Question 6)

(this is my old Q6 code from last week. My new code will be submitted in HW8)

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
6. **import** matplotlib.pyplot as plt
8. ##############################################################################
10. # increment step method for Brownian Motion
11. **def** SimBMStep(T, N):
12. # initialize W brownian motion array
13. W = np.zeros(int(N))
14. W = np.append(W, 0)    # initial 0 value
16. # loop
17. num = 0
18. **while** num < N:
19. W[num + 1] = W[num] + sqrt(T/N)\*np.random.standard\_normal()
20. num += 1
22. **return** W
24. # initial vars
25. s = 139.    # retrieved from APPL stock
26. #s = 100.
27. y = 0.08
28. lam = 3
29. kappa = 0.1
30. rho = -0.8
31. xi = 0.25
33. # initial vars array
34. x0 = np.array([lam, kappa, xi, y, rho])
36. # number retrieved from 30yr treasury bond yield curve rate
37. r = 0.0320
39. # day form
40. T = 60/365.
41. runs = 1000
43. # this is all XDATA
44. # these are the K strike CALL options that have trading volume of more than 10
45. CK = np.array([50, 130, 135, 140, 145, 150, 155, 160, 165, 170, 180])
46. # these are the K strike PUT options that have trading volume of more than 10
47. PK = np.array([70, 75, 80, 100, 105, 110, 115, 120, 125, 130, 135, 140, 145, 150])
49. #CK\_thetaPrices = np.zeros(0)
50. CK\_actualPrices = np.array([mean([96.85,97.60]),
51. mean([10.55,10.60]),
52. mean([6.80,6.90]),
53. mean([3.90,4.00]),
54. mean([2.04,2.06]),
55. mean([0.95,0.97]),
56. mean([0.43,0.44]),
57. mean([0.20,0.21]),
58. mean([0.09,0.10]),
59. mean([0.04,0.05]),
60. mean([0.04,0.05])])
62. #PK\_thetaPrices = np.zeros(0)
63. PK\_actualPrices = np.array([mean([0, 0.03]),
64. mean([0.02,0.04]),
65. mean([0.01,0.05]),
66. mean([0.06,0.07]),
67. mean([0.11,0.12]),
68. mean([0.16,0.17]),
69. mean([0.27,0.28]),
70. mean([0.45,0.47]),
71. mean([0.81,0.84]),
72. mean([1.55,1.57]),
73. mean([2.89,2.92]),
74. mean([5.10,5.15]),
75. mean([8.20,8.30]),
76. mean([11.90,12.25])])
78. **print**("K strike values for Call options (these have volume traded > 10):")
79. **print**(CK)
80. **print**("Call option actual prices (ordered with their respective K's): ")
81. **print**(CK\_actualPrices)
83. **print**("K strike values for Put options:")
84. **print**(PK)
85. **print**("Put option actual prices are: ")
86. **print**(PK\_actualPrices)
88. # payoff function for European Call option
89. **def** payoffEC(ST, K):
90. **return** max(ST - K, 0)
92. # payoff function for European Put option
93. **def** payoffEP(ST, K):
94. **return** max(K - ST, 0)
96. # big MAIN FUNCTION that has all parameters for optimization
97. # and outputs the estimated model prices
98. # xdata is the K value
99. **def** PricingUsingModelEC(K, Plam, Pkappa, Pxi, Py, Prho):
100. #lam, kappa, xi, y, rho, = params
101. #errVal = abs(ydata - CThetha)
103. # K loop?
104. CK\_estimatedPrices = np.zeros(0)
105. i = 0
106. **while** i < K.size:
107. # array holding payoffs for price calculation
108. payoffsArray = np.zeros(0)
109. num = 0
110. **while** num < runs:
111. # get W1 and W~ Brownian Motions, in order to get the W2 BM
112. # note: need to use rho correlation equation
113. W1 = SimBMStep(T, 500.)
114. Wtilda = SimBMStep(T, 500.)
115. W2 = Prho\*W1 + sqrt(1 - Prho\*\*2)\*Wtilda
117. # simulating Yt values and St values
118. temp = 0
119. Y = np.zeros(0)
120. # getting initial Y\_0, which is little y = -1
121. Y = np.append(Y, Py)
122. St = np.zeros(0)
123. St = np.append(St, s)
124. # loop for Yt and St simulation
125. # NOTE, Yt follows CIR process form
126. **while** temp < 500:
127. Y = np.append(Y, max(Y[temp]
128. + Plam\*(Pkappa - Y[temp])\*(T/500.)
129. + Pxi\*sqrt(Y[temp])\*(W2[temp+1] - W2[temp]), 0))
130. St = np.append(St, St[temp]
131. + r\*St[temp]\*(T/500.)
132. + sqrt(Y[temp])\*St[temp]\*(W1[temp+1] - W1[temp])
133. + (0.5 \* sqrt(Y[temp])\*sqrt(Y[temp])\*St[temp]\*((W1[temp+1] - W1[temp])\*\*2 - (T/500.))))
135. temp += 1
136. #
137. ST = St[St.size - 1]
139. # calculating payoff
140. #print("K size is: ")
141. #print(K.size)
142. payment = payoffEC(ST, K[i])
143. payoffsArray = np.append(payoffsArray, payment)
145. num += 1
146. # calculating price
147. EstimatedPrice = exp(-r\*T)\*np.mean(payoffsArray)
148. **print**("estimated price is: " + str(EstimatedPrice))
149. CK\_estimatedPrices = np.append(CK\_estimatedPrices, EstimatedPrice)
150. i += 1
151. **return** CK\_estimatedPrices

154. # finally, we do curve\_fit optimization
155. popt, pcov = optimize.curve\_fit(PricingUsingModelEC, CK, CK\_actualPrices, x0)
156. **print**(popt)