Project 6

Qidi Han

March 25, 2020

Question 1

Generate 1000 samples of the random variable A=x+Y where X N(1,4) and Y N(2,9).....

Theory

In this question, I will use Box Muller method to generate the number, then i will estimate the co-variance between X and Y in the 1000 samples. Next, I will generate the histogram and overlay with the theoretical p.d.f in the histogram. For the X, Y and A, I will generate the mean and variance for each data. Finally, I will compare the answer with theoretical values. After the Box Muller method, I will use Polar Marsaglia Method, but this time, I will simulate 1000000 pairs of independent samples. Then I will compute the sample mean and sample variance and co-variance between both x and y. Repeat the 1000000 with Box Muller method, I will compare the computational times with both method.

Algorithm 1: Code for Monte Carlo simulation

```
/* Box-muller method */
u1 = np.random.rand(N,1)
u2 = np.random.rand(N,1)
e 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)
x = np.sqrt(V1)*X + M1; x N(M1,V1)
y = np.sqrt(V2)*Y + M2; y N(M2,V2)
/* Polar Marsaglia method */
while i<=999: u1 = 2*np.random.rand()-1
u2 = 2*np.random.rand()-1
s = u1*u1 + u2*u2
if s < 1:
X[i] = np.sqrt(-2*np.log(s)/s)*u1
Y[i] = np.sqrt(-2*np.log(s)/s)*u2
i = i+1
x = np.sqrt(V1)*X + M1; x N(M1,V1)
y = np.sqrt(V2)*Y + M2; y N(M2,V2)
```

```
import numpy as np
import time
import matplotlib.pyplot as plt
import math
import scipy.stats as stats

sum_x=[];
sum_y=[];
sum_a=[];
#     start_box = time.time()
N = 1000 #no of samples
M1 = 1 # Mean of X
M2 = 2 # Mean of Y
V1 = 4 # Variance of X
V2 = 9 # Variance of Y

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 a particular mean and variance
x = np.sqrt(V1)*X + M1; # x~ N(M1,V1)
y = np.sqrt(V2)*Y + M2; # y~ N(M2,V2)
A = x+y

for i in range(N):
    sum_x.append(float(x[i]))
    sum_y.append(float(y[i]))
    sum_a.append(float(A[i]))



covariance = np.cov(sum_x, sum_y)
plt.hist(sum_a,density=True)
xplot=np.linspace(-10,16,100)
yplot = stats.norm(3, math. asqrt(13)).pdf(xplot)
plt.plot(xplot,yplot,color='g')
print('Mean of A sampling:',str(np.mean(sum_a)),'Theoretical Mean of A:', M1+M2)
print('Variance of A sampling:',str(np.var(sum_a)),'Theoretical Variance of A:', V1+V2)
print('Mean of x sampling:',str(np.mean(sum_x)))
print('Variance of x sampling:',str(np.var(sum_x)))
print('Mean of y sampling:',str(np.mean(sum_y)))
print('Variance of y sampling:',str(np.var(sum_y)))
print(covariance)
```

Figure 1: Code for Box Muller method

```
Mean of A sampling: 2.9295845439069668 Theoretical Mean of A: 3
Variance of A sampling: 13.716296275331283 Theoretical Variance of A: 13
Mean of x sampling: 0.9138598786686696
Variance of x sampling: 4.1436102906668575
Mean of y sampling: 2.015724665238297
Variance of y sampling: 9.119561335778734
[[4.14775805 0.22678911]
 [0.22678911 9.12869003]]
```

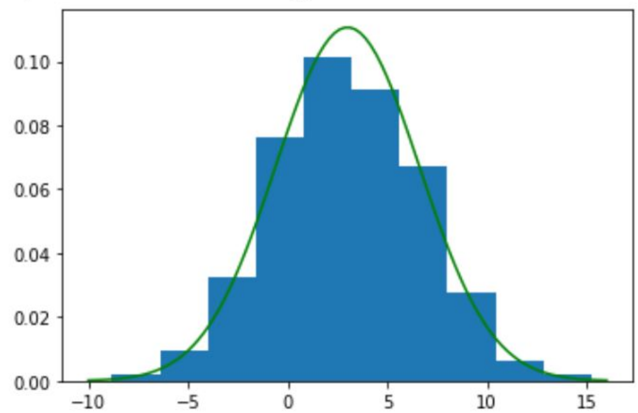


Figure 2: Covariance, Mean result and Histogram result for Box Muller method

2]:

```
import numpy as np
import time

# start_mar = time.time()
M1 = 1 # Mean of X
M2 = 2 # Mean of Y
V1 = 4 # Variance of X
V2 = 9 # Variance of Y
i = 0 # the random number generated by the algorithm

# Generate X and Y that are N(0,1) random variables and indepedent
X=[];
Y=[];

while i< 1000000:
    u1 = 2*np.random.rand()-1
    u2 = 2*np.random.rand()-1
    s = u1*u1 + u2*u2
    if s < 1:
        X.append(np.sqrt(-2*np.log(s)/s)*u1)
        Y.append(np.sqrt(-2*np.log(s)/s)*u2)
        i = i+1
# Scale them to a particular mean and variance

# x = np.sqrt(V1)*X + M1; # x~ N(M1,V1)
# y = np.sqrt(V2)*Y + M2; # y~ N(M2,V2)
i=0;
sum_x=[];
sum_y=[];
while i<=999:
    sum_x.append(np.sqrt(V1)*X[i] + M1);
    sum_y.append(np.sqrt(V2)*Y[i] + M2);
    i=i+1;

covariance = np.cov(sum_x, sum_y)
print('Mean of x sampling:',str(np.mean(sum_x)))
print('Variance of x sampling:',str(np.var(sum_x)))
print('Mean of y sampling:',str(np.mean(sum_y)))
print('Variance of y sampling:',str(np.var(sum_y)))
print(covariance)

# end_mar = time.time()
# print(end_mar - start_mar)
```

Figure 3: Code for Polar Marsaglia method

```
Mean of x sampling: 1.0961490065666553
Variance of x sampling: 4.210447915111898
Mean of y sampling: 2.2785549304320294
Variance of y sampling: 8.621437046235382
[[4.21466258 0.03944962]
 [0.03944962 8.63006711]]
```

```
[1]: import numpy as np
     import time
     import matplotlib.pyplot as plt
     import math
     import scipy.stats as stats
```

Figure 4: Covariance, Mean result and Histogram result for Polar Marsaglia method

```
start_box1 = time.time()
sum_x=[];
sum_y=[];
sum_a=[];
#     start_box = time.time()
N = 1000000 #no of samples
M1 = 1 # Mean of X
M2 = 2 # Mean of Y
V1 = 4 # Variance of X
V2 = 9 # Variance of Y
i=0;
X=[];
Y=[];
while i< N:
    u1 = np.random.rand()
    u2 = np.random.rand()

        #  Generate X and Y that are N(0,1) random variables and independent
    X.append(np.sqrt(-2*np.log(u1))*np.cos(2*np.pi*u2))
    Y.append(np.sqrt(-2*np.log(u1))*np.sin(2*np.pi*u2))
    i=i+1
i=0;
sum_x1=[]
sum_y2=[]
A=[]
while i<N:
    sum_x1.append(np.sqrt(V1)*X[i] + M1);
    sum_y2.append(np.sqrt(V2)*Y[i] + M2);
    A.append(sum_x1[i]+sum_y2[i])
    i=i+1;
end_box1 = time.time()
print(end_box1 - start_box1)


start_mar2 = time.time()
M1 = 1 # Mean of X
M2 = 2 # Mean of Y
V1 = 4 # Variance of X
V2 = 9 # Variance of Y
i = 0 # the random number generated by the algorithm

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

i=0;
sum_x=[];
sum_y=[];
while i<N:
    sum_x.append(np.sqrt(V1)*X[i] + M1);
    sum_y.append(np.sqrt(V2)*Y[i] + M2);
    i=i+1;
end_mar2 = time.time()
print(end_mar2 - start_mar2)
```

```
8.027211427688599
7.2661542892456055
```

Figure 5: Code and result for comparison

Explanation

For the Box Muller method, The covariance is around 0.22, it close to independent, the mean and variance for X,Y, and A are close to theoretical values. The result is shows on Figure 2. It shows the experimental result and theoretical result. For the Polar Marsaglia method, the covariance for X and Y is 0.03944, It's close to 0, therefore, we can tell that X and Y is independent. The mean and variance for X and Y is close to the theoretical result. For theoretical mean for x and y is 1 and 2, the variance for x and y is 4 and 9. The comparison of computational time between the Box Muller method and Polar Marsaglia method is close to each other, but from the experiment result on figure 5, Box-Muller is slower than Polar Marsaglis method. The computation time for Box-Muller method is 8.027211427688599. The computation time for Polar Marsaglia method is 7.2661542892456055. The different for both method is that Polar-Marsaglia method has a term of s = u1*u1 + u2*u2, but the result will not affect too much.

Question 2

Sampling from a Gamma random variable. Generate 1000 samples from Gamma(5.5,1) by using accept-reject method

Theory

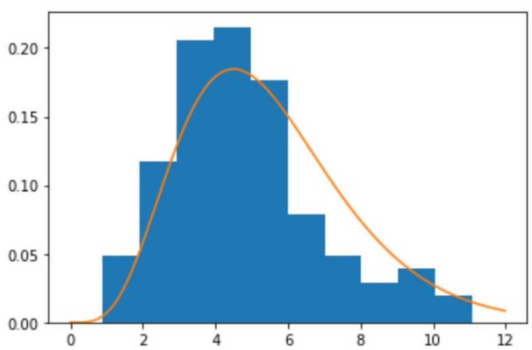
In this question, I will generate the theoretical value for pdfx and pdfy first, I will put the formula in the next section. Then I will calculate the maximal ratio for pdfx and pdfy. I will use f(x)/(c*g(x)) as the condition for determination for accept-reject method. As I generate the experimental value, I will generate the histogram and overlay the theoretical pdfx. I will output the acceptance rate and discuss the result in the explaination part.

Algorithm 2: Code and formula for Gamma random varible.

```
/* PDF for f(x) */
pdfX = lambda x: (1/((4.5*3.5*2.5*1.5*0.5)*np.sqrt(np.pi)))*(x**4.5)*(np.exp(-x))
/* PDF for g(x) */
pdfY = lambda y: (1/5.5)*(np.exp(-(1/5.5)*np.array(y)))  /* Compute the max ratio
    between f(x) and g(x) */
t = np.arange(0,8,0.01)
ratio = np.divide(pdfX(t),pdfY(t))
c = np.max(ratio)
/* accept-reject method */
while i < 1000:
u1 = np.random.rand();
y=-5.5*math.log10(u1);
count= count+1;
if np.random.rand()<((32*5.5/945/2.5)/np.sqrt(np.pi))*(y**4.5)*np.exp(-9*y/11):
x[i]=y
i=i+1;
if(i>=100):
break;
```

```
]: import numpy as np
import matplotlib.pyplot as plt
import math

pdfX = lambda x: (1/((4.5*3.5*2.5*1.5*0.5)*np.sqrt(np.pi)))*(x**4.5)*(np.exp(-x))
pdfY = lambda y: (1/5.5)*(np.exp(-(1/5.5)*np.array(y)))

# plt.plot(pdfY(t),label = 'pdf of Exponential(3/2)')
# plt.plot(pdfY(t)*c,label = 'c * pdf of Exponential(3/2)')

count=0;
x=[]
y=0;
x=np.zeros(100)
i=0;
while i < 1000:
    u1 = np.random.rand();
    y=-5.5*math.log10(u1);
    count= count+1;
    if np.random.rand()<((32*5.5/945/2.5)/np.sqrt(np.pi))*(y**4.5)*np.exp(-9*y/11):
        x[i]=y
        i=i+1;
        if(i>=100):
            break;
plt.hist(x,density=True)
t = np.arange(0,int(max(x))+1,0.01)

plt.plot(t,pdfX(t),label = 'pdf of Gamma(5.5,1)')
```

```
]: [<matplotlib.lines.Line2D at 0x21675d17e88>]
```




```
]: print('Acceptance rate:',str(i/count))
```

```
Acceptance rate: 0.36363636363636365
```

Figure 6: Code and Histogram and acceptance rate for this experiment

```python
import numpy as np
import matplotlib.pyplot as plt
pdfX = lambda x: (1/((4.5*3.5*2.5*1.5*0.5)*np.sqrt(np.pi)))*(x**4.5)*(np.exp(-x))
pdfY = lambda y: (1/5.5)*(np.exp(-(1/5.5)*np.array(y)))

t = np.arange(0,8,0.01)
ratio = np.divide(pdfX(t),pdfY(t))
c = np.max(ratio)
print(c)
fig = plt.figure(figsize=(8,6),dpi=100)
plt.plot(pdfX(t),label = 'pdf of Gamma(3/2,1)')
plt.plot(pdfY(t),label = 'pdf of Exponential(3/2)')
plt.plot(pdfY(t)*c,label = 'c * pdf of Exponential(3/2)')
plt.legend()
plt.show()
```

```
2.505029572089816
```

