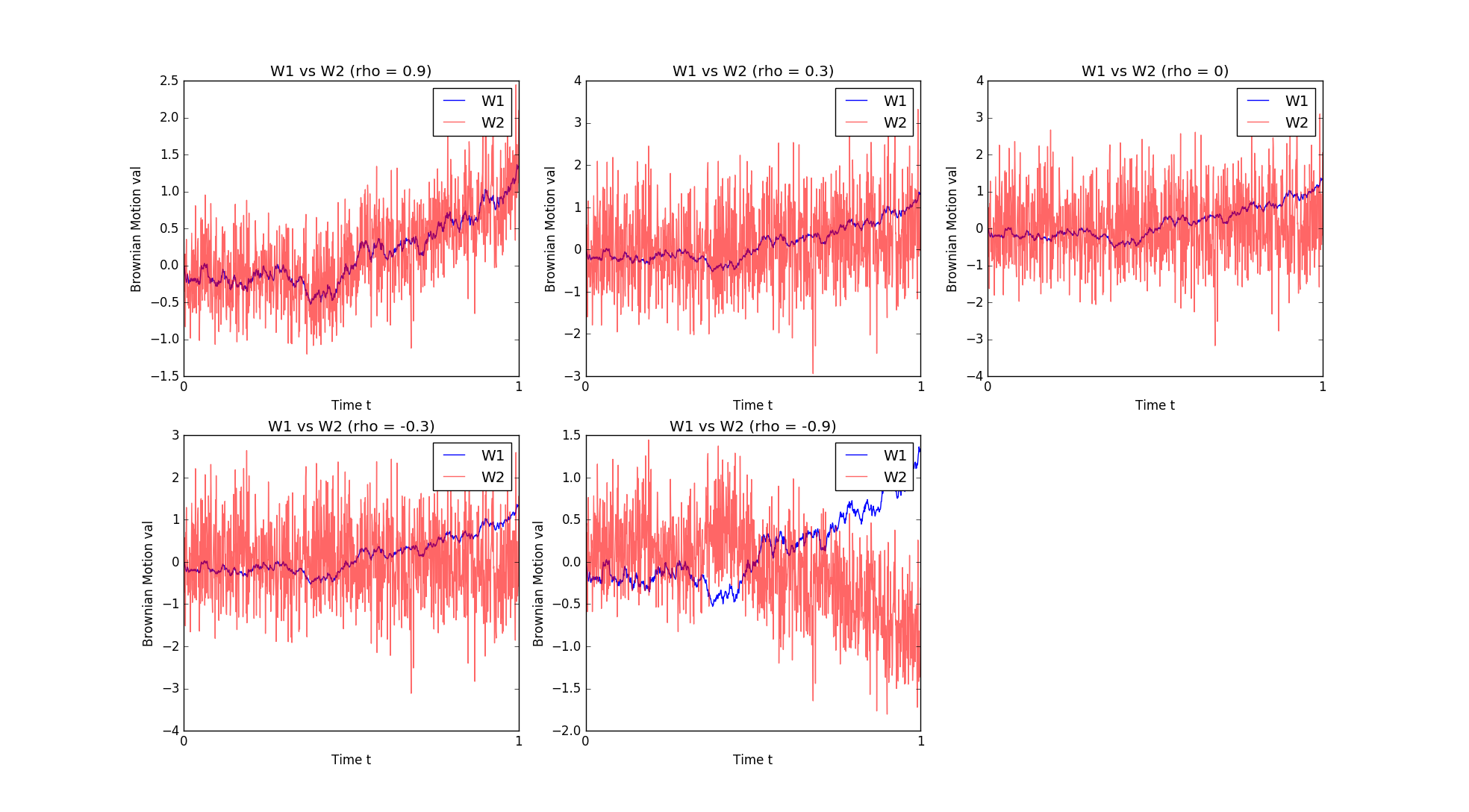
Question 1) c)

1. # importing the necessary packages
2. **import** numpy as np
3. **from** numpy **import** sqrt
4. **import** matplotlib.pyplot as plt
6. ##############################################################################
8. # increment step method for Brownian Motion
9. **def** SimBMStep(T, N):
10. # initialize W brownian motion array
11. W = np.zeros(int(N))
12. W = np.append(W, 0)    # initial 0 value
14. # loop
15. num = 0
16. **while** num < N:
17. W[num + 1] = W[num] + sqrt(T/N)\*np.random.standard\_normal()
18. num += 1

