Question 3)

Code:

1. # importing the necessary packages
2. **import** numpy as np
3. **from** numpy **import** sqrt, exp, log, mean
4. **from** scipy.stats **import** norm
5. **from** scipy **import** optimize
7. ##############################################################################
9. # increment step method for Brownian Motion
10. **def** SimBMStep(T, N):
11. # initialize W brownian motion array
12. W = np.zeros(int(N))
13. W = np.append(W, 0)    # initial 0 value
15. # loop
16. num = 0
17. **while** num < N:
18. W[num + 1] = W[num] + sqrt(T/N)\*np.random.standard\_normal()
19. num += 1
21. **return** W
23. # initial vars
24. s = 143.    # retrieved from APPL stock
25. sigma\_init = 3.0   # initial guess for sigma
27. # number retrieved from 30yr treasury bond yield curve rate
28. r = 0.0320
30. # day form
31. T = 36/365.
32. runs = 1000
34. # this is all XDATA
35. # these are the K strike CALL options that have trading volume of more than 10
36. CK = np.array([50, 110, 120, 130, 135, 140, 145, 150, 155, 160, 165, 170, 175, 180])
38. #CK\_thetaPrices = np.zeros(0)
39. CK\_actualPrices = np.array([mean([86.40,87.10]),
40. mean([33.50,34.25]),
41. mean([24.15,24.45]),
42. mean([14.50,14.65]),
43. mean([10.15,10.25]),
44. mean([6.50,6.55]),
45. mean([3.75,3.80]),
46. mean([1.89,1.89]),
47. mean([0.89,0.91]),
48. mean([0.42,0.44]),
49. mean([0.21,0.22]),
50. mean([0.10,0.11]),
51. mean([0.05,0.06]),
52. mean([0.02,0.03])])

55. **print**("K strike values for Call options (these have volume traded > 10):")
56. **print**(CK)
57. **print**("Call option actual prices (ordered with their respective K's): ")
58. **print**(CK\_actualPrices)
60. # payoff function for European Call option
61. **def** payoffEC(ST, K):
62. **return** max(ST - K, 0)
64. # big MAIN FUNCTION that has all parameters for optimization
65. # and outputs the estimated model prices
66. # xdata is the K value
67. **def** PricingUsingModelEC(K, sigma\_init):
68. num = 0
69. totalPayoffs = np.zeros\_like(CK)
70. # monte carlo simulation loop
71. **while** num < runs:
72. payoffs = np.zeros(0)
73. W1 = SimBMStep(T, 500.)
74. temp = 0
75. St = np.zeros(0)
76. St = np.append(St, s)
77. # St simulation loop
78. **while** temp < 500:
79. St = np.append(St, St[temp]
80. + r\*St[temp]\*(T/500.)
81. + sigma\_init\*St[temp]\*(W1[temp+1] - W1[temp]))
82. temp += 1
83. # retrieving the terminal ST
84. ST = St[St.size - 1]
85. payoffs = ST - K
86. # this makes every payoff value >= 0
87. payoffs = [0 **if** i<0 **else** i **for** i **in** payoffs]
88. totalPayoffs = np.sum([payoffs, totalPayoffs], axis = 0)
89. num += 1
90. AveragePayoffs = [i/runs **for** i **in** totalPayoffs]
91. EstimatedPrices = [exp(-r\*T)\*i **for** i **in** AveragePayoffs]
92. **return** EstimatedPrices
94. # regular BS pricing formula
95. **def** PricingUsingClosedForm(K, sigma\_init):
96. d1 = (log(s/K) + (r + (sigma\_init\*\*2)/2)\*T) / (sigma\_init \* sqrt(T))
97. d2 = d1 - sigma\_init\*sqrt(T)
98. # black scholes formula price calculation
99. PriceBS = s\*norm.cdf(d1) - K\*exp(-r\*T)\*norm.cdf(d2)
100. **return** PriceBS

103. # finally, we do curve\_fit optimization, Monte Carlo integration version
104. popt, pcov = optimize.curve\_fit(PricingUsingModelEC, CK, CK\_actualPrices)
105. **print**("MC: If initial sigma guess = " + str(sigma\_init) + ", then optimal sigma = " + str(round(popt,2)))
107. # BS formula version
108. popt, pcov = optimize.curve\_fit(PricingUsingClosedForm, CK, CK\_actualPrices)
109. **print**("BS: If initial sigma guess = " + str(sigma\_init) + ", then optimal sigma = " + str(round(popt,2)))

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.