EE511 Project6

Name: Yijing Jiang

Email: yijingji@usc.edu

USC ID: 5761477714

Question 1

Explane:

- Generate 1000 samples of A = X + Y, where X~N(1,4), Y~N(2,9):
  - theoretically, A~N(3,13)
- Use the **Box− Muller method**:
  - generate two independent vars u1, u2, both uniformly distributed on [0,1]
  - compute z0 and z1 as follow:

$$Z_0 = R\cos(\Theta) = \sqrt{-2\ln U_1}\cos(2\pi U_2)$$

and

$$Z_1 = R\sin(\Theta) = \sqrt{-2\ln U_1}\sin(2\pi U_2).$$

  - get X and Y: X = 2*z0 +1, Y = 3*z1+2
  - get A: A = X + Y
- Use the **Polar Marsaglia method**:
  - generate two independent vars u1, u2, both uniformly distributed on [0,1]
  - compute s = u1^2 + u2^2, only accept when 0<s<1.
  - compute z0 and z1 as follow:

$$z_0 = \sqrt{-2\ln U_1}\cos(2\pi U_2) = \sqrt{-2\ln s}\left(\frac{u}{\sqrt{s}}\right) = u\cdot\sqrt{\frac{-2\ln s}{s}}$$

and

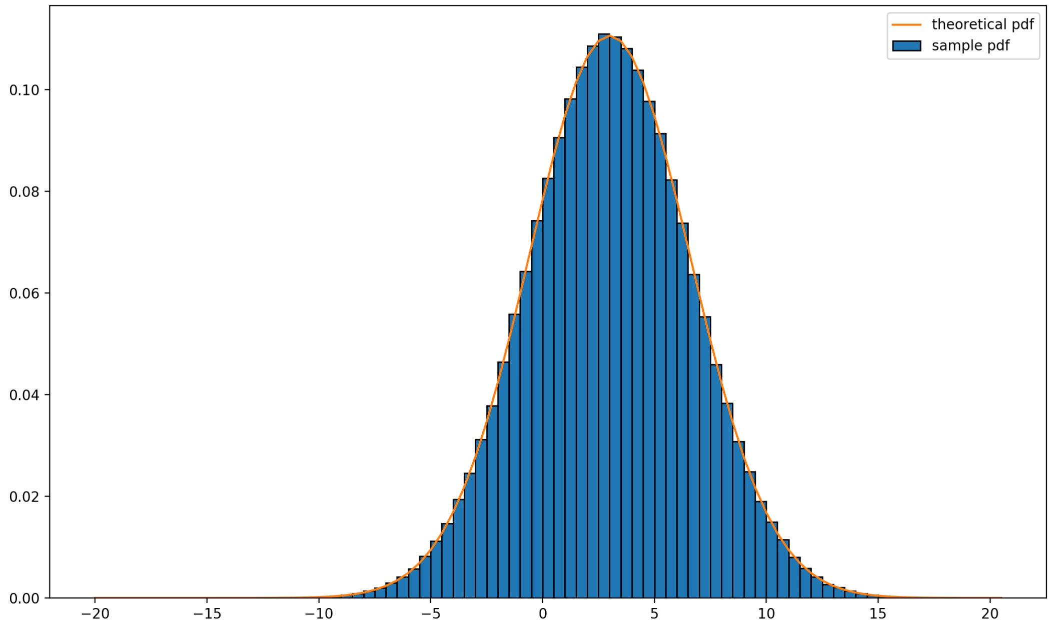
$$z_1 = \sqrt{-2\ln U_1}\sin(2\pi U_2) = \sqrt{-2\ln s}\left(\frac{v}{\sqrt{s}}\right) = v\cdot\sqrt{\frac{-2\ln s}{s}}.$$

  - get X and Y: X = 2*z0 +1, Y = 3*z1+2
  - get A: A = X + Y
- For computational time compare, use *time.time()* to record.

Result:

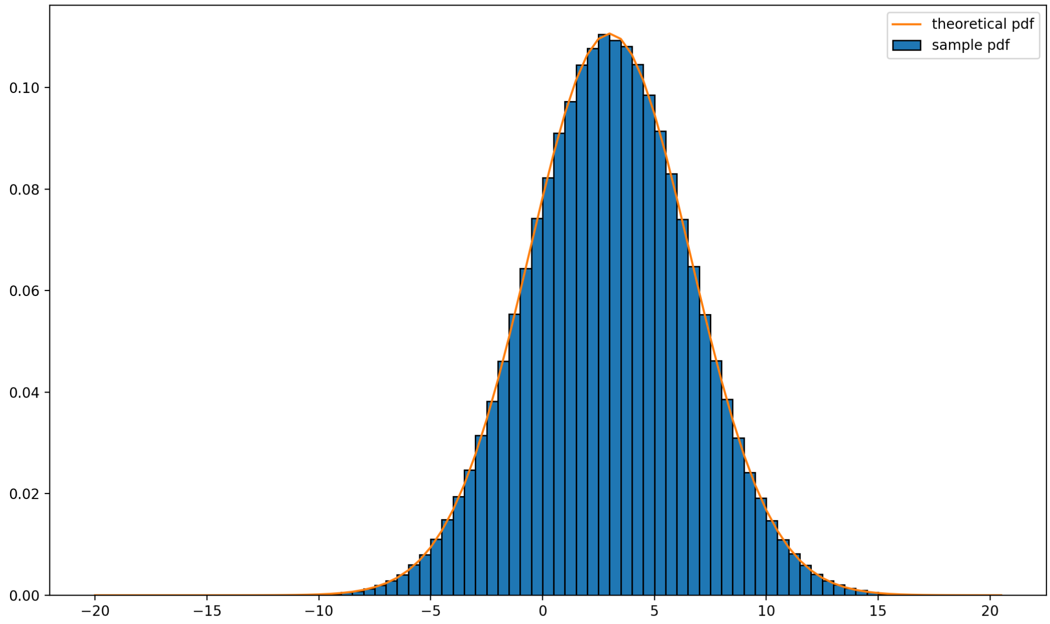
- **Box– Muller method:**
  - covariance of X and Y is very small, thus X and Y are uncorrelated.
  - the sample mean and sample variance are both similar to the theoretical values.
  - the histogram of sample pdf is also similar to theoretical plot of pdf.
  - computation time of this method is 0.17099

```
computational time:  0.17099595069885254
cov(X,Y) =  -2.835719264845381e-31
sample mean =  2.996303915979312
sample variance =  13.01427163194585
theoretical mean =  3
theoretical variance =  13
```

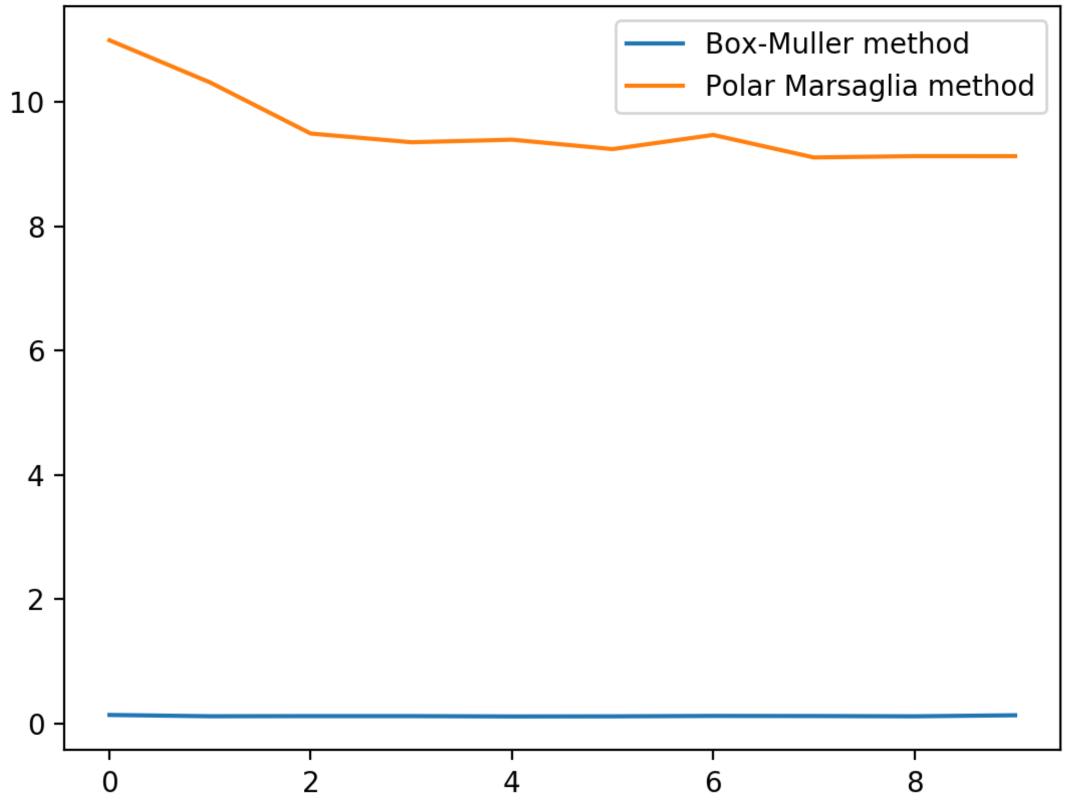


- **Polar Marsaglia method:**
  - covariance of X and Y is very small, thus X and Y are uncorrelated.
  - the sample mean and sample variance are both similar to the theoretical values.
  - the histogram of sample pdf is also similar to theoretical plot of pdf.
  - the computation time of this method is 10.98511

```
computational time:  10.985111713409424
cov(X,Y) =  -3.771730335740132e-33
sample mean =  3.001066824000856
sample variance =  13.01187190272292
theoretical mean =  3
theoretical variance =  13
```



- Compare **computation time** of these two method:
  - Box− Muller method is much faster then Polar Marsaglia method. The reason why the second method needs more computation time may be the while loop and the probability of rejection of u1 and u2.



Code:

[Box-Muller method]

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

start_box = time.time()
N = 1000000 #num of samples
M1 = 1 #Mean of X
V1 = 4 #Variance of X
M2 = 2 #Mean of Y
V2 = 9 #Variance of Y

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

#Generate Z0,Z1 ~ N(0,1)
Z0 = np.sqrt(-2*np.log(u1)) * np.cos(2*np.pi*u2)
Z1 = np.sqrt(-2*np.log(u1))*np.sin(2*np.pi*u2)

#Scale them to a particular mean and variance
X = np.sqrt(V1)*Z0 + M1 # X~ N(M1,V1)
Y = np.sqrt(V2)*Z1 + M2 # Y~ N(M2,V2)

end_box = time.time()
print("computational time: ", end_box - start_box)

mean_X = np.mean(X)
mean_Y = np.mean(Y)
cov_XY = np.mean(X-mean_X) * np.mean(Y-mean_Y)
print("cov(X,Y) = ", cov_XY)

#get A
A = X + Y
sample_mean_A = np.mean(A)
sample_var_A = np.var(A)
theo_mean_A = M1 + M2
theo_var_A = V1 + V2
print("sample mean = ",sample_mean_A)
print("sample variance = ",sample_var_A)
print("theoretical mean = ",theo_mean_A)
print("theoretical variance = ",theo_var_A)

#histogram of A
bins = np.arange(-20, 21, 0.5)
plt.hist(A, bins, edgecolor = "black", density = True, label = "sample pdf")

```

```
#theoretical pdf of A
plt.plot(bins, norm.pdf(bins, loc = theo_mean_A, scale = np.sqrt(theo_var_A)), label = "theoretical pdf")

plt.legend()
plt.show()
```

[Polar Marsaglia method]

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

start_box = time.time()
N = 1000000 #num of samples
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
Z0 = np.zeros((N,))
Z1 = np.zeros((N,))
while i<N:
    u1 = 2*np.random.rand()-1 #generate u1~[-1,1]
    u2 = 2*np.random.rand()-1 #generate u2~[-1,1]
    s = u1*u1 + u2*u2
    if s > 0 and s < 1:
        Z0[i] = np.sqrt(-2*np.log(s)/s) * u1
        Z1[i] = np.sqrt(-2*np.log(s)/s) * u2
        i = i+1

#Scale them to a particular mean and variance
X = np.sqrt(V1)*Z0 + M1 # X ~ N(M1,V1)
Y = np.sqrt(V2)*Z1 + M2 # Y ~ N(M2,V2)

end_box = time.time()
print("computational time: ", end_box - start_box)

mean_X = np.mean(X)
mean_Y = np.mean(Y)
cov_XY = np.mean(X-mean_X) * np.mean(Y-mean_Y)
print("cov(X,Y) = ", cov_XY)

#get A
A = X + Y
```

```
sample_mean_A = np.mean(A)
sample_var_A = np.var(A)
theo_mean_A = M1 + M2
theo_var_A = V1 + V2
print("sample mean = ",sample_mean_A)
print("sample variance = ",sample_var_A)
print("theoretical mean = ",theo_mean_A)
print("theoretical variance = ",theo_var_A)

#histogram of A
bins = np.arange(-20, 21, 0.5)
plt.hist(A, bins, edgecolor = "black", density = True, label = "sample pdf")

#theoretical pdf of A
plt.plot(bins, norm.pdf(bins, loc = theo_mean_A, scale = np.sqrt(theo_var_A)), label = "theoretical pdf")

plt.legend()
plt.show()
```

[compare computation time]

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

time_basic = np.zeros((10,))
time_polar = np.zeros((10,))

for i in range (10):
    N = 1000000 #num of samples
    M1 = 1 #Mean of X
    V1 = 4 #Variance of X
    M2 = 2 #Mean of Y
    V2 = 9 #Variance of Y

    start_box = time.time()
    u1 = np.random.rand(N,1) #shape:(N,)
    u2 = np.random.rand(N,1) #shape:(N,)
    #Generate Z0,Z1 ~ N(0,1)
    Z0 = np.sqrt(-2*np.log(u1)) * np.cos(2*np.pi*u2)
    Z1 = np.sqrt(-2*np.log(u1)) * np.sin(2*np.pi*u2)
    #Scale them to a particular mean and variance
    X = np.sqrt(V1)*Z0 + M1 # X~ N(M1,V1)
    Y = np.sqrt(V2)*Z1 + M2 # Y~ N(M2,V2)
    end_box = time.time()
    time_basic[i] = end_box - start_box
```

```

    start_box = time.time()

    #Generate X and Y that are N(0,1) random variables and indepedent
    Z0 = np.zeros((N,))
    Z1 = np.zeros((N,))
    j = 0
    while j<N:
        u1 = 2*np.random.rand()-1 #generate u1~[-1,1]
        u2 = 2*np.random.rand()-1 #generate u2~[-1,1]
        s = u1*u1 + u2*u2
        if s > 0 and s < 1:
            Z0[j] = np.sqrt(-2*np.log(s)/s) * u1
            Z1[j] = np.sqrt(-2*np.log(s)/s) * u2
            j = j+1
    #Scale them to a particular mean and variance
    X = np.sqrt(V1)*Z0 + M1 # X ~ N(M1,V1)
    Y = np.sqrt(V2)*Z1 + M2 # Y ~ N(M2,V2)
    end_box = time.time()
    time_polar[i] = end_box - start_box



bins = np.arange(0, 10, 1)
#theoretical pdf of A
plt.plot(bins, time_basic, label = "Box-Muller method")
plt.plot(bins, time_polar, label = "Polar Marsaglia method")