21. **return** W
23. # initial vars
24. rho1 = 0.9
25. rho2 = 0.3
26. rho3 = 0
27. rho4 = -0.3
28. rho5 = -0.9
29. # setting up t x-axis variable
30. t = np.arange(0.0, 1001.0, 1.0)
32. W1 = SimBMStep(1, 1000.)
33. Z = np.random.standard\_normal(1001)
35. plt.clf()
37. ###################################
38. plt.subplot(2,3,1)
40. # W2 part i
41. W2\_i = rho1\*W1 + sqrt(1 - rho1\*\*2)\*Z
43. plt.plot(t,W1, color = 'blue', label = 'W1')
44. plt.plot(t,W2\_i, color = 'red', alpha = 0.6, label = 'W2')
45. plt.title("W1 vs W2 (rho = 0.9)")
47. # this is to label the x-axis as 0 to 1 time t
48. ticks = np.arange(t.min(), t.max() + 1, 1000)
49. labels = range(ticks.size)
50. plt.xticks(ticks, labels)
51. plt.xlabel("Time t")
52. plt.ylabel("Brownian Motion val")
53. plt.legend()
55. ###################################
56. plt.subplot(2,3,2)
58. # W2 part ii
59. W2\_ii = rho2\*W1 + sqrt(1 - rho2\*\*2)\*Z
61. plt.plot(t,W1, color = 'blue', label = 'W1')
62. plt.plot(t,W2\_ii, color = 'red', alpha = 0.6, label = 'W2')
63. plt.title("W1 vs W2 (rho = 0.3)")
65. # this is to label the x-axis as 0 to 1 time t
66. ticks = np.arange(t.min(), t.max() + 1, 1000)
67. labels = range(ticks.size)
68. plt.xticks(ticks, labels)
69. plt.xlabel("Time t")
70. plt.ylabel("Brownian Motion val")
71. plt.legend()
73. ###################################
74. plt.subplot(2,3,3)
76. # W2 part iii
77. W2\_iii = rho3\*W1 + sqrt(1 - rho3\*\*2)\*Z
79. plt.plot(t,W1, color = 'blue', label = 'W1')
80. plt.plot(t,W2\_iii, color = 'red', alpha = 0.6, label = 'W2')
81. plt.title("W1 vs W2 (rho = 0)")
83. # this is to label the x-axis as 0 to 1 time t
84. ticks = np.arange(t.min(), t.max() + 1, 1000)
85. labels = range(ticks.size)
86. plt.xticks(ticks, labels)
87. plt.xlabel("Time t")
88. plt.ylabel("Brownian Motion val")
89. plt.legend()
91. ###################################
92. plt.subplot(2,3,4)
94. # W2 part iv
95. W2\_iv = rho4\*W1 + sqrt(1 - rho4\*\*2)\*Z
97. plt.plot(t,W1, color = 'blue', label = 'W1')
98. plt.plot(t,W2\_iv, color = 'red', alpha = 0.6, label = 'W2')
99. plt.title("W1 vs W2 (rho = -0.3)")
101. # this is to label the x-axis as 0 to 1 time t
102. ticks = np.arange(t.min(), t.max() + 1, 1000)
103. labels = range(ticks.size)
104. plt.xticks(ticks, labels)
105. plt.xlabel("Time t")
106. plt.ylabel("Brownian Motion val")
107. plt.legend()
109. ###################################
110. plt.subplot(2,3,5)
112. # W2 part v
113. W2\_v = rho5\*W1 + sqrt(1 - rho5\*\*2)\*Z
115. plt.plot(t,W1, color = 'blue', label = 'W1')
116. plt.plot(t,W2\_v, color = 'red', alpha = 0.6, label = 'W2')
117. plt.title("W1 vs W2 (rho = -0.9)")
119. # this is to label the x-axis as 0 to 1 time t
120. ticks = np.arange(t.min(), t.max() + 1, 1000)
121. labels = range(ticks.size)
122. plt.xticks(ticks, labels)
123. plt.xlabel("Time t")
124. plt.ylabel("Brownian Motion val")
125. plt.legend()

