Project 6

Wenjun Li

March 24, 2020

Question 1

Generate 1000 samples of the random variable $A = X + Y$ where $X\ N(1,4)$ and $Y\ N(2,9)$. Use the Box-Muller method to generate independent random samples from the component normal distributions. Use every sample generated by the Box Muller method (do not throw one of the pair away). Estimate the covariance between your 1000 samples. Generate a histogram for Overlay the theoretical p.d.f. the histogram. Calculate the sample mean and sample variance for your samples and compare you estimates with the theoretical values.

The Polar Marsaglia method is an alternate method to generate samples from normal random variables. The method works by choosing a random point in the square -1<x<1 and -1<y<1 until $s = x^2 + y^2 < 1$ and then returning $R_1 = x\sqrt{\frac{-2\ln s}{s}}$ and $R_2 = y\sqrt{\frac{-2\ln s}{s}}$. Simulate 1,000,000 pairs of independent samples from a standard normal random variable using the Polar Marsaglia method. Compute the sample mean, sample variance, and covariance between the paired random samples. Repeat the experiment many times and compare the computational time required to generate 1,000,000 pairs of independent samples using the Polar Marsaglia method and the Box-Muller method.

Results and Analysis

In this question, we are required to simulate sampling from Normal distribution. Before we start, we have to realize the properties of Normal distribution, which will facilitate our simulation. Note that if Z N(0, 1) then X := µ + Z N(μ, σ^2), so we need to first generate N(0, 1) random variables and then scale them into particular Normal distributions. There are two methods to generate random variables from Normal distributions: 1. Box-Muller method; 2. Marsaglia's polar method.

First, for the Box-Muller method, it takes u1 and u2, two independent uniformly distributed random variables on $Uni(0,1)$ and defines

$$\begin{aligned} X &= \sqrt{-2\log(u_1) * cos(2\pi u_1)} \\ Y &= \sqrt{-2\log(u_2) * sin(2\pi u_2)} \end{aligned} \tag{1}$$

It can be proved that X and Y are N(0, 1) random variables, and independent.

Second, for the Marsaglia's polar method, it also takes u1 and u2 from uniform distribution $Uni(1,1)$. Then it accept if $s = u_1^2 + u_2^2 < 1$, otherwise get new u_1, u_2. It defines

$$\begin{aligned} X &= \sqrt{(}2\frac{\log s}{s}) * u_1 \\ Y &= \sqrt{(}2\frac{\log s}{s}) * u_2 \end{aligned} \tag{2}$$

Again it can be proved that X and Y are independent N(0, 1) random variables.

In my simulation, I use Box-Muller and Marsaglia's separately to generate 1000 pairs of random variables from standard normal distribution. Then, I utilize the property of Normal distribution and scale them into proper distribution as follows:

$$X = 1 + 2 * N(0, 1), Y = 2 + 3 * N(0, 1)$$

Now, we have $X\ N(1, 4)$ and $Y\ N(2, 9)$. Therefore, we can directly get $A = X + Y$. The covariance between X and Y, sample mean and sample variance of A, and the histogram of A are given below.

Results of Box-Muller Method

Cov(x, y): [-0.0140]
Sample mean of A: 2.959
Sample variance of A: 13.040

Results of Marsaglia's Method

Cov(x, y): [0.040]
Sample mean of A: 3.010
Sample variance of A: 13.227

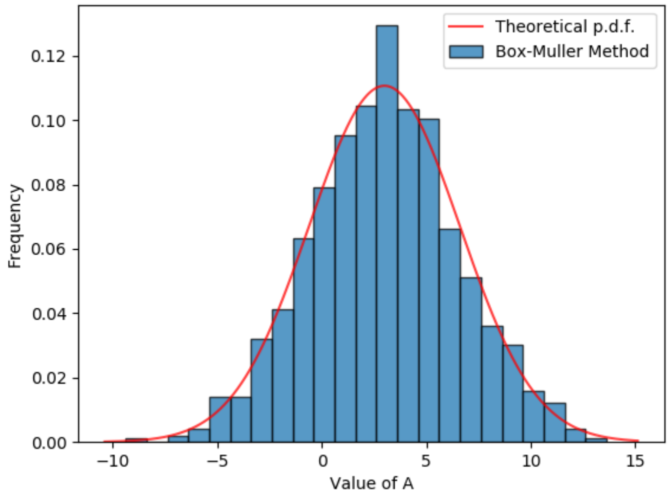


Figure 1: Histogram of A

From the above results we can say that our simulation of Box-Muller and Marsaglia's is successful, because the sample mean of A $\mu_A = \mu_X + \mu_Y = 1 + 2 = 3$, the sample variance of A

$Var_A = Var_X + Var_Y = 4 + 9 = 13$, and the histogram of A is very close to the theoretical normal distribution. Also, the covariance between X and Y is a value close to zero, from which we can conclude that X and Y are independent. Thus, we prove the correctness of Box-Muller method and Marsaglia's method.

Next, I will compare the computational time needed to generate 1,000,000 pairs of normal random variable using Box-Muller and Marsaglia's. To eliminate irrelevant factors, I only consider three steps: generating two random normal variables, scaling them and summing them up to get A. The computational time results are as below:

Time Consumed for Box-Muller: 0.11 s
Time Consumed for Marsaglia's: 7 s

From this result, we can conclude that Box-Muller method has a much shorter computational time compared to Marsaglia's method. This is because there is a `accept−reject` step in Marsaglia's method, i.e. if $s \geq 1$ we will reject it and generate another pair of standard normal random variables.

Python Code

Listing 1: Code of Question 1

```
import numpy as np
import matplotlib.pyplot as plt
import time


# Box−Muller Method
# start_box = time.time()
N = 1000
X_mean = 1; X_var = 4
Y_mean = 2; Y_var = 9

u1 = np.random.rand(N, 1)
u2 = np.random.rand(N, 1)

# generate X and Y that are N(0,1) random variables and independent
X = np.sqrt(−2*np.log(u1)) * np.cos(2*np.pi*u2)
Y = np.sqrt(−2*np.log(u1)) * np.sin(2*np.pi*u2)

# scale them to particular mean and variance
x = X_mean + np.sqrt(X_var)*X       # x ~ N(X_mean, X_var)
y = Y_mean + np.sqrt(Y_var)*Y       # y ~ N(X_mean, X_var)
A = x + y

```

```
# print time
# end_box = time.time()


# compute cov(x, y), sample_mean, sample_variance
print('Results of Box-Muller Method')
print('Cov(x, y): {}'.format(sum((x - np.mean(x)) * (y - np.mean(y))) / N))
print('Sample mean of A: {}'.format(np.mean(A)))
print('Sample variance of A: {}\n'.format(np.var(A, ddof=1)))

# theoretical values and theoretical p.d.f
theo_pdf = []
miu = X_mean + Y_mean
delta_2 = X_var + Y_var

x_range = np.arange(min(A)-1, max(A)+1, 0.05)
for x in x_range:
    theo_pdf.append((np.exp(-(x-miu)**2 / (2*delta_2)) / np.sqrt(2*np.pi*delta_2))
        )


# histogram of A
plt.figure()
bins = np.arange(np.min(A), np.max(A), 1)
plt.hist(A, bins, edgecolor='black', alpha=0.75, density=1, label='Box-Muller
    Method')
plt.plot(x_range, theo_pdf, color='r', label='Theoretical p.d.f.', alpha=0.75)
plt.xlabel('Value of A')
plt.ylabel('Frequency')
plt.legend()
plt.savefig('Q1_hist.png')
plt.show()


# Marsaglia's Polar method
# start_mar = time.time()

# generate X and Y that are N(0, 1)
i = 0
while i <= 999:
    u1 = 2*np.random.rand() - 1
    u2 = 2*np.random.rand() - 1
    s = u1**2 + u2**2
    if s < 1:
        X[i] = np.sqrt(−2*np.log(s)/s)*u1
        Y[i] = np.sqrt(−2*np.log(s)/s)*u2
        i += 1

# scale them
x = X_mean + np.sqrt(X_var)*X       # x ~ N(X_mean, X_var)
y = Y_mean + np.sqrt(Y_var)*Y       # y ~ N(X_mean, X_var)
A = x + y

# print time
# end_mar = time.time()


# compute covariance, sample_mean, sample_variance
print('Results of Marsaglia\'s Method')
print('Cov(x, y): {}'.format(sum((x − np.mean(x)) * (y − np.mean(y))) / N))
print('Sample mean of A: {}'.format(np.mean(A)))
print('Sample variance of A: {}\n'.format(np.var(A, ddof=1)))


# Console Results
# Results of Box−Muller Method
# Cov(x, y): [−0.014009676]
# Sample mean of A: 2.959960372120547
# Sample variance of A: 13.040561584217165
#
# Results of Marsaglia's Method
# Cov(x, y): [0.04065821]
# Sample mean of A: 3.010860622723196
# Sample variance of A: 13.227198179944219


# compare time needed to generate 1,000,000 samples
# 1. Box−Muller Method
start_box = time.time()

for i in range(1000):
    X_mean = 1; X_var = 4
    Y_mean = 2; Y_var = 9

    u1 = np.random.rand(N, 1)
    u2 = np.random.rand(N, 1)

    X = np.sqrt(−2*np.log(u1)) * np.cos(2*np.pi*u2)
    Y = np.sqrt(−2*np.log(u1)) * np.sin(2*np.pi*u2)

    x = X_mean + np.sqrt(X_var)*X       # x ~ N(X_mean, X_var)
    y = Y_mean + np.sqrt(Y_var)*Y       # y ~ N(X_mean, X_var)
    A = x + y

# print time
end_box = time.time()
print('Time Consumed for Box−Muller: %.2f s' % (end_box − start_box))


# 2. Marsaglia's Polar method
start_mar = time.time()

for j in range(1000):
    i = 0
    while i <= 999:
        u1 = 2*np.random.rand() − 1
        u2 = 2*np.random.rand() − 1
        s = u1**2 + u2**2
        if s < 1:
            X[i] = np.sqrt(−2*np.log(s)/s)*u1
            Y[i] = np.sqrt(−2*np.log(s)/s)*u2
            i += 1

    x = X_mean + np.sqrt(X_var)*X       # x ~ N(X_mean, X_var)
    y = Y_mean + np.sqrt(Y_var)*Y       # y ~ N(X_mean, X_var)
    A = x + y

# print time
end_mar = time.time()
print('Time Consumed for Marsaglia\'s: %2.f s' % (end_mar − start_mar))


# Console Results
# Time Consumed for Box−Muller: 0.11 s
# Time Consumed for Marsaglia's:  7 s
```

Question 2

Consider sampling from a $Gamma(\theta, 1)$ random variable. If θ is an integer then you can perform the sampling by summing θ different $Exp(1)$ random variables. But if s not an integer then it's more difficult. Generate 1000 samples from $Gamma(5.5, 1)$ using an accept-reject method (i.e. do not use built-in functions to generate from the gamma distribution). Generate a histogram and overlay the theoretical p.d.f. the histogram. Comment on the acceptance rate and your overall fit.

Results and Analysis

In *Question*2 we need to simulate sampling from gamma distribution $Gamma(k, \theta)$. If k is integer, we cen directly perform the sampling by summing k different $Exp(1)$ random variables. But if k is not integer then it's more complex and we need to generate the samples using an accept-reject method. The gamma distribution has a p.d.f. as:

$$Gamma(k,\theta) = \frac{1}{\Gamma(k) * \theta^k} * x^{k-1} * e^{-\frac{x}{\theta}} \tag{3}$$

and it has some unique properties that will help to simulate:

$$\begin{aligned} \Gamma(\alpha) &= (\alpha-1)\Gamma(\alpha-1) \\ \Gamma(\frac{1}{2}) &= \sqrt{\pi} \end{aligned} \tag{4}$$

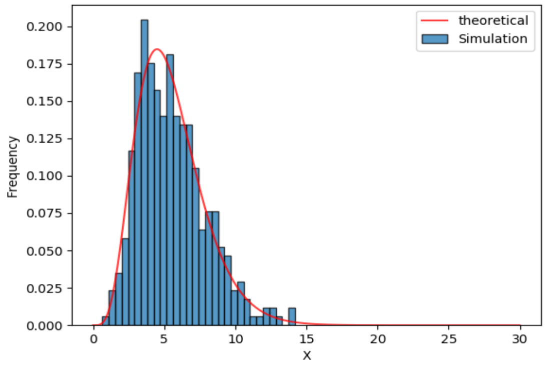
With the help of above p.d.f. equation and properties, we can perform the below deduction:

$$\begin{aligned} Gamma(k=5.5,\theta=1) &= \frac{1}{\Gamma(k) * \theta^k} * x^{k-1} * e^{-\frac{x}{\theta}} \\ &= \frac{1}{\Gamma(5.5) * 1^{5.5}} * x^{4.5} * e^{-\frac{x}{1}} \\ &= \frac{1}{\Gamma(5.5)} * x^{4.5} * e^{-x} \\ &= \frac{1}{4.5 * 3.5 * 2.5 * 1.5 * 0.5 * \Gamma(0.5)} * x^{4.5} * e^{-x} \\ &= \frac{1}{4.5 * 3.5 * 2.5 * 1.5 * 0.5 * \sqrt{\pi})} * x^{4.5} * e^{-x} \end{aligned} \tag{5}$$

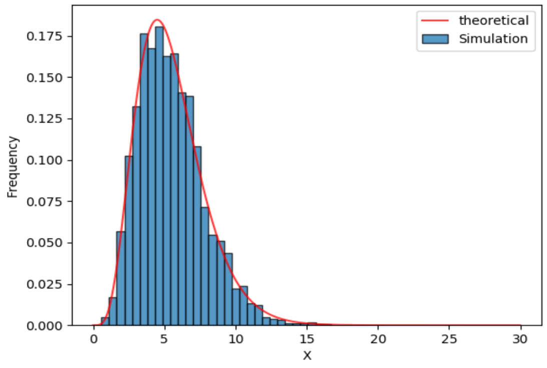
Except for the $f(x, k, \theta)$ function to generate gamma random variables, I also need a $g(y)$ function to generate Exponential random variables ($Exp(5.5)$). Thus, I can combine and use them to do 'accept-reject'.

In my simulation, I tried three different sample numbers, i.e. $N = 1000, 10000, 100000$. The plots in Figure 2. clearly show my simulation result. The acceptance rates of my simulations are:

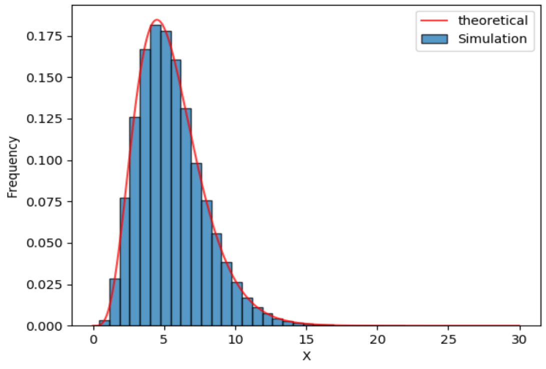
N = 1000, Acceptance Rate = 0.36
N = 10000, Acceptance Rate = 0.37
N = 100000, Acceptance Rate = 0.37



(a)



(b)



(c)

Figure 2: Generated $Gamma(5.5, 1)$ and Theoretical Distribution

From Figure 2., we can say our simulation is very close to the theoretical distribution and the overall fit is quite good. Also, the overall fit becomes better with an increasing sampling number N.

Python Code

Listing 2: Code of Question 2

```
import numpy as np
import matplotlib.pyplot as plt


N = 100000
k = 5.5
theta = 1
```

```


def f(x, k, theta):
    value = (2 / (np.sqrt(np.pi)*4.5*3.5*2.5*1.5)) * (1/(theta**k)) * (x**(k-1)) *
        np.exp(-x/theta)
    return value


def g(y):
    value = (1/k) * np.exp(-y/k)
    return value


result = []
theo = []
accept_num = 0

# use accept-reject to generate
for i in range(N):
    u = -k * np.log(np.random.uniform())
    if np.random.uniform() < f(u, k, theta)/(np.e*g(u)):
        result.append(u)
        accept_num += 1

# theoretical sequence
xrange = np.arange(0, 30, 0.01)
for y in xrange:
    theo.append(f(y, k, theta))


# result & plot
print('Acceptance Rate = %.2f' % (accept_num / N))
plt.figure()
plt.hist(result, bins=30, edgecolor='black', alpha=0.75, density=1, label='
    Simulation')
plt.plot(xrange, theo, color='r', alpha=0.75, label='theoretical')
plt.ylabel('Frequency')
plt.xlabel('X')
plt.legend()
plt.savefig('Q2_a')
plt.show()
```

Question 3

Thick-tailed alpha-stable pdfs have found many applications in physics and engineering where thicker tails can model energetic or impulsive processes. Alpha-stable pdfs are bell curves whose tails get thicker as the parameter α gets smaller for $\alpha \in (0, 2]$. The Gaussian pdf has the thinnest tails and corresponds to $\alpha = 2$. There are two main problems with using stable pdfs in physical models. The first is that only a few stable pdfs have a known closed form. These special cases include the symmetric alpha-stable pdfs of the Gaussian ($\alpha = 2$) and the Cauchy or Lorentzian ($\alpha = 1$) as well as the asymmetric Levy ($\alpha = 0.5$). An alpha-stable pdf f has characteristic function

$$\varphi(\omega) = expic\omega - \gamma|\omega|^{\alpha}(1 + i\beta sgn\omega)\Gamma)$$

where

$$\Gamma = \begin{cases} tan\dfrac{\alpha\pi}{x}, \alpha \neq 1 \\ -\dfrac{2}{\pi}ln|\omega|, \alpha = 1 \end{cases}$$

$i = \sqrt{-1}, 0 < \alpha \leq 2, -1 \leq \beta \leq 1$ and $\gamma > 0$. The parameter α is the characteristic exponent. It is a measure of the tail thickness for symmetric bell curves. The α controls the tail thickness of the resulting bell curve because the bell curve has thicker tails as α falls. The location parameter c is the median of a symmetric stable density. β is a skewness parameter. The density is symmetric about c if $\beta = 0$. The dispersion parameter γ acts like a variance because it controls the width of an alpha-stable bell curve even though such densities have no variance except in the Gaussian case of $\alpha = 2$.

The Chambers-Mallows-Stuck method describes how to generate samples from an arbitrary alpha stable distribution [1]. Use the Proposition 2.1 and Theorem 3.1 from the formulation by Weron [2] to sample from symmetric ($\beta = 0$) alpha stable pdfs using for four different values of α (α =0.5, 1, 1.8, 2.0).

For each value of α produce a histogram and a time series plot. Comment on the sample magnitude as a function of α. Use the Python function scipy.stats.levy_stable to overlay the corresponding theoretical alpha-stable pdf on your histogram. Repeat the above procedure assuming a right-skewed alpha stable distribution with $\beta = 0.75$.

Results and Analysis

The alpha-stable is a four-parameter family of distributions and is (usually) denoted by $S(\alpha, \beta, \gamma, \delta)$, where $\alpha \in (0, 2]$ is called the $characteristic exponent$ and describes the tail of the distribution, $\beta \in [-1, 1]$ is the $skewness$, γ and δ are similar to the variance and mean in the normal distribution. For example, if $Z\ S(\alpha, \beta, \gamma, \delta)$, then if $\alpha = 1$, $\gamma Z + \delta\ S(\alpha, \beta, \gamma, \delta)$.

Proposition 2.1

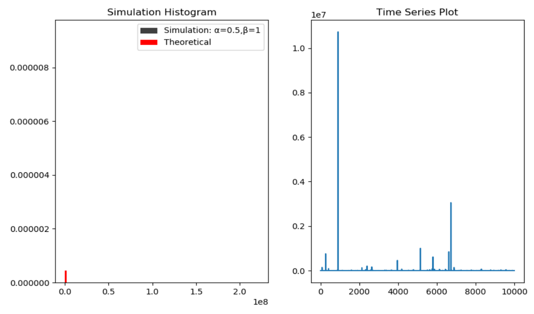
$$\gamma_0 = -\frac{\pi}{2}\beta_2\frac{K(\alpha)}{\alpha}, whereK(\alpha) = \begin{cases} \alpha, & \alpha < 1 \\ \alpha - 2, & \alpha > 1 \end{cases} \tag{6}$$

Theorem 3.1

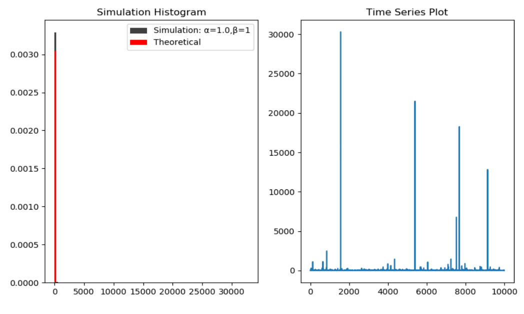
$$\begin{aligned} &for\alpha \neq 1 \qquad X = \frac{sin\alpha(\gamma - \gamma_0)}{(cos\gamma)^{1/\alpha}}\left(\frac{cos(\gamma - \alpha(\gamma - \gamma_0))}{W}\right)^{(1-\alpha)/\alpha} \\ &\text{for } \alpha = 1 \qquad X = (\tfrac{\pi}{2} + \beta_2\gamma)tan\gamma - \beta_2 log(\tfrac{Wcos\gamma}{\frac{\pi}{2}+\beta_2\gamma}) \end{aligned} \tag{7}$$

In this question, we need to perform eight pairs of α and β, i.e. $\alpha = [0.5, 1.0, 1.8, 2.0]$ and $\beta = [1, 0.75]$. Using **Proposition 2.1** ($Equation(2)$) and **Theorem 3.1** ($Equation(7)$), we can sample from symmetric ($\beta = 0$) alpha stable pdfs using four different values of $\alpha = [0.5, 1.0, 1.8, 2.0]$. Similarly, when $\beta \neq 0$, we can plug the β value in the above equations.

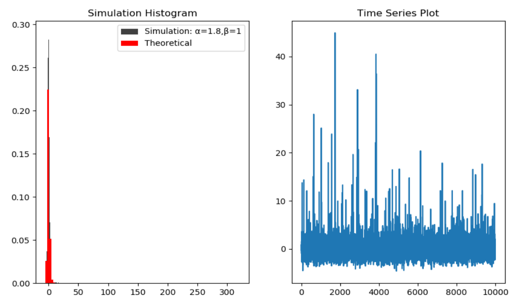
For each pair of α and β, I plot a histogram and a time series plot. The eights plots are displayed in Figure 3. (a), (b), (c) and (d) are plots of $\beta = 1$ and (e), (f), (g) and (h) are plots of $\beta = 0.75$. As we can see from Figure 3., when $\beta = 0$, the alpha-stable distributions are not skewed, while they are kind of right skewed when $\beta = 0.75$. Besides, we can see that the tail of the distribution becomes larger when α increases.



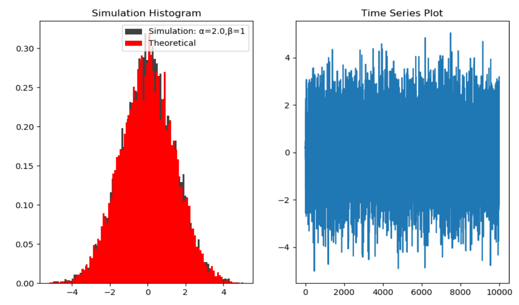
(a)



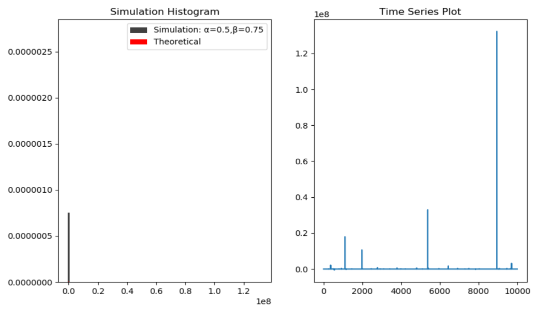
(b)



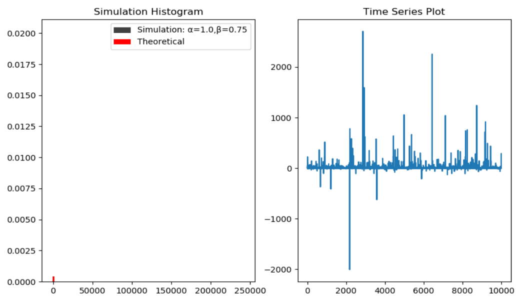
(c)



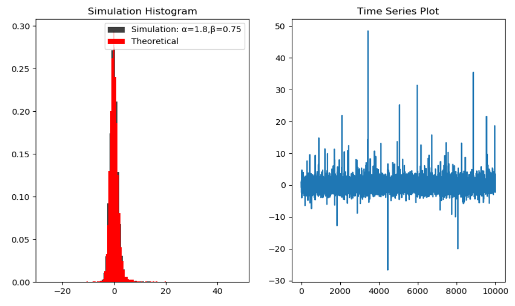
(d)



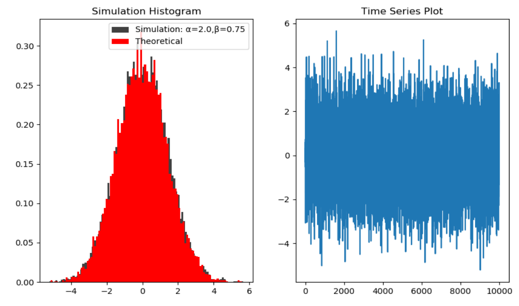
(e)



(f)



(g)



(h)

Figure 3: Different Pair

Python Code

Listing 3: Code of Question 3

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import levy_stable


N = 10000
alpha = [0.5, 1.0, 1.8, 2.0]        # alpha
beta = [0, 0.75]                    # beta


def K(alpha):
    if alpha < 1:
        return alpha
    elif alpha > 1:
        return alpha - 2


def f(W, alpha, beta):
    gamma = np.random.uniform(-np.pi/2, np.pi/2)
    if alpha == 1:
        return (np.pi/2 + beta*gamma) * np.tan(gamma) - beta * np.log((W*np.cos(gamma)) / (np.pi/2 + beta*gamma))

    else:
        gamma0 = (-np.pi / 2) * beta * (K(alpha) / alpha)
        return (np.sin(alpha*(gamma-gamma0)) / (np.cos(gamma)**(1/alpha))) * ((np.cos(gamma-alpha*(gamma-gamma0)) / W)**((1-alpha)/alpha))


for a in alpha:
    for b in beta:
        x = []
        for i in range(N):
            W = np.random.exponential(1)
            x.append(f(W, a, b))

        plt.figure(figsize=(10, 6))

        # alpha-stable simulations
        plt.subplot(1, 2, 1)
        plt.hist(x, bins=100, color='black', density=1, alpha=0.75, label='
            Simulation: '+chr(945)+'='+str(a)+','+chr(946)+'='+str(b))
        theo = levy_stable.rvs(a, b, size=N, scale=1)
        plt.hist(theo, bins=100, color='r', density=1, label='Theoretical')
        plt.title('Simulation Histogram')
        plt.legend()

        # plots of time series
        plt.subplot(1, 2, 2)
        plt.plot(np.arange(0, N), x)
        plt.title('Time Series Plot')

        plt.savefig('Q3_a{}_b{}.png'.format(a, b))
        plt.show()
```