plt.legend()
plt.show()
```

Question 2

Explane:

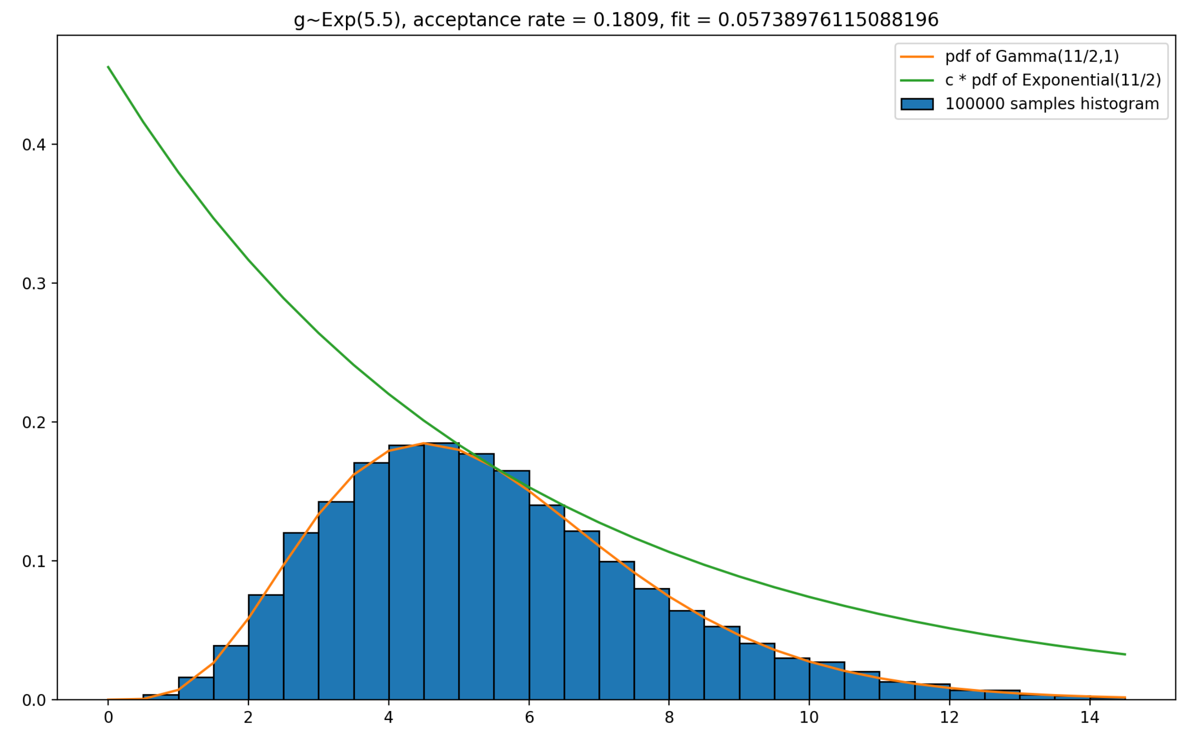
- Gamma distribution has the pdf as follow:

$$f(x;k,\theta) = \frac{x^{k-1}e^{-\frac{x}{\theta}}}{\theta^k\Gamma(k)} \quad \text{for } x > 0 \text{ and } k,\theta > 0.$$

- Use accept-refect method to sample from Gamma(5.5,1):
  - define g(x) ~ Exp(5.5), find the scalar c
  - generate a sample from g(x): generate y from U[0,1], compute j such that g(j) = y
  - generate u from U[0,1]
  - if u is less and equal to f(j)/[c*g(j)], then accept and record this j, otherwise, reject.
- Acceptance rate = number of accepted / total number of sampling

Results:

- **Plot of samples histogram and theoretical pdf:**
  - acceptance rate is 0.1809. We can see that the c*g is tangent to f.
  - overall fitness is 0.0573, which is really small and can show that our samples are fit to the distribution of Gamma(5.5,1).



Code:

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

#find gamma function when alpha = 5.5
a = 0.5
G = np.sqrt(np.pi)
while a < 5.5:
    G = a * G
    a = a + 1

#pdfX is pdf of gamma distribution
pdfX = lambda x: (x**4.5)*(np.exp(-x))/G
#pdfY is pdf of helper function(exponential distribution)
l = 2/11
pdfY = lambda y: l*(np.exp(-l*np.array(y)))

bins = np.arange(0,15,0.5)

#find c
ratio = np.divide(pdfX(bins),pdfY(bins))
c = np.max(ratio)

accept = []
for i in range (100000): #1000 is not enough
    y = np.random.rand()
    j = np.log(y/(c*l))/(-l)
    u = np.random.rand()
    if u <= pdfX(j)/(c*pdfY(j)):
        accept.append(j)

#acceptance rate
effi = len(accept)/100000

#plot the histogram
x = plt.hist(accept, bins, edgecolor = "black", density = True, label = "100000 samples histogram")
#fit
sample = x[0]
exp = pdfX(bins)
fit = 0
for i in range(np.shape(bins)[0]-1):
    if exp[i] > 0:
        fit = fit + (sample[i]-exp[i])**2/exp[i]
print("acceptance rate is ", effi)
print("overall fit is ", fit)

#plot the theoretical pdf
plt.plot(bins, pdfX(bins),label = 'pdf of Gamma(11/2,1)')
```

```
plt.plot(bins, pdfY(bins)*c,label = 'c * pdf of Exponential(1
1/2)')
plt.title("g~Exp(11/2), acceptance rate = "+str(effi)
+", fit = "+str(fit))

plt.legend()
plt.show()
```

Question 3

Explain:

- Alpha-stable distribution has four core parameters: alpha, beta, c and gamma.
  - alpha is the tail thickness parameter. Smaller alpha will lead to thicker tail. Alpha is in range (0,2].
  - beta is the symmetry parameter. Beta is in range [-1,1]. Distribution is symmetric when beta is equal to 0.
  - c is the location parameter, which shows the median of the distribution. Here we set c = 0.
  - gamma is the diperison parameter, which can be seen as the width of the distribution. Here we set gamma = 1.
- Use the Chambers−Mallows−Stuck method to generate samples from alpha stable distribution. In python code, we define a stblrnd function to realize this method. The formula of this method:

Proposition 2.1 (Zolotarev, 1986, Remark 1, page 78). *Let*

$$\varepsilon(\alpha) = \mathrm{sign}(1-\alpha),$$

$$\gamma_0 = -\frac{\pi}{2}\beta_2\frac{K(\alpha)}{\alpha},$$

Theorem 3.1. *Let γ_0 be defined as in Proposition 2.1. Let γ be uniformly distributed on $(-\frac{\pi}{2}, \frac{\pi}{2})$ and W be an independent exponential random variable with mean 1. Then*

- *for $\alpha \neq 1$*

$$X = \frac{\sin\alpha(\gamma-\gamma_0)}{(\cos\gamma)^{1/\alpha}}\left(\frac{\cos(\gamma-\alpha(\gamma-\gamma_0))}{W}\right)^{(1-\alpha)/\alpha}, \tag{3.2}$$

is $S_\alpha(1, \beta_2, 0)$ and

- *for $\alpha = 1$*

$$X = \left(\frac{\pi}{2}+\beta_2\gamma\right)\tan\gamma - \beta_2\log\left(\frac{W\cos\gamma}{\frac{\pi}{2}+\beta_2\gamma}\right) \tag{3.3}$$

is $S_1(1, \beta_2, 0)$

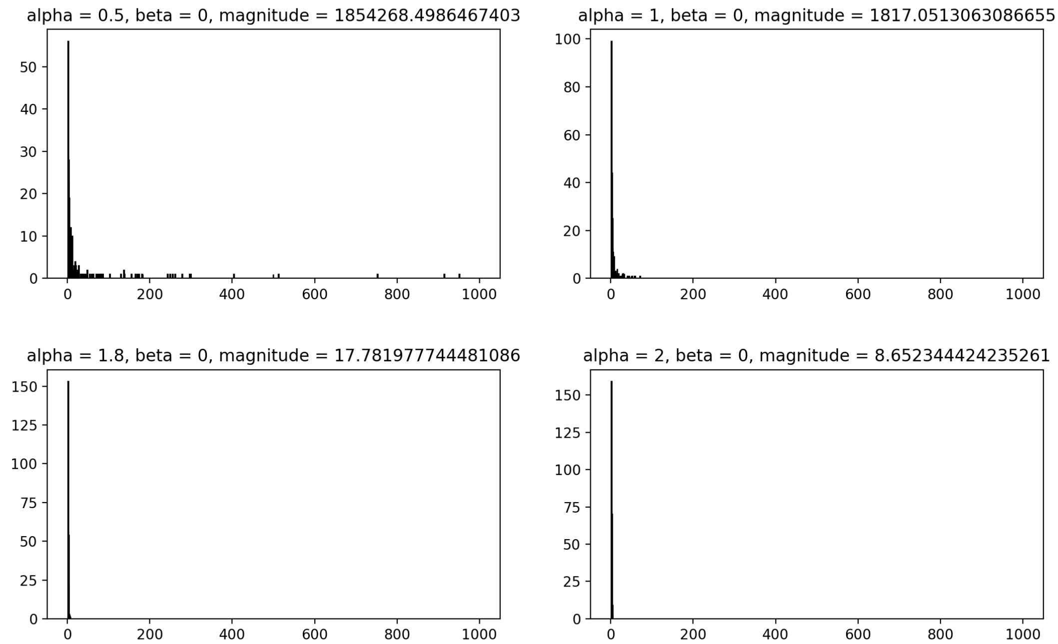
for the representation (2.5).

- Then show the samples in time series, that is the xray of histogram is 1 to 1000. Find the maximum and the minimum absolute values in the samples, the magnitude = maximum - minimum.
- Use *scipy.stats.levy_stable* to find the theoretical alpha-stable pdf with different alpha and beta. Also plot the hist of pdf of the samples.

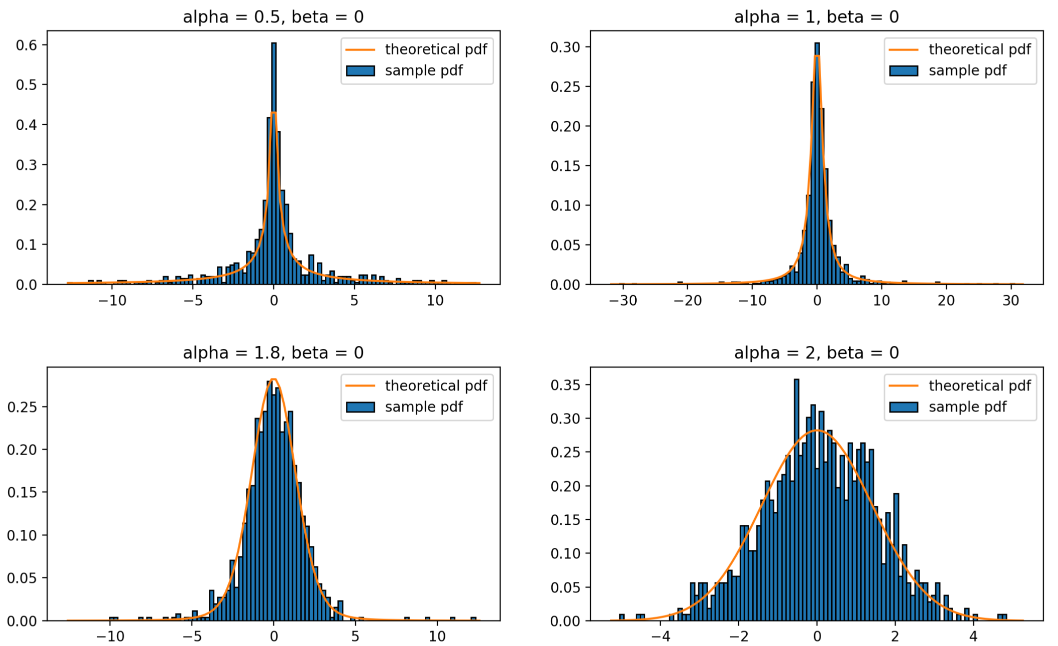
<u>Results:</u>

- **When alpha = 0.5, 1, 1.8, 2 and beta = 0:**

  - Histogram and the time series with magnitude:



  - Samples pdf and theoretical pdf:

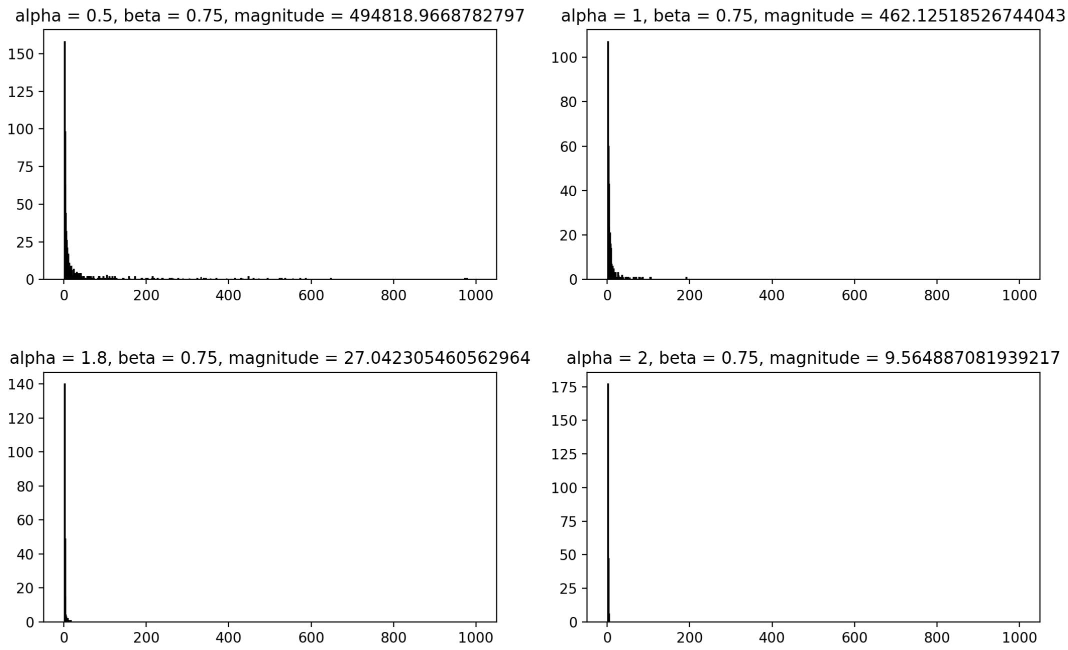


  - <u>The magnitude is decreasing rapidly when alpha is bigger.</u> If alpha is small, there will be very large values at the beginning of sampling, which causes the large magnitude. Otherwise, when alpha is big, there will not be very large values at the beginning of sampling.

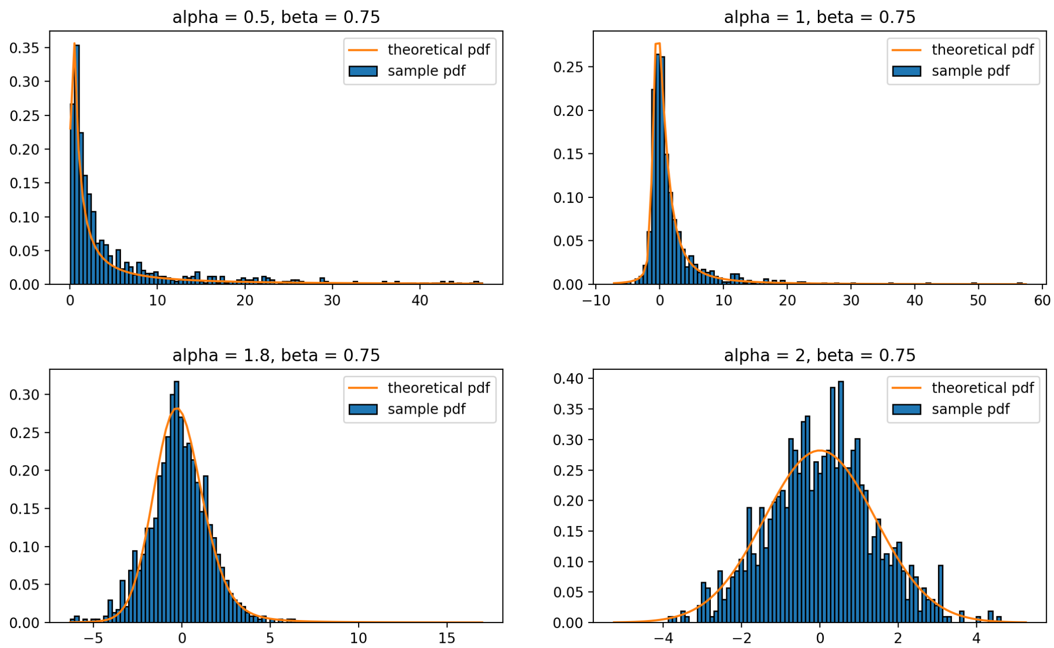
- In time series, alpha-stable distribution will close to zero finally. If alpha is small, the distribution will close to zero at around 100 samples and still has some values floating from zero. If alpha is big, the distribution will close to zero in small samples and almost has no values floating from zero.
- For the pdf, our theoretical pdf is similar to samples pdf.

- When alpha = 0.5, 1, 1.8, 2 and beta = 0.75:

- Histogram and the time series with magnitude:



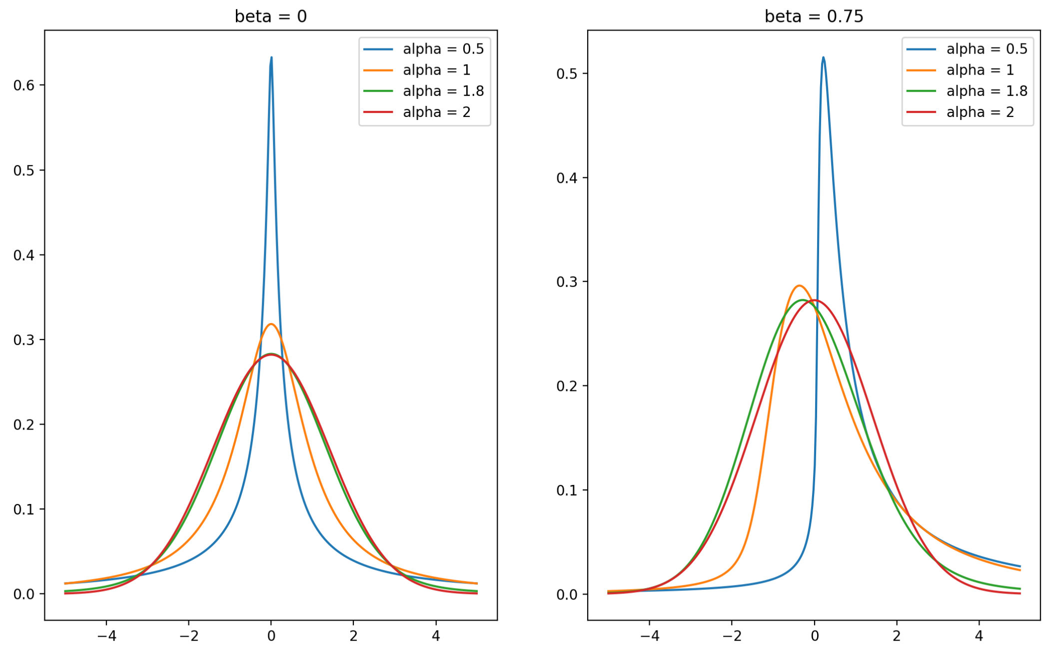
- Samples pdf and theoretical pdf:

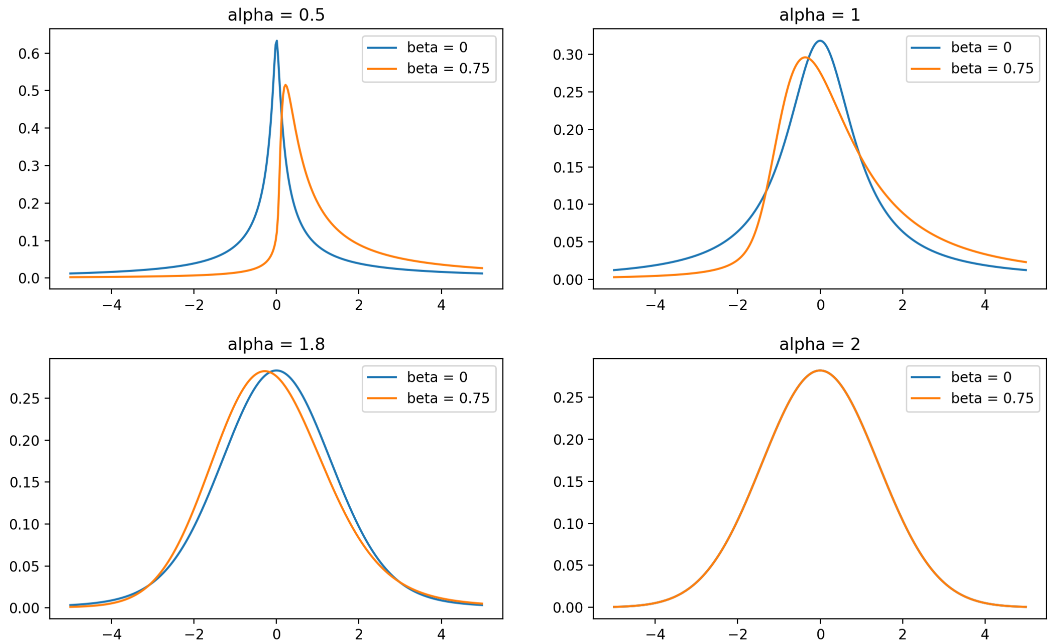


- The magnitude and the time series have the same properties as deta = 0.
- For the pdf, our theoretical pdf is also similar to samples pdf.

- Compare distribution with different alpha and beta:

- alpha: in both cases, when alpha is smaller, the tail of pdf is thicker.
- beta: when beta is not equal to zero, the distribution will not be symmetric to the location, excpet when alpha is equal to 2. When alpha is equal to 2, it is Gaussian distribution, such that, no matter what the beta is, the distribution will be symmetric to the median.





Code:

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


#define the function to generate stable distribution with scale = 1 and location = 0
def stblrnd(alpha, beta, size):
    samples = []
    for i in range (size):
        u = np.random.rand()*np.pi-np.pi/2
        w = -np.log(np.random.rand())
        s1 = -beta*np.tan(np.pi*alpha/2)
        if alpha == 1:
            s2 = np.pi/2
            sample = (1/s2)*((np.pi/2+beta*u)*np.tan(u)-beta*np.log((np.pi*w*np.cos(u)/2)/(np.pi/2+beta*u)))
        else:
            s2 = (1/alpha)*np.arctan(-s1)
            sample = ((1+s1*s1)**(1/2*alpha))*(np.sin(alpha*u+s2)/(np.cos(u)**(1/alpha)))*((np.cos(u-alpha*(u+s2))/w)**((1-alpha)/alpha))

        samples.append(sample)
    return samples



N = 1000

alpha = [0.5,1,1.8,2]
beta = [0,0.75]

#beta = 0, time series
beta0 = [] #save 4 group of samples
for i in range(4):
    plt.subplot(2,2,i+1)
    x0 = np.arange(1,N+1,1)
    r = stblrnd(alpha[i],beta[0],N)
    beta0.append(r)
    r_abs = np.abs(r)
    size = np.amax(r_abs) - np.amin(r_abs)
    plt.hist(r_abs, x0, edgecolor = "black")
    plt.title("alpha = "+str(alpha[i])+", beta = "+str(beta[0])+", magnitude = "+str(size))
plt.show()

#beta = 0, compare with theoretical
```

```
for i in range(4):
    plt.subplot(2,2,i+1)
    left = 1/(10**(i+1))
    right = 1-left

    x = np.linspace(levy_stable.ppf(left, alpha[i], beta[0]), levy_stable.ppf(right, alpha[i], beta[0]), 100)

    plt.hist(beta0[i], x, edgecolor = "black", density = True, label = "sample pdf")
    #theoretical
    rv = levy_stable(alpha[i], beta[0])
    plt.plot(x, rv.pdf(x), label="theoretical pdf")
    plt.title("alpha = "+str(alpha[i])+", beta = "+str(beta[0]))
    plt.legend()
plt.show()


#beta = 0.75, time series
beta075 = [] #save 4 group of samples
mag = []
for i in range(4):
    plt.subplot(2,2,i+1)
    x0 = np.arange(1,N+1,1)
    r = stblrnd(alpha[i],beta[1],N)
    beta075.append(r)
    r_abs = np.abs(r)
    size = np.amax(r_abs) - np.amin(r_abs)
    mag.append(size)
    plt.hist(r_abs, x0, edgecolor = "black")
    plt.title("alpha = "+str(alpha[i])+", beta = "+str(beta[1])+", magnitude = "+str(size))
plt.show()

plt.plot(alpha,mag)
plt.xlabel("alpha")
plt.ylabel("magnitude")
plt.show()

#beta = 075, compare with theoretical
for i in range(4):
    plt.subplot(2,2,i+1)
    left = 1/(10**(i+1))
    right = 1-left

    x = np.linspace(levy_stable.ppf(left, alpha[i], beta[1]), levy_stable.ppf(right, alpha[i], beta[1]), 100)

    plt.hist(beta075[i], x, edgecolor = "black", density = True, label = "sample pdf")
    #theoretical
```

```
    rv = levy_stable(alpha[i], beta[1])
    plt.plot(x, rv.pdf(x), label="theoretical pdf")
    plt.title("alpha = "+str(alpha[i])+", beta = "+str(beta[1]))
    plt.legend()
plt.show()
```

Reference:

① **https://en.wikipedia.org/wiki/Stable_distribution**

② **https://en.wikipedia.org/wiki/Box%E2%80%93Muller_transform**

③ **https://en.wikipedia.org/wiki/Gamma_distribution**