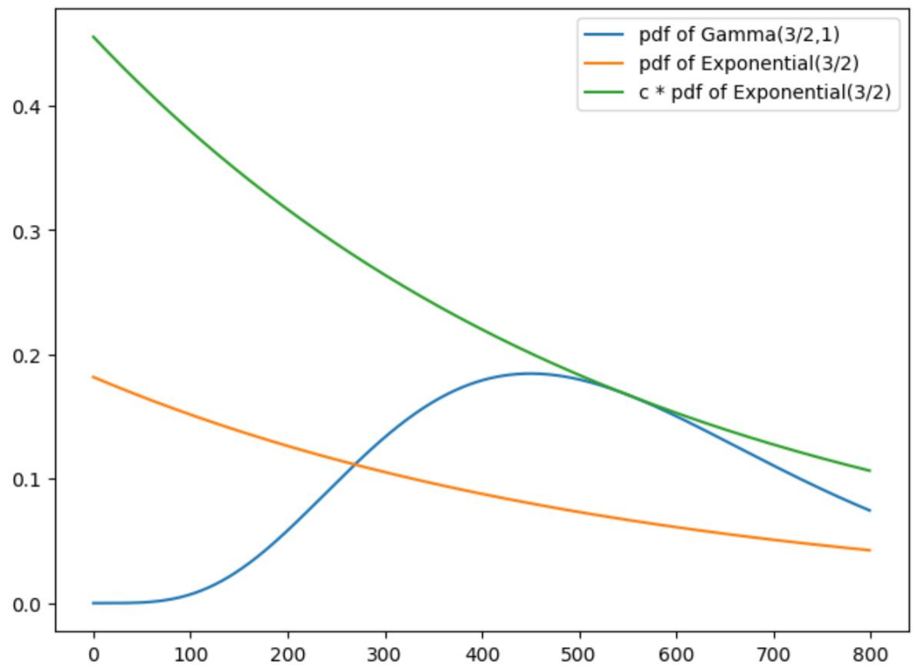


Figure 7: Code and result for compute c

Explanation

The experiment histogram is fit with theoretical p.d.f. The acceptance rate is 0.363636. The ration for pdf f(x) and pdf g(x) is reasonable.

Question 3

Thick-tailed alpha-stable PDF. I will use formulate to generate the experiment result and compare with the theoretical result

Theory

In this question, I will generate the Alpha-stable pdfs. We are going to use Chambers-Mallows-Stuck method to generate the samples from the an arbitrary alpha stable distribution. There are two important formula in this method, I will use python code to generate present in next section. I will plot the result and the time series, for the histogram result, I will overlay the theoretical pdf by using scipy.stats.levy.stable function. There are 8 result for our question. There are two beta, each beta will have 4 different alpha.

Algorithm 3: .

/* Formula for sample value */

1. Generate the W, PHi,k(a)

$$w = np.random.exponential(1) \tag{1}$$

$$k(a) = 1 - |1 - \alpha| \tag{2}$$

$$\Phi o = -\frac{1}{2}\pi(\frac{k(a)}{a}) \tag{3}$$

2. generate z

$$z = -\frac{\cos e\Phi - \tan\alpha\Phi o \sin e\Phi}{W\cos\Phi} \tag{4}$$

3. generate x

$$x = (-\frac{\sin\alpha\Phi}{\cos\Phi} - \tan\alpha\Phi o\frac{\cos\alpha\Phi}{\sin\Phi} - 1)z^{e/(1-e)} + \tan\alpha\Phi o(1 - z^{e/(1-e))} \tag{5}$$

/* Theoretical alpha-stable pdf */

x = np.linspace(levy$_s$$table.ppf(0.01, alpha, beta), levy_stable.ppf(0.99, alpha, beta), 100)$

plt.plot(x, levy$_s$$table.pdf(x, alpha, beta), 'r-', lw = 5, alpha = 0.6, label =' levy_stablepdf')$

```python
def pdf_theo2(a,beta):
    w=np.random.exponential(1)
    ka=1-abs(1-a)
    fa=(-1/2)*np.pi*beta*ka/a
    f=np.random.uniform(-(np.pi)/2,(np.pi)/2);
    e=1-a
    z=((np.cos(e*f)-(math.tan(a*fa)*np.sin(e*f)))/(w*np.cos(f)))
    s=(((np.sin(a*f)/np.cos(f))-(math.tan(a*fa)*((np.cos(a*f)/np.cos(f))-1)))*(z**(e/(1-e))))+(math.tan(a*fa)*(1-z**e/(1-e)))
    return s
```

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


N = 1000
alpha, beta = 2, 0.75
x1=np.linspace(levy_stable.ppf(0.01,alpha,beta),levy_stable.ppf(0.99,alpha,beta),N)
y1=[]
for j in range(N):
    y1.append(pdf_theo2(alpha, beta))

# r = levy_stable.rvs(alpha, beta, size=N, scale = 1)
plt.hist(y1,density='true')
plt.plot(x1, levy_stable.pdf(x1, alpha, beta),    'r-', lw=5, alpha=0.6, label='levy_stable pdf')
plt.show()
```

Figure 8: Code for this experiment and two equations

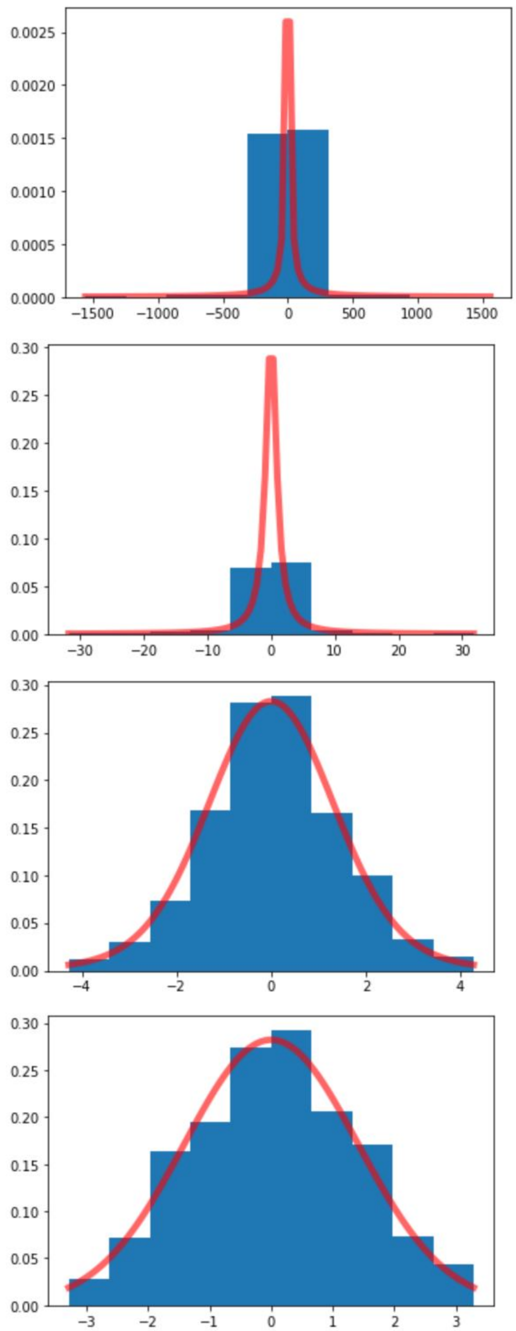


Figure 9: Result for histogram when beta=0

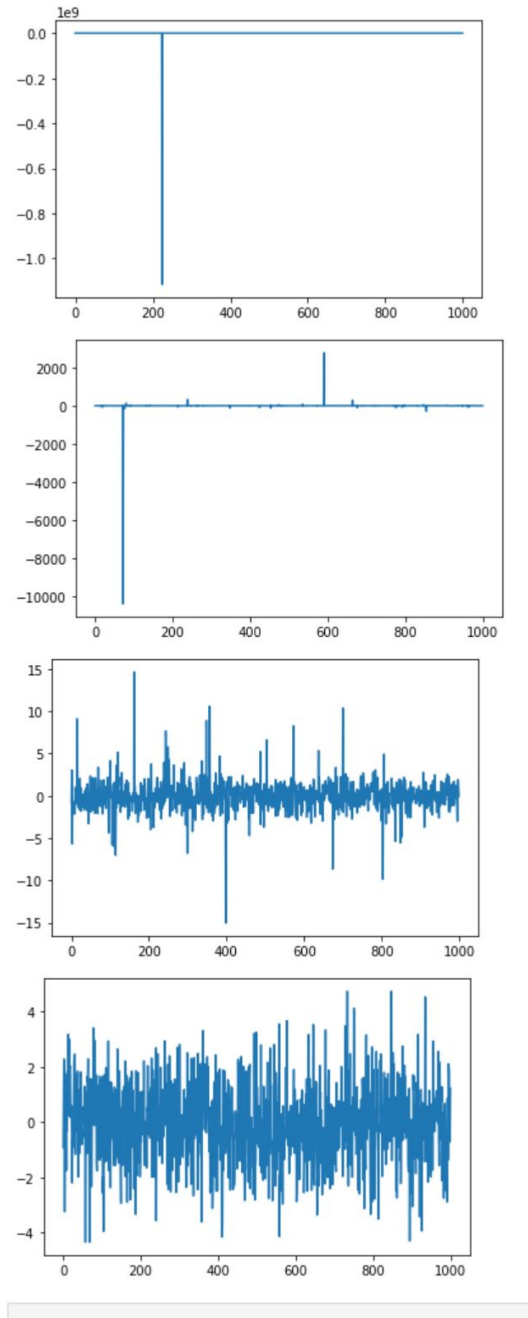


Figure 10: Result for time series when beta=0

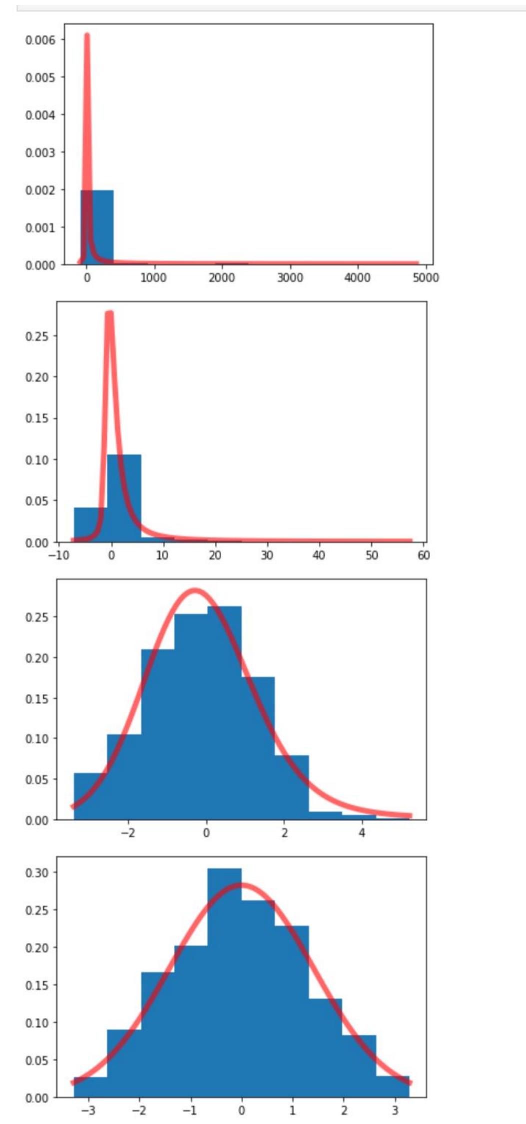


Figure 11: Result for histogram when beta=0.75

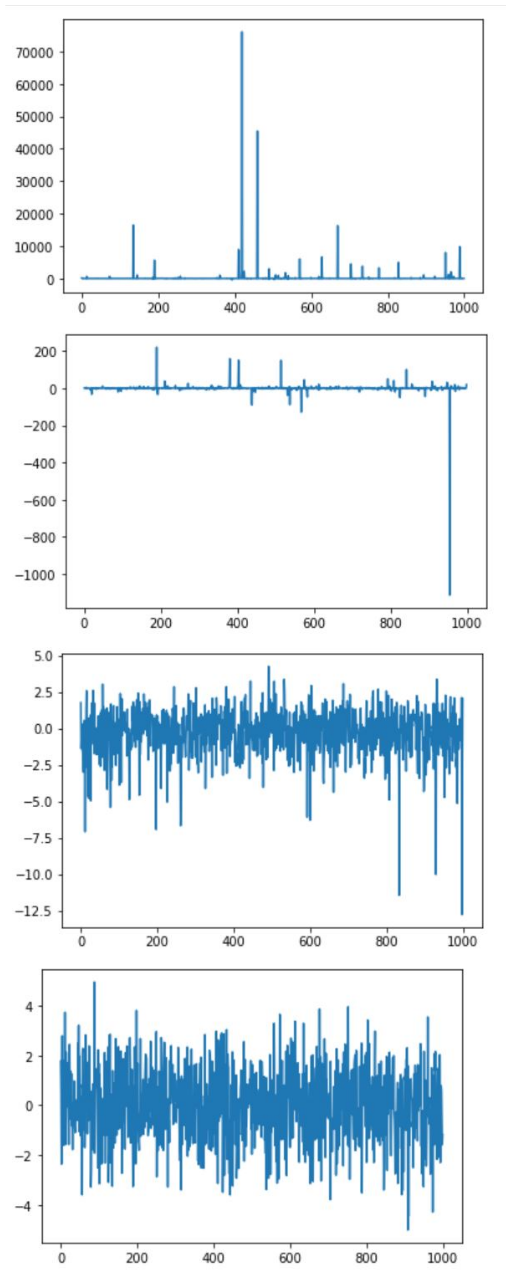


Figure 12: Result for time series when beta=0.75

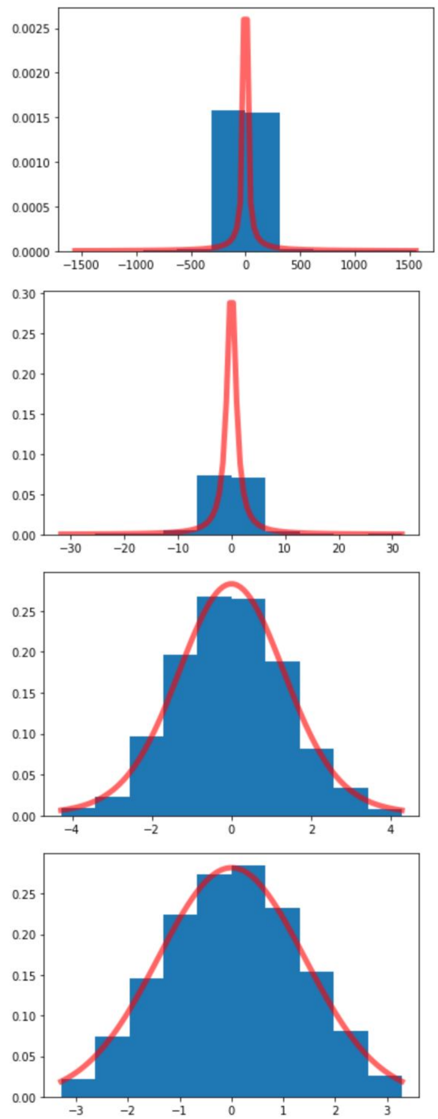


Figure 13: Built-in function Result for histogram when beta=0

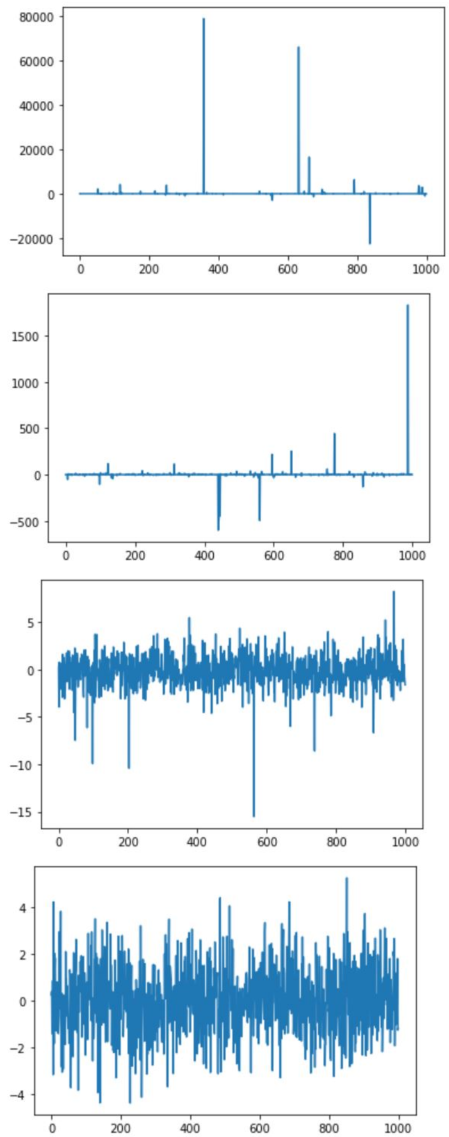


Figure 14: Built-in function Result for time series when beta=0

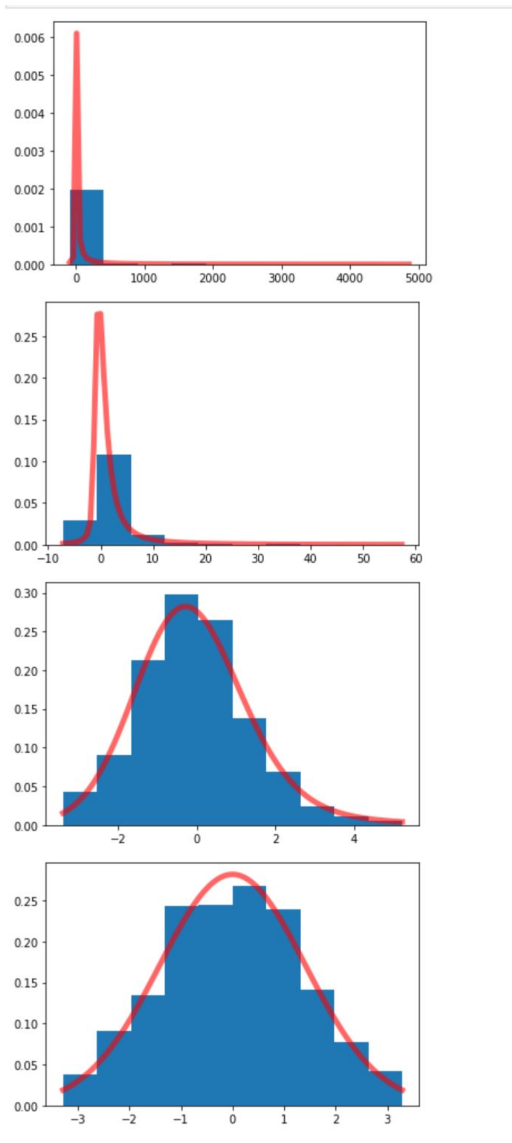


Figure 15: Built-in function Result for histogram when beta=0.75

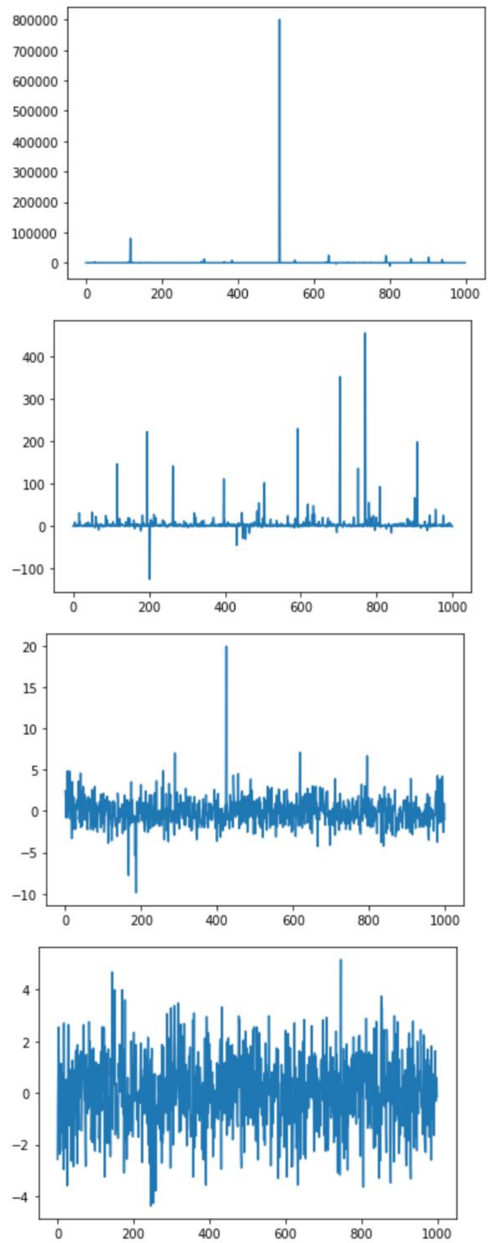


Figure 16: Built-in function Result for time series when beta=0.75

Explanation

From the experiment, the result is almost perfect. When I use rvs() function to generate the random samples, It will have the same reason. That means Chambers-Mallows-Stuck can perfectly use for thick-tailed alpha0stable pdfs. For the features of the data, when the alpha is increasing, there are more data oscillation between 0 to 0.3. From the figure 9, it shows, when the alpha is small, the number is close to 0. As the beta increase, there are no big different from the view perspective, but there are more none negative values. The equation can explain the reason. The time series can better explain the result, as the value are more random for larger alpha. For the small alpha, it close to 0.