**Figure:**



Question 2)

1. # importing the necessary packages
2. **import** numpy as np
3. **from** numpy **import** sqrt, exp
4. **import** matplotlib.pyplot as plt
6. ##############################################################################
8. # increment step method for Brownian Motion
9. **def** SimBMStep(T, N):
10. # initialize W brownian motion array
11. W = np.zeros(int(N))
12. W = np.append(W, 0)    # initial 0 value
14. # loop
15. num = 0
16. **while** num < N:
17. W[num + 1] = W[num] + sqrt(T/N)\*np.random.standard\_normal()
18. num += 1
20. **return** W
22. # initial vars
23. s = 100.
24. y = -1
25. lam = 5
26. kappa = -1.5
27. rho = -0.2
28. xi = 0.25
29. r = 0.03
30. T = 0.3
31. runs = 1000
33. # payoff function for Lookback Put option
34. **def** payoffLP(Smax, ST):
35. **return** max(Smax - ST, 0)
37. # array holding payoffs for price calculation
38. payoffsArray = np.zeros(0)
40. num = 0
41. **while** num < runs:
42. # get W1 and W~ Brownian Motions, in order to get the W2 BM
43. # note: need to use rho correlation equation
44. W1 = SimBMStep(T, 500.)
45. Wtilda = SimBMStep(T, 500.)
46. W2 = rho\*W1 + sqrt(1 - rho\*\*2)\*Wtilda
48. # simulating Yt values
49. temp = 0
50. Y = np.zeros(0)
51. # getting initial Y\_0, which is little y = -1
52. Y = np.append(Y, y)
53. # loop for Yt simulation
54. **while** temp < 500:
55. Y = np.append(Y, Y[temp] + lam\*(kappa - Y[temp])\*(T/500.) + xi\*(W2[temp+1] - W2[temp]))
56. temp += 1
58. # St simulation (MILSTEIN SCHEME)
59. # derivative of exp(Yt)\*St is just exp(Yt)
60. temp = 0
61. St = np.zeros(0)
62. St = np.append(St, s)
63. **while** temp < 500:
64. St = np.append(St, St[temp]
65. + r\*St[temp]\*(T/500.)
66. + exp(Y[temp])\*St[temp]\*(W1[temp+1] - W1[temp])
67. + (0.5 \* exp(Y[temp])\*exp(Y[temp])\*St[temp]\*((W1[temp+1] - W1[temp])\*\*2 - (T/500.))))
68. temp += 1
70. # getting Smax from simulated St's, and ST terminal value
71. Smax = max(St)
72. ST = St[St.size - 1]
74. # calculating payoff
75. payoffsArray = np.append(payoffsArray, payoffLP(Smax, ST))
77. num += 1
78. # calculating price
79. PriceLP = exp(-r\*T)\*np.mean(payoffsArray)
80. **print**("The price of the lookback put option is: ")
81. **print**(round(PriceLP,2))

**Output:**

The price of the lookback put option is:

12.43

Question 3) a) and b)

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. # 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 = 100.
25. sigma = 0.2
26. r = 0.03
27. T = 1.
28. runs = 1000
29. K = 85
30. beta = 0
31. **print**("Beta is " + str(beta))
33. # payoff function for European call option
34. **def** payoffEC(ST, K):
35. **return** max(ST - K, 0)
37. # defined function to get implied volatility
38. **def** ImplVol(MarketPrice, sigma\_test):
39. **while** sigma\_test < 2.0:
40. d1 = (log(s/K) + (r + (sigma\_test\*\*2)/2)\*T) / (sigma\_test \* sqrt(T))
41. d2 = d1 - sigma\_test\*sqrt(T)
42. # black scholes formula price calculation
43. PriceBS = s\*norm.cdf(d1) - K\*exp(-r\*T)\*norm.cdf(d2)
44. # check if the two prices are close to each other
45. **if** abs(PriceBS - MarketPrice) < 0.001:
46. **return** sigma\_test
47. # else, do recursion of volatility calculation
48. **else**:
49. vega = s\*sqrt(T)\*norm.pdf(d1)
50. sigma\_test = sigma\_test - (PriceBS - MarketPrice)/vega
52. # big loop to print out various K values
53. i = 0
54. **while** i < 9:
56. # array holding payoffs for price calculation
57. payoffsArray = np.zeros(0)
59. num = 0
60. **while** num < runs:
61. # get W1 BM
62. W1 = SimBMStep(T, 500.)
64. # St simulation
65. temp = 0
66. St = np.zeros(0)
67. St = np.append(St, s)
68. **while** temp < 500:
69. St = np.append(St, St[temp]
70. + r\*St[temp]\*(T/500.)
71. + sigma\*(St[temp]\*\*beta)\*St[temp]\*(W1[temp+1] - W1[temp]))
72. temp += 1
73. # terminal ST
74. ST = St[St.size - 1]
75. # calculating payoff
76. payoffsArray = np.append(payoffsArray, payoffEC(ST, K))
77. num += 1
79. # calculating price
80. PriceECmarket = exp(-r\*T)\*np.mean(payoffsArray)
81. **print**("K = " + str(K) + ", beta = " + str(beta) + ", Euro Call Price = " + str(round(PriceECmarket,2)))
83. # assume volatility is 1.0, in order to get correct implied volatility
84. ImpliedVolatility = ImplVol(PriceECmarket, 1.0)
85. **print**("Implied volatility = " + str(ImpliedVolatility))

