_0 = 100.
14.
15. mu = 0.1
16. sigma = 0.2
17.
18. T = 10
19. N = 10000.    #10 years, and 1000 steps per year
20. dt = T/N
21.
22. lambda_var = 2.
23.
24. #simulation runs
25. runs = 1000.
26.
27. # our large Y array which will have 1000 samples
28. Y_stratified = np.zeros(0)
29.
30. # inverse CDF of exponential
31. def InverseExp(x):
32.     y = - 0.5 * (log(1 - x))
33.     return y
34.
35. # Need to get stratified intervals
36. a_stratPoints = np.zeros(0)
37. for i in range(10):
38.     val = (i+1)/float(10)
39.     a_stratPoints = np.append(a_stratPoints, InverseExp(val))
40.
41. U1 = np.random.random_sample(100)
42. Y_stratified = np.append(Y_stratified, 0 + (a_stratPoints[0] - 0)*U1 )
43.
44. # loop for Yi values
45. for j in range(9):
46.     U2 = np.random.random_sample(100)
47.
48.     Y_stratified = np.append(Y_stratified, a_stratPoints[j] + (a_stratPoints[j+1] - a_s
       stratPoints[j])*U2)
49.
50.
51. # plot and info for regular density of exponential, and non-stratified sampling
52. x = np.random.random_sample(1000)
53. y = InverseExp(x)
54. z = np.arange(0, 3, 0.1)
55. w = lambda_var * np.exp(-z * lambda_var)*100
56.
57. plt.plot(z, w, color = 'red', label = 'Exp density')
58. plt.hist(y, range = (0,3), bins = 30, color = 'blue', label = 'Non-
    stratified sampling')
```

```

59. plt.hist(Y_stratified, range = (0,3), bins = 30, color = 'yellow', label = 'Stratified
    sampling')
60.
61. plt.title ("Exponential Histogram")
62. plt.xlabel ("x value")
63. plt.ylabel ("Density x 1000")
64.
65. plt.legend()
66. plt.show()
67.
68. #####
69. #part b)
70.
71. # array to keep ST terminal stock values, for expectation calculation
72. ST_array = np.zeros(0)
73.
74. num = 0
75. while num < runs:
76.     S_t = np.zeros(0)
77.     BM_sum = 0
78.
79.     # calculating St array
80.     k = 0
81.     tau = Y_stratified[num]
82.
83.     # if no default occurred, then set time to final terminal time (aka. T = 10)
84.     if tau > 1.0:
85.         tau = 1.0
86.
87.     # St simulation loop, until time tau.
88.     while k < tau:
89.         # have to make new Brownian Motion for each increment step
90.         Z = np.random.standard_normal()
91.         BM_increment = sqrt(dt) * Z
92.         BM_sum += BM_increment
93.
94.         S_t = np.append(S_t, S_0 * exp((mu - 0.5*(sigma**2)*(k+1)*(dt) + sigma * BM_sum
    )))
95.         k += 0.001    #1000 steps each year
96.
97.     ST_array = np.append(ST_array, S_t[S_t.size - 1])
98.     num += 1
99.
100.    PriceAtDefault = sum(ST_array) / ST_array.size
101.
102.    print("The expected price at default is: ")
103.    print(PriceAtDefault)

```

**Output:**

The expected price at default is:

111.740436214



Figure:

