Question 1)

1. # importing the necessary packages
2. **import** numpy as np
3. **from** numpy **import** exp, log, sqrt, pi
5. ##############################################################################
7. # initial var
8. N = 100000
10. # defining our g(X) function, aka Y = indicator of (X >= a)
11. **def** funcY(x, a):
12. **if** x >= a:
13. **return** 1
14. **else**:
15. **return** 0
17. # N sample points of X and Y rv
18. X\_vals = np.random.standard\_normal(N)
20. # let a = 3
21. a\_1 = 3
23. Y\_vals1 = []
24. **for** i,X\_value **in** enumerate(X\_vals):
25. Y\_vals1.append(funcY(X\_value, a\_1))
27. # b\_optimal value which was calculated from Q1 part a
28. b\_optimal\_1 = 1/sqrt(2\*pi)\*exp(-a\_1\*\*2 /2.)
30. # new sampling, using the control variate
31. # note, E{X] = G is always 0, because X is standard normal rv.
32. ControlVariateEstimator1 = Y\_vals1 - b\_optimal\_1 \* (X\_vals - 0)
34. ValueEstimation\_1 = np.mean(ControlVariateEstimator1)
36. **print**("The estimation value (with a = 3) is: ")
37. **print**(ValueEstimation\_1)
39. #####################################
40. # let a = 8
41. a\_2 = 8
42. Y\_vals2 = []
43. **for** i,X\_value **in** enumerate(X\_vals):
44. Y\_vals2.append(funcY(X\_value, a\_2))
46. # b\_optimal value which was calculated from Q1 part a
47. b\_optimal\_2 = 1/sqrt(2\*pi)\*exp(-a\_2\*\*2 /2.)
49. # new sampling, using the control variate
50. # note, E{X] = G is always 0, because X is standard normal rv.
51. ControlVariateEstimator2 = Y\_vals2 - b\_optimal\_2 \* (X\_vals - 0)
53. ValueEstimation\_2 = np.mean(ControlVariateEstimator2)
55. **print**("The estimation value (with a = 8) is: ")
56. **print**(ValueEstimation\_2)

**Output:**

The estimation value (with a = 3) is:

0.001264311662

The estimation value (with a = 8) is:

4.91525955201e-18

Question 2)

1. # importing the necessary packages
2. **import** numpy as np
3. **from** numpy **import** exp, log, sqrt, pi
4. **from** scipy.stats **import** norm
6. ##############################################################################
8. # initial var
9. N = 100000
11. # defining our g(X) function, aka Y = indicator of (X >= a)
12. **def** funcY(x, a):
13. **if** x >= a:
14. **return** 1
15. **else**:
16. **return** 0
18. # N sample points of X and Y rv
19. X\_vals = np.random.standard\_normal(N)
21. # let a = 3
22. a\_1 = 3
24. Y\_vals1 = []
25. **for** i,X\_value **in** enumerate(X\_vals):
26. Y\_vals1.append(funcY(X\_value, a\_1))
28. # mu\_optimal was calculated in Question 2, part a
29. dens\_1 = 1/sqrt(2\*pi) \* exp( -2\*(a\_1\*\*2))
30. mu\_optimal\_1 = a\_1 - (2\*a\_1\*(1 - norm.cdf(2\*a\_1)) - dens\_1)/((4\*a\_1\*\*2 + 2)\*(1 - norm.cdf(2\*a\_1)) - 2\*a\_1\*dens\_1)
32. # shift the X values, because now we're in new measure P\_~
33. X\_vals\_shifted1 = np.random.normal(mu\_optimal\_1, 1, N)
35. # calculate f(X)/g(X) ratio
36. ratio = exp(-mu\_optimal\_1\*X\_vals\_shifted1 + (mu\_optimal\_1\*\*2)/2.)
38. # calculate Importance Sampling estimator
39. IS\_estimator1 = Y\_vals1\*ratio
41. ValueEstimation\_1 = np.mean(IS\_estimator1)
43. **print**("The estimation value (with a = 3) is: ")
44. **print**(ValueEstimation\_1)