88. K += 5
89. i += 1

**Output:**

K = 85, beta = 0, Euro Call Price = 18.78

Implied volatility = 0.184003503105

K = 90, beta = 0, Euro Call Price = 16.13

Implied volatility = 0.223356059083

K = 95, beta = 0, Euro Call Price = 12.75

Implied volatility = 0.216199749266

K = 100, beta = 0, Euro Call Price = 9.42

Implied volatility = 0.200207290968

K = 105, beta = 0, Euro Call Price = 7.28

Implied volatility = 0.203812957256

K = 110, beta = 0, Euro Call Price = 4.78

Implied volatility = 0.186659084449

K = 115, beta = 0, Euro Call Price = 4.31

Implied volatility = 0.212430179682

K = 120, beta = 0, Euro Call Price = 2.89

Implied volatility = 0.203695072607

K = 125, beta = 0, Euro Call Price = 1.68

Implied volatility = 0.189991788903

K = 85, beta = -0.3, Euro Call Price = 17.98

Implied volatility = 0.138511841585

K = 90, beta = -0.3, Euro Call Price = 13.27

Implied volatility = 0.113685727285

K = 95, beta = -0.3, Euro Call Price = 8.25

Implied volatility = 0.0716150836257

K = 100, beta = -0.3, Euro Call Price = 3.77

Implied volatility = 0.0494287139351

K = 105, beta = -0.3, Euro Call Price = 1.28

Implied volatility = 0.0520328059972

K = 110, beta = -0.3, Euro Call Price = 0.25

Implied volatility = None

K = 115, beta = -0.3, Euro Call Price = 0.02

Implied volatility = 0.0472742516509

K = 120, beta = -0.3, Euro Call Price = 0.0

Implied volatility = 0.0503292228721

K = 125, beta = -0.3, Euro Call Price = 0.0

Implied volatility = 0.056491997428

K = 85, beta = -0.5, Euro Call Price = 18.11

Implied volatility = 0.147353340733

K = 90, beta = -0.5, Euro Call Price = 13.12

Implied volatility = 0.105224876388

K = 95, beta = -0.5, Euro Call Price = 8.06

Implied volatility = 0.0614743128677

K = 100, beta = -0.5, Euro Call Price = 3.13

Implied volatility = 0.0269013914866

K = 105, beta = -0.5, Euro Call Price = 0.17

Implied volatility = 0.0191667371791

K = 110, beta = -0.5, Euro Call Price = 0.0

Implied volatility = 0.000293248345683

K = 115, beta = -0.5, Euro Call Price = 0.0

Implied volatility = 0.0333351987452

K = 120, beta = -0.5, Euro Call Price = 0.0

Implied volatility = 0.0470901696856

K = 125, beta = -0.5, Euro Call Price = 0.0

Implied volatility = 0.056491997428

K = 85, beta = -0.7, Euro Call Price = 18.02

Implied volatility = 0.14143418072

K = 90, beta = -0.7, Euro Call Price = 13.01

Implied volatility = 0.0979190199733

K = 95, beta = -0.7, Euro Call Price = 8.08

Implied volatility = 0.0622625724996

K = 100, beta = -0.7, Euro Call Price = 3.03

Implied volatility = 0.021075922155

K = 105, beta = -0.7, Euro Call Price = 0.0

Implied volatility = 0.00842065817146

K = 110, beta = -0.7, Euro Call Price = 0.0

Implied volatility = 0.000274416855772

K = 115, beta = -0.7, Euro Call Price = 0.0

Implied volatility = 0.0333351987452

K = 120, beta = -0.7, Euro Call Price = 0.0

Implied volatility = 0.0470901696856

K = 125, beta = -0.7, Euro Call Price = 0.0

Implied volatility = 0.056491997428

K = 85, beta = -1.0, Euro Call Price = 18.06

Implied volatility = 0.143754335366

K = 90, beta = -1.0, Euro Call Price = 13.04

Implied volatility = 0.100385165424

K = 95, beta = -1.0, Euro Call Price = 8.05

Implied volatility = 0.0609180707467

K = 100, beta = -1.0, Euro Call Price = 3.05

Implied volatility = 0.0223719089738

K = 105, beta = -1.0, Euro Call Price = 0.0

Implied volatility = 0.00670529407276

K = 110, beta = -1.0, Euro Call Price = 0.0

Implied volatility = 0.000274416855772

K = 115, beta = -1.0, Euro Call Price = 0.0

Implied volatility = 0.0333351987452

K = 120, beta = -1.0, Euro Call Price = 0.0

Implied volatility = 0.0470901696856

K = 125, beta = -1.0, Euro Call Price = 0.0

Implied volatility = 0.056491997428

Role of Beta on the volatility smile:

As Beta decreases into more and more negative, the volatility smile decreases.

We can see there are lower implied volatility values for lower beta values.

Question 4)

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. # 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 = 100.
25. y = 0.08
26. lam = 3
27. kappa = 0.1
28. rho = -0.8
29. xi = 0.1
30. r = 0.03
32. T = 1.0
33. runs = 1000
34. K = 85
36. # payoff function for European Call option
37. **def** payoffEC(ST, K):
38. **return** max(ST - K, 0)
40. # defined function to get implied volatility
41. **def** ImplVol(MarketPrice, sigma\_test):
42. **while** sigma\_test < 2.0:
43. d1 = (log(s/K) + (r + (sigma\_test\*\*2)/2)\*T) / (sigma\_test \* sqrt(T))
44. d2 = d1 - sigma\_test\*sqrt(T)
45. # black scholes formula price calculation
46. PriceBS = s\*norm.cdf(d1) - K\*exp(-r\*T)\*norm.cdf(d2)
47. # check if the two prices are close to each other
48. **if** abs(PriceBS - MarketPrice) < 0.001:
49. **return** sigma\_test
50. # else, do recursion of volatility calculation
51. **else**:
52. vega = s\*sqrt(T)\*norm.pdf(d1)
53. sigma\_test = sigma\_test - (PriceBS - MarketPrice)/vega
55. # big loop to get difference price values for different K values
56. i = 0
57. **while** i < 9:
58. # array holding payoffs for price calculation
59. payoffsArray = np.zeros(0)
61. num = 0
62. **while** num < runs:
63. # get W1 and W~ Brownian Motions, in order to get the W2 BM
64. # note: need to use rho correlation equation
65. W1 = SimBMStep(T, 500.)
66. Wtilda = SimBMStep(T, 500.)
67. W2 = rho\*W1 + sqrt(1 - rho\*\*2)\*Wtilda
69. # simulating Yt values
70. temp = 0
71. Y = np.zeros(0)
72. # getting initial Y\_0, which is little y = -1
73. Y = np.append(Y, y)
74. # loop for Yt simulation
75. # NOTE, Yt follows CIR process form
76. **while** temp < 500:
77. Y = np.append(Y, max(Y[temp]
78. + lam\*(kappa - Y[temp])\*(T/500.)
79. + xi\*sqrt(Y[temp])\*(W2[temp+1] - W2[temp]), 0))
80. temp += 1
82. # St simulation (MILSTEIN SCHEME)
83. temp = 0
84. St = np.zeros(0)
85. St = np.append(St, s)
86. **while** temp < 500:
87. St = np.append(St, St[temp]
88. + r\*St[temp]\*(T/500.)
89. + sqrt(Y[temp])\*St[temp]\*(W1[temp+1] - W1[temp])
90. + (0.5 \* sqrt(Y[temp])\*sqrt(Y[temp])\*St[temp]\*((W1[temp+1] - W1[temp])\*\*2 - (T/500.))))
91. temp += 1
93. ST = St[St.size - 1]
94. # calculating payoff
95. payoffsArray = np.append(payoffsArray, payoffEC(ST, K))
97. num += 1
98. # calculating price
99. PriceECmarket = exp(-r\*T)\*np.mean(payoffsArray)
100. **print**("K = " + str(K) + ", European Call price = " + str(round(PriceECmarket,2)))
101. # assume volatility is 1.0, in order to get correct implied volatility
102. ImpliedVolatility = ImplVol(PriceECmarket, 1.0)
103. **print**("Implied volatility = " + str(ImpliedVolatility))
105. K += 5
106. i += 1

