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
7. ##############################################################################
9. plt.clf()
11. # variables given
12. S\_0 = 100.
13. B0 = 1
14. mu = 0.05
15. sigma = 0.35
16. r = 0.05
18. # number of simulation runs/sample size for Monte Carlo
19. runs = 5000
21. # setting up sample size n x-axis variable for plotting
22. n = np.arange(0.0, runs, 1.0)
24. # European Call, using Black Scholes formula
25. K = 120.
26. T = 2
28. d1 = (log(S\_0/K) + (r + (sigma\*\*2)/2)\*T) / (sigma \* sqrt(T))
29. d2 = d1 - sigma\*sqrt(T)
31. # black scholes formula price calculation
32. Price1 = S\_0\*norm.cdf(d1) - K\*exp(-r\*T)\*norm.cdf(d2)
34. **print**("European Call option price (using Black Scholes formula) is:")
35. **print**(round(Price1,2))
37. plt.axhline(y = Price1, color = 'black', linestyle = '-', label = 'BS formula')
38. plt.legend()
40. #####################################
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())
46. # defining payoff function, for this case it's regular European call
47. **def** payoff1(S, K):
48. **return** max(S - K, 0)
50. # defining monte carlo integration formula (direct version)
51. **def** MonteCarlo\_Direct(runs):
52. # initialize array that will have payoffs of option
53. 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)
60. # append to the payoffs list whatever the payoff is
61. payoffs = np.append(payoffs,
62. exp(-r\*T)\*payoff1(S\_T, K))
64. **return** sum(payoffs)/runs
66. # array holding Monte Carlo direct approximation values
67. # this is also initial point, n = 0
68. MC1 = np.zeros(0)
70. # counter for loop
71. num = 1
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
78. plt.plot(n, MC1, color = 'red', alpha = 0.7, label = 'Monte Carlo (direct)')
79. plt.legend()
81. **print**("European Call option price (using direct Monte Carlo) is:")
82. **print**(round(MC1[MC1.size - 1],2))
84. #####################################
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)
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))
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))
102. sum\_payoffs = (payoffs1 + payoffs2) / 2
103. **return** np.mean(sum\_payoffs)
105. # array holding Monte Carlo antithetic approximation
106. MC2 = np.zeros(0)
108. # counter for loop
109. num = 1
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
116. plt.plot(n, MC2, color = 'blue', alpha = 0.7, label = 'Monte Carlo (antithetic)')
117. plt.legend()
119. **print**("European Call option price (using Monte Carlo & Antithetic method) is:")
120. **print**(round(MC2[MC2.size - 1],2))


124. plt.title("Monte Carlo price estimation")
125. plt.xlabel("# of samples")
126. plt.ylabel("Value")
128. plt.show()

**Output:**

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

16.37

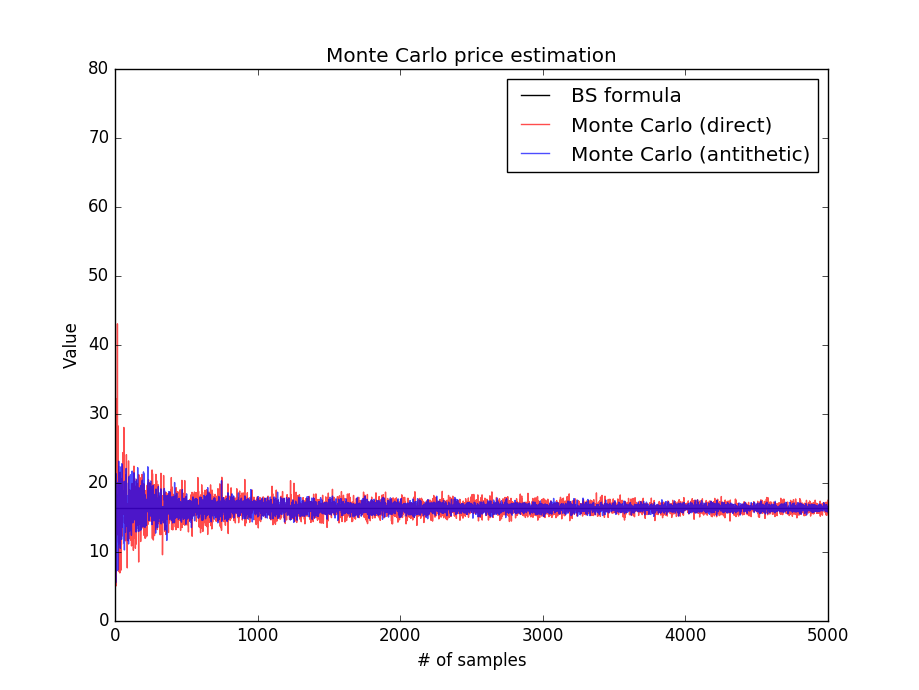
European Call option price (using direct Monte Carlo) is:

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
6. ##############################################################################
8. plt.clf()
10. # variables given
11. S\_0 = 100.
12. B0 = 1
13. mu = 0.1
14. sigma = 0.2
15. r = 0.02
17. # number of simulation runs/sample size for Monte Carlo
18. runs = 5000
20. # initalize our Z array used for S\_T calculation
21. Z1 = np.random.standard\_normal(runs)
23. ###########################################
24. # part a)
26. plt.subplot(2,3,1)
28. T = 0.5
29. K = 95.
30. N = 500
31. #dt = T/runs
33. # calculating S\_T array
34. S\_T = S\_0 \* exp((r - (sigma\*\*2)/2)\*T + sigma \* sqrt(T) \* Z1)
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
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])
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")
53. ###########################################
54. # part b)
56. plt.subplot(2,3,2)
58. # use same T and K
60. # calculating S\_T array
61. S\_T = S\_0 \* exp((r - (sigma\*\*2)/2)\*T + sigma \* sqrt(T) \* Z1)
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
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])
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")
80. ###########################################
81. # part c)
83. plt.subplot(2,3,3)
85. # use same T and K
87. # initialize array to hold average values
88. Asian\_average\_arith1 = np.zeros(0)
90. # array to hold S(T) terminal values, for plotting purposes
91. ST\_array1 = np.zeros(0)
93. # loop for getting St arithmetic average values. Always use (1/N) for
94. # average calculation
96. # USING 180 DAYS AS HALF A YEAR (T = 0.5)
98. num = 0
99. **while** num < runs:
100. BM\_sum = 0
101. S\_t = np.zeros(0)
103. # consider cumulative sum if time permits
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
113. S\_t = np.append(S\_t, S\_0 \* exp((r - (sigma\*\*2)/2)\*(i+1)\*(1/180.) + sigma \* BM\_sum))
114. i += 1
116. # keep the S\_T terminal value for plotting
117. ST\_array1 = np.append(ST\_array1, S\_t[S\_t.size - 1])
119. # calculate average value
120. Asian\_average\_arith1 = np.append(Asian\_average\_arith1, sum(S\_t)/S\_t.size)
121. num += 1
123. # Asian arithmatic 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

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])
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")

142. ###########################################
143. # part d)
145. plt.subplot(2,3,4)
147. # use same T and K
149. # initialize array to hold average values
150. Asian\_average\_arith2 = np.zeros(0)
152. # array to hold S(T) terminal values, for Euro put payoff calculations
153. ST\_array2 = np.zeros(0)
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)
161. # consider cumulative sum if time permits
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
171. S\_t = np.append(S\_t, S\_0 \* exp((r - (sigma\*\*2)/2)\*(i+1)\*(1/180.) + sigma \* BM\_sum))
172. i += 1
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])
177. # calculate average value
178. Asian\_average\_arith2 = np.append(Asian\_average\_arith2, sum(S\_t)/S\_t.size)
179. num += 1
181. # Asian arithmatic 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
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
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])
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")
204. ###########################################
205. # part e)
207. plt.subplot(2,3,5)
209. # use same T and K
211. # initialize array to hold average values
212. Asian\_average\_geo1 = np.zeros(0)
214. # array to hold S(T) terminal values, for Euro put payoff calculations
215. ST\_array3 = np.zeros(0)
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)
223. # consider cumulative sum if time permits
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
233. S\_t = np.append(S\_t, S\_0 \* exp((r - (sigma\*\*2)/2)\*(i+1)\*(1/180.) + sigma \* BM\_sum))
234. i += 1
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])
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
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
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
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])
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")
268. ###########################################
269. # part f)
271. plt.subplot(2,3,6)
273. # use same T and K
275. # initialize array to hold average values
276. Asian\_average\_geo2 = np.zeros(0)
277. Asian\_average\_arith3 = np.zeros(0)
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)
285. # consider cumulative sum if time permits
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
295. S\_t = np.append(S\_t, S\_0 \* exp((r - (sigma\*\*2)/2)\*(i+1)\*(1/180.) + sigma \* BM\_sum))
296. i += 1
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])
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
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
313. # Asian arithmatic 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
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])
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")
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:

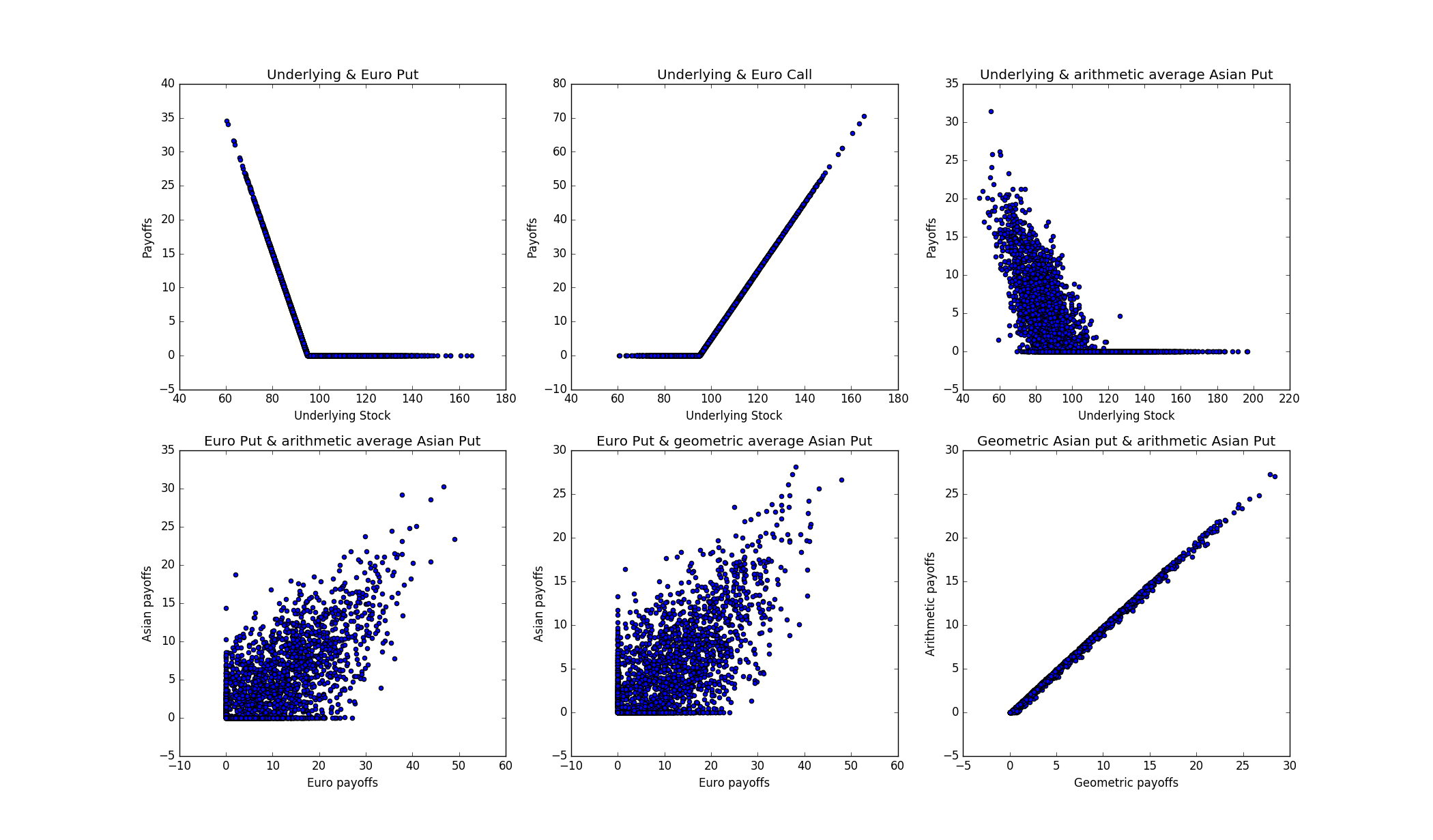
0.78219059056

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
7. ##############################################################################
9. # variables given
10. S\_0 = 100.
11. B0 = 1
12. mu = 0.3
13. sigma = 0.3
14. r = 0.03
16. # Asian Call
17. K = 120.
18. T = 5
20. # number of simulation runs/sample size for Monte Carlo
21. runs = 10000
23. # number of average time for Asian payoff calculation
24. N = 1260.
26. dt = T/N
28. #####################################
29. # part c)
31. # Using the Geometric Asian Average as the Control Variate
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))
40. # initialize array to hold average values
41. Asian\_avg\_geo = np.zeros(0)
42. Asian\_avg\_arith = np.zeros(0)
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)
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
58. S\_t = np.append(S\_t, S\_0 \* exp((r - (sigma\*\*2)/2)\*(i+1)\*(dt) + sigma \* BM\_sum))
59. i += 1
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
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
72. # Asian arithmatic 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
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]
82. # new sampling, using the control variate
83. Asian\_sampling = Asian\_call\_payoffs\_arith - b\_optimal \* (Asian\_call\_payoffs\_geo - Asian\_geo\_price)
85. Asian\_call\_price = exp(-r\*T) \* np.mean(Asian\_sampling)
86. **print**("The estimated price of arithmetic average Asian Call Option (with Control Variate) is:")
87. **print**(round(Asian\_call\_price,2))
89. #####################################
90. # part d)
92. Y\_asian\_call\_control = np.zeros(0)
93. Y\_asian\_call\_NoControl = np.zeros(0)
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)))
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)])
106. / (Asian\_call\_payoffs\_arith[:(j+1)].size)))