47. #####################################
48. # let a = 3
49. a\_2 = 8
50. Y\_vals2 = []
51. **for** i,X\_value **in** enumerate(X\_vals):
52. Y\_vals2.append(funcY(X\_value, a\_2))
54. # mu\_optimal was calculated in Question 2, part a
55. dens\_2 = 1/sqrt(2\*pi) \* exp( -2\*(a\_2\*\*2))
56. mu\_optimal\_2 = a\_2 - (2\*a\_2\*(1 - norm.cdf(2\*a\_2)) - dens\_2)/((4\*a\_2\*\*2 + 2)\*(1 - norm.cdf(2\*a\_2)) - 2\*a\_2\*dens\_2)
58. # shift the X values, because now we're in new measure P\_~
59. X\_vals\_shifted2 = np.random.normal(mu\_optimal\_2, 1, N)
61. # calculate f(X)/g(X) ratio
62. ratio = exp(-mu\_optimal\_2\*X\_vals\_shifted2 + (mu\_optimal\_2\*\*2)/2.)
64. # calculate Importance Sampling estimator
65. IS\_estimator2 = Y\_vals2\*ratio
67. ValueEstimation\_2 = np.mean(IS\_estimator2)
69. **print**("The estimation value (with a = 8) is: ")
70. **print**(ValueEstimation\_2)

**Output:**

The estimation value (with a = 3) is:

0.000761871079619

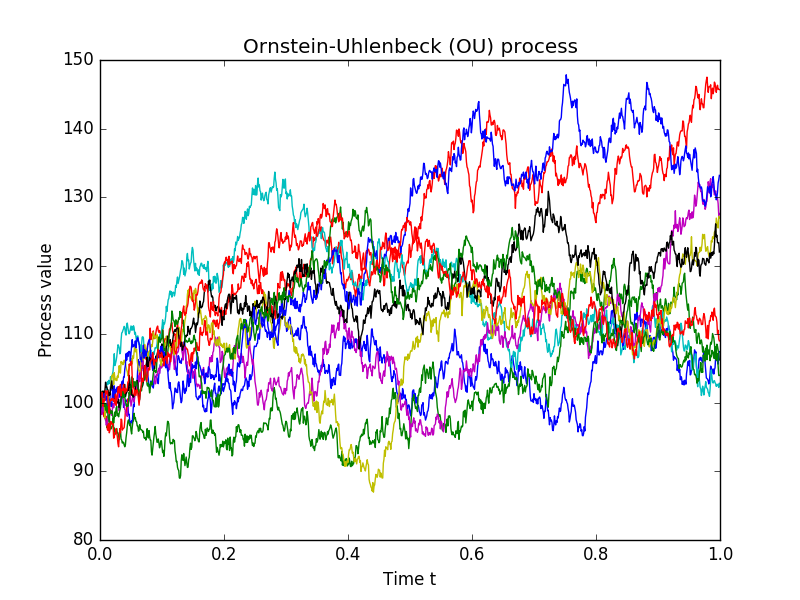
The estimation value (with a = 8) is:

0.0

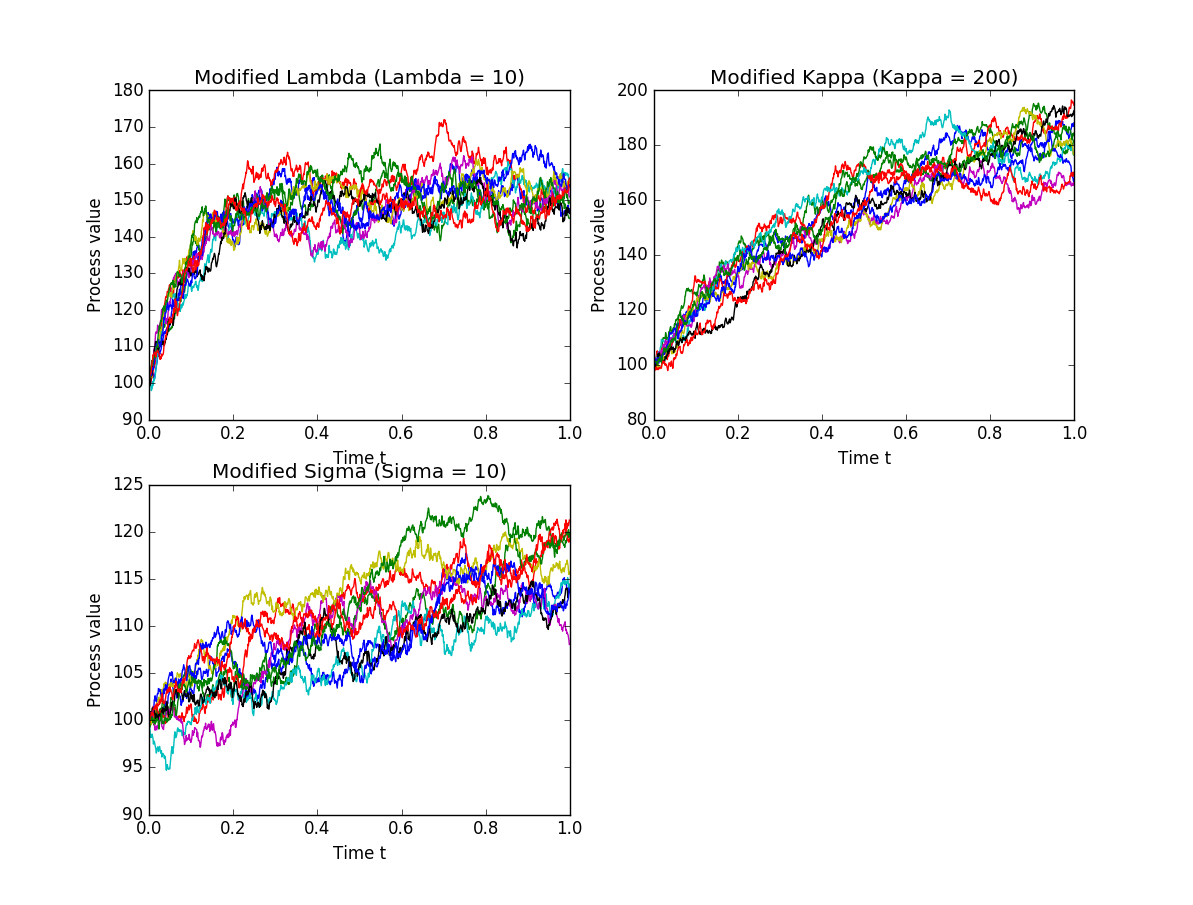
Question 3)

1. # importing the necessary packages
2. **import** numpy as np
3. **from** numpy **import** exp, log, sqrt
4. **import** matplotlib.pyplot as plt
6. ##############################################################################
8. # Ornstein-Uhlenbeck (OU) process
10. # initial vars
11. lam = 2.
12. kappa = 120.
13. sigma = 25.
14. X\_0 = 100.
16. N = 1000.
17. T = 1
18. dt = T/N
20. # t variable for x-axis plotting
21. t = np.arange(0, 1, 0.001)
23. num = 0
24. **while** num < 10:
25. X\_t = np.array([X\_0])
27. i = 0
28. **while** (i+1) < N:
29. Z = np.random.standard\_normal()
30. X\_t = np.append(X\_t,
31. X\_t[i] + lam\*(kappa - X\_t[i])\*dt + sigma\*sqrt(dt)\*Z)
32. i += 1
34. plt.plot(t, X\_t)
35. num += 1
37. plt.title("Ornstein-Uhlenbeck (OU) process")
38. plt.xlabel("Time t")
39. plt.ylabel("Process value")
41. plt.show()
43. ####################################
44. # changed parameter plots
46. lam\_edit = 10.
47. kappa\_edit = 200.
48. sigma\_edit = 10.
50. plt.subplot(2,2,1)
51. num = 0
52. **while** num < 10:
53. X\_t = np.array([X\_0])
55. i = 0
56. **while** (i+1) < N:
57. Z = np.random.standard\_normal()
58. X\_t = np.append(X\_t,
59. X\_t[i] + lam\_edit\*(150 - X\_t[i])\*dt + sigma\*sqrt(dt)\*Z)
60. i += 1
62. plt.plot(t, X\_t)
63. num += 1
65. plt.title("Modified Lambda (Lambda = 10)")
66. plt.xlabel("Time t")
67. plt.ylabel("Process value")
69. plt.subplot(2,2,2)
70. num = 0
71. **while** num < 10:
72. X\_t = np.array([X\_0])
74. i = 0
75. **while** (i+1) < N:
76. Z = np.random.standard\_normal()
77. X\_t = np.append(X\_t,
78. X\_t[i] + lam\*(kappa\_edit - X\_t[i])\*dt + sigma\*sqrt(dt)\*Z)
79. i += 1
81. plt.plot(t, X\_t)
82. num += 1
84. plt.title("Modified Kappa (Kappa = 200)")
85. plt.xlabel("Time t")
86. plt.ylabel("Process value")
88. plt.subplot(2,2,3)
89. num = 0
90. **while** num < 10:
91. X\_t = np.array([X\_0])
93. i = 0
94. **while** (i+1) < N:
95. Z = np.random.standard\_normal()
96. X\_t = np.append(X\_t,
97. X\_t[i] + lam\*(kappa - X\_t[i])\*dt + sigma\_edit\*sqrt(dt)\*Z)
98. i += 1
100. plt.plot(t, X\_t)
101. num += 1
103. plt.title("Modified Sigma (Sigma = 10)")
104. plt.xlabel("Time t")
105. plt.ylabel("Process value")
107. plt.show()

**Figure (a):**



**Figure (b):**



Each parameter of the Ornstein-Uhlenbeck (OU) process has the following definition:

* (lambda) is the speed of the mean-reversion.
* (kappa) is the mean/average that the process Xt tends towards.
* (sigma) is the process volatility, so a lower number means there’s less randomness.

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
6. ##############################################################################
7. # interest rate swap
9. # initial vars
10. lam = 0.7
11. kappa = 0.05
12. sigma = 0.006
13. r\_0 = 0.02
14. r\_fix = 0.04
15. N\_notional = 10000
17. N = 2000.
18. T = 2.
19. dt = T/N
21. runs = 1000
23. # interest rate follows Ornstein-Uhlenbeck dynamics
24. r\_T = np.zeros(0)
26. num = 0
27. **while** num < runs:
29. # loop for r\_t process
30. r\_t = np.array([r\_0])
31. i = 0
33. **while** (i+1) < N:
34. Z = np.random.standard\_normal()
35. r\_t = np.append(r\_t, r\_t[i] + lam\*(kappa - r\_t[i])\*dt + sigma\*sqrt(dt)\*Z)
36. i += 1
38. # integral r\_s calculation
39. r\_T = np.append(r\_T, sum(r\_t\*dt))
41. num += 1
43. payoffs = N\_notional \* (exp(r\_T) - exp(r\_fix \* T))
45. PriceIntSwap = np.mean(payoffs)
47. **print**("Expected payoff of the interest rate swap at maturity T = 2 is: ")
48. **print**(round(PriceIntSwap,2))

**Output:**

Expected payoff of the interest rate swap at maturity T = 3 is:

-80.96

After trial and error, in order for the Int Rate swap to trade at par at T = 2, the rfix interest rate would have to be ~3.39%.