**Output:**

K = 85, European Call price = 22.08

Implied volatility = 0.310180904529

K = 90, European Call price = 18.9

Implied volatility = 0.30869756875

K = 95, European Call price = 17.48

Implied volatility = 0.346619156161

K = 100, European Call price = 13.78

Implied volatility = 0.312728538256

K = 105, European Call price = 11.65

Implied volatility = 0.313429144511

K = 110, European Call price = 8.65

Implied volatility = 0.285122524159

K = 115, European Call price = 7.23

Implied volatility = 0.289347967669

K = 120, European Call price = 6.28

Implied volatility = 0.299679538249

K = 125, European Call price = 5.18

Implied volatility = 0.300697706649

When changing rho, the volatility smile seems to get flatter if there’s a large positive/negative correlation.

When changing xi, if xi is large then the volatility smile decreases gradually.

But if xi is small then the volatility isn’t really a “smile,” and it remains mostly around 0.3.

Question 5) c)

1. **import** numpy as np
2. **from** numpy **import** sqrt, exp, log
3. **import** matplotlib.pyplot as plt
5. ##############################################################################
7. # increment step method for Brownian Motion
8. **def** SimBMStep(T, N):
9. # initialize W brownian motion array
10. W = np.zeros(int(N))
11. W = np.append(W, 0)    # initial 0 value
13. # loop
14. num = 0
15. **while** num < N:
16. W[num + 1] = W[num] + sqrt(T/N)\*np.random.standard\_normal()
17. num += 1
19. **return** W
21. # initial vars
22. y = 0.08
23. lam = 3.
24. kappa = 0.1
25. xi = 0.25
27. T = 1
28. N = 100.
30. t = np.arange(0, 1.01, 0.01)
32. i = 0
33. **while** i < 10:
34. W1 = SimBMStep(T, N)
36. Y = np.zeros\_like(W1)
37. X = np.zeros\_like(W1)
38. Y[0] = y
39. X[0] = sqrt(y)
40. temp = 0
41. **while** temp < X.size-1:
42. X[temp + 1] = (xi\*(W1[temp+1] - W1[temp])
43. + sqrt(xi\*\*2 \* (W1[temp+1] - W1[temp])\*\*2
44. + 4\*(1+lam\*(T/N))\*(X[temp]\*\*2 + (T/N)\*(lam\*kappa - (xi\*\*2/2)))))/(2\*(1+lam\*(T/N)))
45. Y[temp + 1] = X[temp+1]\*\*2
46. temp += 1
47. plt.plot(t, Y)
48. i += 1
50. plt.title("Simulated CIR Yt paths")
51. plt.xlabel("Time t")
52. plt.ylabel("Process value")
53. plt.show()

**Figure:**

(I assumed initial values for y, xi, kappa, and lambda are same as previous question)