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")
112. plt.title("Monte Carlo Calculation of Arithmetic Asian Call Price")
113. plt.xlabel("Sample size")
114. plt.ylabel("Price")
116. plt.legend(loc = 1)
117. plt.show()
119. #####################################
120. # part d)
121. **print**("----------------------------------------------------------------------")
122. **print**("Actual price of Asian geometric call is: ")
123. **print**(round(Asian\_geo\_price,2))
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 Monte Carlo) is: ")
128. **print**(round(Y\_asian\_call\_NoControl[runs - 1], 2))
130. # calculations for Euro Call calculation
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)
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:

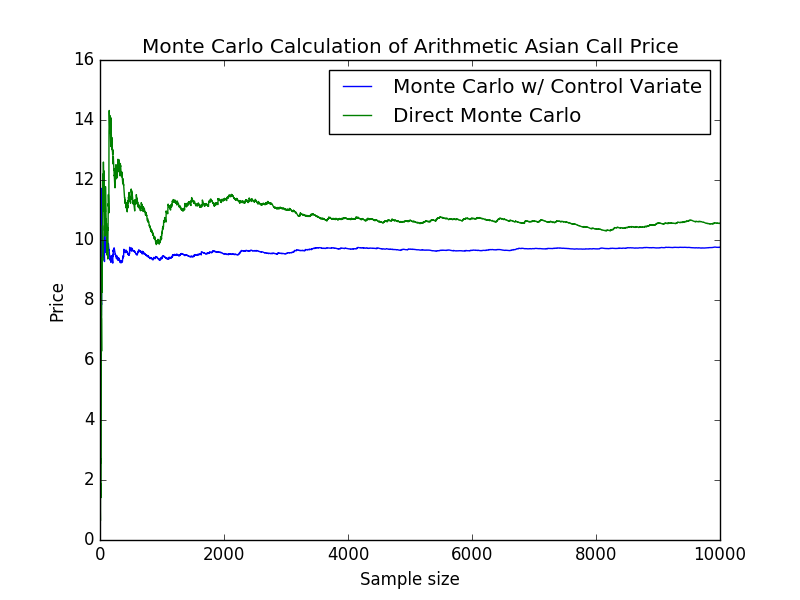
10.71

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
7. ##############################################################################
8. #part a)
10. plt.clf()
12. # variables given
13. S\_0 = 100.
15. mu = 0.1
16. sigma = 0.2
18. T = 10
19. N = 10000.   #10 years, and 1000 steps per year
20. dt = T/N
22. lambda\_var = 2.
24. #simulation runs
25. runs = 1000.
27. # our large Y array which will have 1000 samples
28. Y\_stratified = np.zeros(0)
30. # inverse CDF of exponential
31. **def** InverseExp(x):
32. y = - 0.5 \* (log(1 - x))
33. **return** y
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))
41. U1 = np.random.random\_sample(100)
42. Y\_stratified = np.append(Y\_stratified, 0 + (a\_stratPoints[0] - 0)\*U1 )
44. # loop for Yi values
45. **for** j **in** range(9):
46. U2 = np.random.random\_sample(100)
48. Y\_stratified = np.append(Y\_stratified, a\_stratPoints[j] + (a\_stratPoints[j+1] - a\_stratPoints[j])\*U2)

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
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')
61. plt.title ("Exponential Histogram")
62. plt.xlabel ("x value")
63. plt.ylabel ("Density x 1000")
65. plt.legend()
66. plt.show()
68. #############################################################################
69. #part b)
71. # array to keep ST terminal stock values, for expectation calculation
72. ST\_array = np.zeros(0)
74. num = 0
75. **while** num < runs:
76. S\_t = np.zeros(0)
77. BM\_sum = 0
79. # calculating St array
80. k = 0
81. tau = Y\_stratified[num]
83. # if no default occured, then set time to final terminal time (aka. T = 10)
84. **if** tau > 1.0:
85. tau = 1.0
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
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
97. ST\_array = np.append(ST\_array, S\_t[S\_t.size - 1])
98. num += 1
100. PriceAtDefault = sum(ST\_array) / ST\_array.size
102. **print**("The expected price at default is: ")
103. **print**(PriceAtDefault)

**Output:**

The expected price at default is:

111.740436214

**Figure**:

