Question 1)

```
# importing the necessary packages
import numpy as np
from numpy import exp, log, sqrt, pi

#################################################################################

# initial var
N = 100000

# defining our g(X) function, aka Y = indicator of (X >= a)
def funcY(x, a):
    if x >= a:
        return 1
    else:
        return 0

# N sample points of X and Y rv
X_vals = np.random.standard_normal(N)

# let a = 3
a_1 = 3

Y_vals1 = []
for i,X_value in enumerate(X_vals):
    Y_vals1.append(funcY(X_value, a_1))

# b_optimal value which was calculated from Q1 part a
b_optimal_1 = 1/sqrt(2*pi)*exp(-a_1**2 /2.)

# new sampling, using the control variate
# note, E{X] = G is always 0, because X is standard normal rv.
ControlVariateEstimator1 = Y_vals1 - b_optimal_1 * (X_vals - 0)

ValueEstimation_1 = np.mean(ControlVariateEstimator1)

print("The estimation value (with a = 3) is: ")
print(ValueEstimation_1)

#####################################
# let a = 8
a_2 = 8
Y_vals2 = []
for i,X_value in enumerate(X_vals):
    Y_vals2.append(funcY(X_value, a_2))

# b_optimal value which was calculated from Q1 part a
b_optimal_2 = 1/sqrt(2*pi)*exp(-a_2**2 /2.)

# new sampling, using the control variate
# note, E{X] = G is always 0, because X is standard normal rv.
ControlVariateEstimator2 = Y_vals2 - b_optimal_2 * (X_vals - 0)

ValueEstimation_2 = np.mean(ControlVariateEstimator2)

print("The estimation value (with a = 8) is: ")
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)

```
# importing the necessary packages
import numpy as np
from numpy import exp, log, sqrt, pi
from scipy.stats import norm

################################################################################

# initial var
N = 100000

# defining our g(X) function, aka Y = indicator of (X >= a)
def funcY(x, a):
    if x >= a:
        return 1
    else:
        return 0

# N sample points of X and Y rv
X_vals = np.random.standard_normal(N)

# let a = 3
a_1 = 3

Y_vals1 = []
for i,X_value in enumerate(X_vals):
    Y_vals1.append(funcY(X_value, a_1))

# mu_optimal was calculated in Question 2, part a
dens_1 = 1/sqrt(2*pi) * exp( -2*(a_1**2))
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)

# shift the X values, because now we're in new measure P_~
X_vals_shifted1 = np.random.normal(mu_optimal_1, 1, N)

# calculate f(X)/g(X) ratio
ratio = exp(-mu_optimal_1*X_vals_shifted1 + (mu_optimal_1**2)/2.)

# calculate Importance Sampling estimator
IS_estimator1 = Y_vals1*ratio

ValueEstimation_1 = np.mean(IS_estimator1)

print("The estimation value (with a = 3) is: ")
print(ValueEstimation_1)


#####################################
# let a = 3
a_2 = 8
Y_vals2 = []
for i,X_value in enumerate(X_vals):
    Y_vals2.append(funcY(X_value, a_2))

# mu_optimal was calculated in Question 2, part a
dens_2 = 1/sqrt(2*pi) * exp( -2*(a_2**2))
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)

# shift the X values, because now we're in new measure P_~
```

```
59. X_vals_shifted2 = np.random.normal(mu_optimal_2, 1, N)
60.
61. # calculate f(X)/g(X) ratio
62. ratio = exp(-mu_optimal_2*X_vals_shifted2 + (mu_optimal_2**2)/2.)
63.
64. # calculate Importance Sampling estimator
65. IS_estimator2 = Y_vals2*ratio
66.
67. ValueEstimation_2 = np.mean(IS_estimator2)
68.
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)

```
# importing the necessary packages
import numpy as np
from numpy import exp, log, sqrt
import matplotlib.pyplot as plt

################################################################################

# Ornstein-Uhlenbeck (OU) process

# initial vars
lam = 2.
kappa = 120.
sigma = 25.
X_0 = 100.

N = 1000.
T = 1
dt = T/N

# t variable for x-axis plotting
t = np.arange(0, 1, 0.001)

num = 0
while num < 10:
    X_t = np.array([X_0])

    i = 0
    while (i+1) < N:
        Z = np.random.standard_normal()
        X_t = np.append(X_t,
                        X_t[i] + lam*(kappa - X_t[i])*dt + sigma*sqrt(dt)*Z)
        i += 1

    plt.plot(t, X_t)
    num += 1

plt.title("Ornstein-Uhlenbeck (OU) process")
plt.xlabel("Time t")
plt.ylabel("Process value")

plt.show()

####################################
# changed parameter plots

lam_edit = 10.
kappa_edit = 200.
sigma_edit = 10.

plt.subplot(2,2,1)
num = 0
while num < 10:
    X_t = np.array([X_0])

    i = 0
    while (i+1) < N:
        Z = np.random.standard_normal()
        X_t = np.append(X_t,
                        X_t[i] + lam_edit*(150 - X_t[i])*dt + sigma*sqrt(dt)*Z)
        i += 1
```

```
61.
62.     plt.plot(t, X_t)
63.     num += 1
64.
65. plt.title("Modified Lambda (Lambda = 10)")
66. plt.xlabel("Time t")
67. plt.ylabel("Process value")
68.
69. plt.subplot(2,2,2)
70. num = 0
71. while num < 10:
72.     X_t = np.array([X_0])
73.
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
80.
81.     plt.plot(t, X_t)
82.     num += 1
83.
84. plt.title("Modified Kappa (Kappa = 200)")
85. plt.xlabel("Time t")
86. plt.ylabel("Process value")
87.
88. plt.subplot(2,2,3)
89. num = 0
90. while num < 10:
91.     X_t = np.array([X_0])
92.
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
99.
100.            plt.plot(t, X_t)
101.            num += 1
102.
103.        plt.title("Modified Sigma (Sigma = 10)")
104.        plt.xlabel("Time t")
105.        plt.ylabel("Process value")
106.
107.        plt.show()
```

Figure (a):

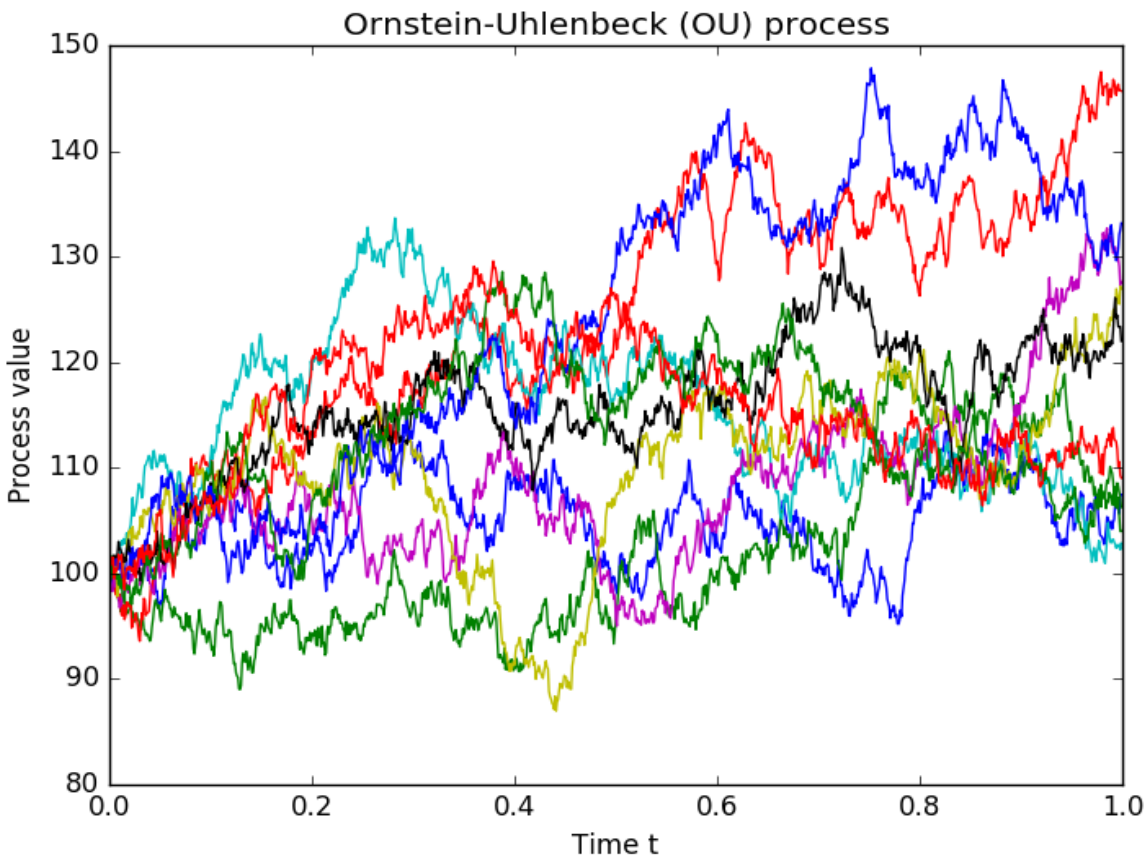
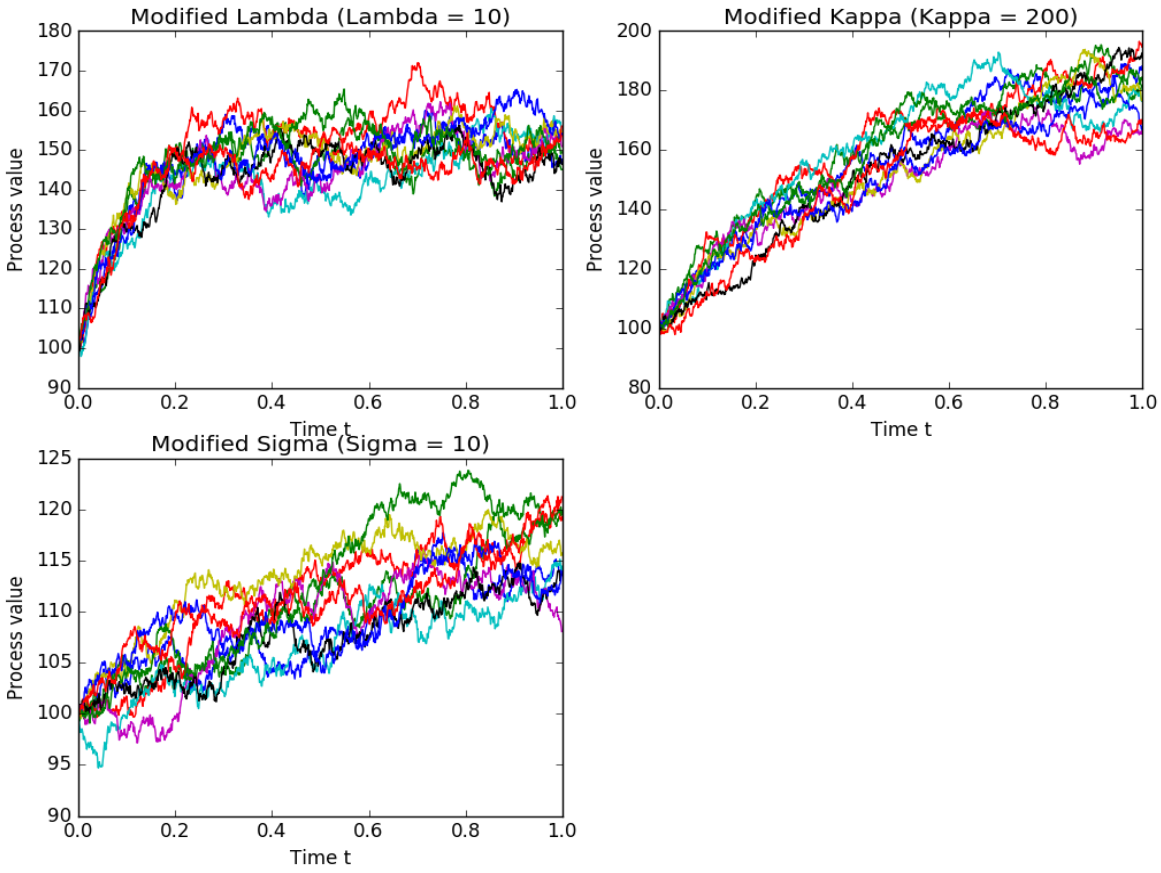

Ornstein-Uhlenbeck (OU) process
Process value
Time t
150
140
130
120
110
100
90
80
0.0
0.2
0.4
0.6
0.8
1.0

Figure (b):



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

- λ (lambda) is the speed of the mean-reversion.
- κ (kappa) is the mean/average that the process X_t tends towards.
- σ (sigma) is the process volatility, so a lower number means there's less randomness.

Question 4)

```
# importing the necessary packages
import numpy as np
from numpy import exp, log, sqrt
import matplotlib.pyplot as plt

################################################################################
# interest rate swap

# initial vars
lam = 0.7
kappa = 0.05
sigma = 0.006
r_0 = 0.02
r_fix = 0.04
N_notional = 10000

N = 2000.
T = 2.
dt = T/N

runs = 1000

# interest rate follows Ornstein-Uhlenbeck dynamics
r_T = np.zeros(0)

num = 0
while num < runs:

    # loop for r_t process
    r_t = np.array([r_0])
    i = 0

    while (i+1) < N:
        Z = np.random.standard_normal()
        r_t = np.append(r_t, r_t[i] + lam*(kappa - r_t[i])*dt + sigma*sqrt(dt)*Z)
        i += 1

    # integral r_s calculation
    r_T = np.append(r_T, sum(r_t*dt))

    num += 1

payoffs = N_notional * (exp(r_T) - exp(r_fix * T))

PriceIntSwap = np.mean(payoffs)

print("Expected payoff of the interest rate swap at maturity T = 2 is: ")
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 r_{fix} interest rate would have to be ~3.39%.