Sina Mahbobi

March 22, 2020

1 EE511 Project 6

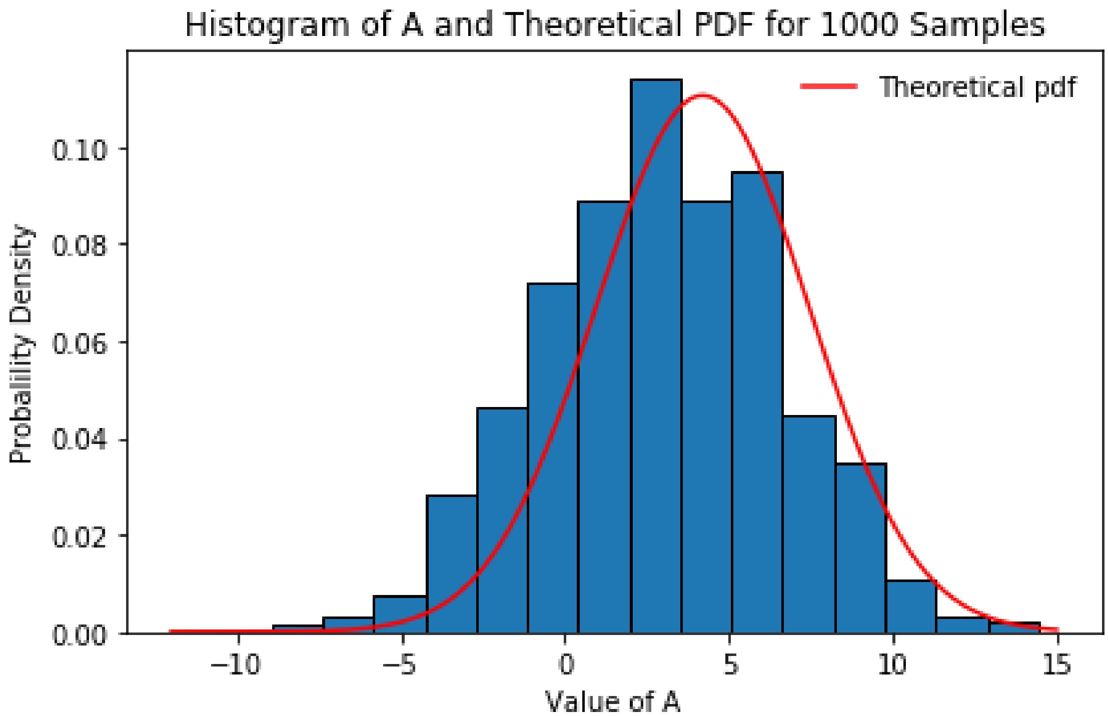
1.0.1 Question 1

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

[2]:
```
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


theory_min = -15
theory_max = 15
theory_range = np.linspace(theory_min, theory_max, 1000)
theory_pdf = scipy.stats.norm.pdf(theory_range,3,math.sqrt(9+4))
```

```python
#plt.hist(A, bins = 15, edgecolor = 'black', facecolor = 'orange')
#plt.plot(x_range, theory_pdf)
#plt.show()

#print(theory_pdf)

num_bins = 15
fig, ax = plt.subplots()
# the histogram of the data
n, bins, patches = ax.hist(A, num_bins, edgecolor = 'black', density = True)

plt_range = np.linspace(-12, 15, 1000)

ax.plot(plt_range, theory_pdf, ms=8, label='Theoretical pdf', color = 'red')
ax.legend(loc='best', frameon=False)
plt.xlabel("Value of A")
plt.ylabel("Probalility Density")
plt.title("Histogram of A and Theoretical PDF for 1000 Samples")
fig.tight_layout()

print("Mean of A: ", str(A.mean()))
print("Theoretical Mean: 3")

mean_x = x.mean()
mean_y = y.mean()

cov = sum((a - mean_x) * (b - mean_y) for (a,b) in zip(x,y)) / len(x)
print("Estimated Covariance of X, Y: ", str(cov))
print("Estimated Variance of A: ",str(np.var(A)) )
print("Theoretical Variance of A: 13")
```

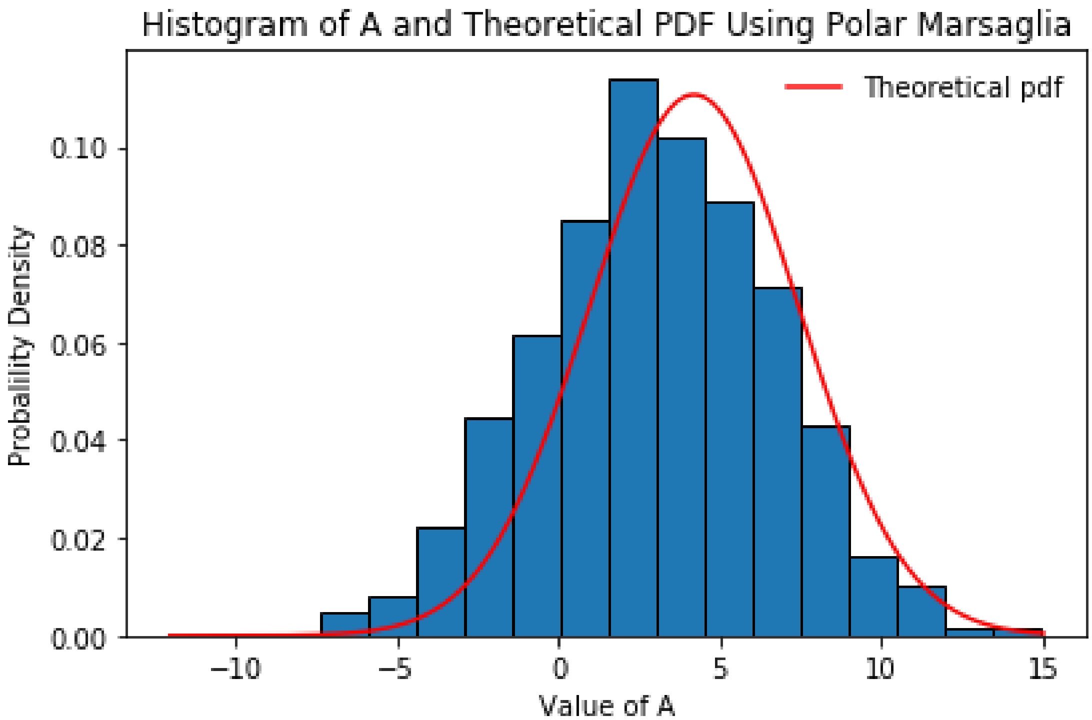
```
Mean of A:  2.976872418708574
Theoretical Mean: 3
Estimated Covariance of X, Y:  [-0.03121696]
Estimated Variance of A:  13.105438620349764
Theoretical Variance of A: 13
```



[9]:
```
#start_mar = time.time()
M1 = 1 # Mean of X
M2 = 2 # Mean of Y
V1 = 4 # Variance of X
V2 = 9 # Variance of Y
j = 0 # the random number generated by the algorithm
X1 = np.empty(1000)
Y1 = np.empty(1000)
# Generate X and Y that are N(0,1) random variables and indepedent
while j<=999:
    u1 = 2*np.random.rand()-1
    u2 = 2*np.random.rand()-1
    s = u1*u1 + u2*u2
    if s < 1:
        X1[j] = np.sqrt(-2*np.log(s)/s)*u1
        Y1[j] = np.sqrt(-2*np.log(s)/s)*u2
        j = j+1

# Scale them to a particular mean and variance
x = np.sqrt(V1)*X1 + M1; # x~ N(M1,V1)
y = np.sqrt(V2)*Y1 + M2; # y~ N(M2,V2)
A_p = x + y
#end_mar = time.time()
```

```python
#print(len(A_p))
#print(end_mar - start_mar)

num_bins = 15
fig, ax = plt.subplots()
# the histogram of the data
n, bins, patches = ax.hist(A_p, num_bins, edgecolor = 'black', density = True)

plt_range = np.linspace(-12, 15, 1000)
theory_min = -15
theory_max = 15
theory_range = np.linspace(theory_min, theory_max, 1000)
theory_pdf = scipy.stats.norm.pdf(theory_range,3,math.sqrt(9+4))


ax.plot(plt_range, theory_pdf, ms=8, label='Theoretical pdf', color = 'red')
ax.legend(loc='best', frameon=False)
plt.xlabel("Value of A")
plt.ylabel("Probalility Density")
plt.title("Histogram of A and Theoretical PDF Using Polar Marsaglia")
fig.tight_layout()

print("Mean of A: ", str(A_p.mean()))
print("Theoretical Mean: 3")

mean_x = x.mean()
mean_y = y.mean()
cov = sum((a - mean_x) * (b - mean_y) for (a,b) in zip(x,y)) / len(x)
print("Estimated Covariance of X, Y: ", str(cov))
print("Estimated Variance of A: ",str(np.var(A_p)) )
print("Theoretical Variance of A: 13")
```

```
Mean of A:  3.010556805141089
Theoretical Mean: 3
Estimated Covariance of X, Y:  -0.1470796926566637
Estimated Variance of A:  12.813747993133235
Theoretical Variance of A: 13
```

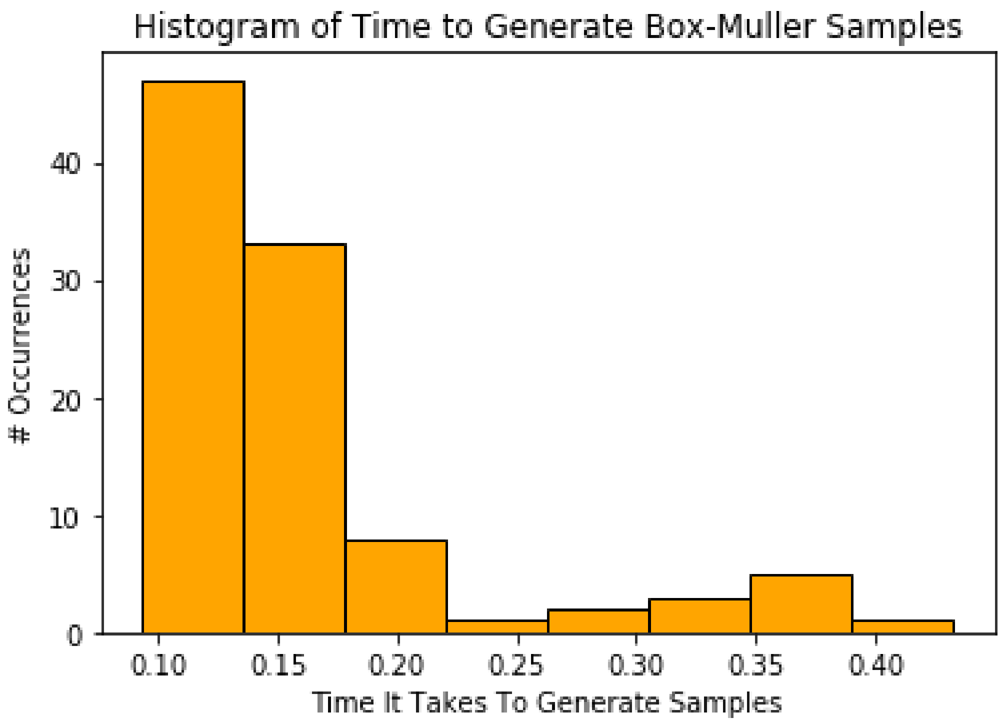


[10]:

```
box_arr = []
for i in range(0,100):
    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

    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)

    end_box = time.time()
    box_arr.append(end_box - start_box)
plt.hist(box_arr, bins = 8, edgecolor = 'black', facecolor = 'orange')
```

```
plt.xlabel("Time It Takes To Generate Samples")
plt.ylabel("# Occurrences")
plt.title("Histogram of Time to Generate Box-Muller Samples")
```

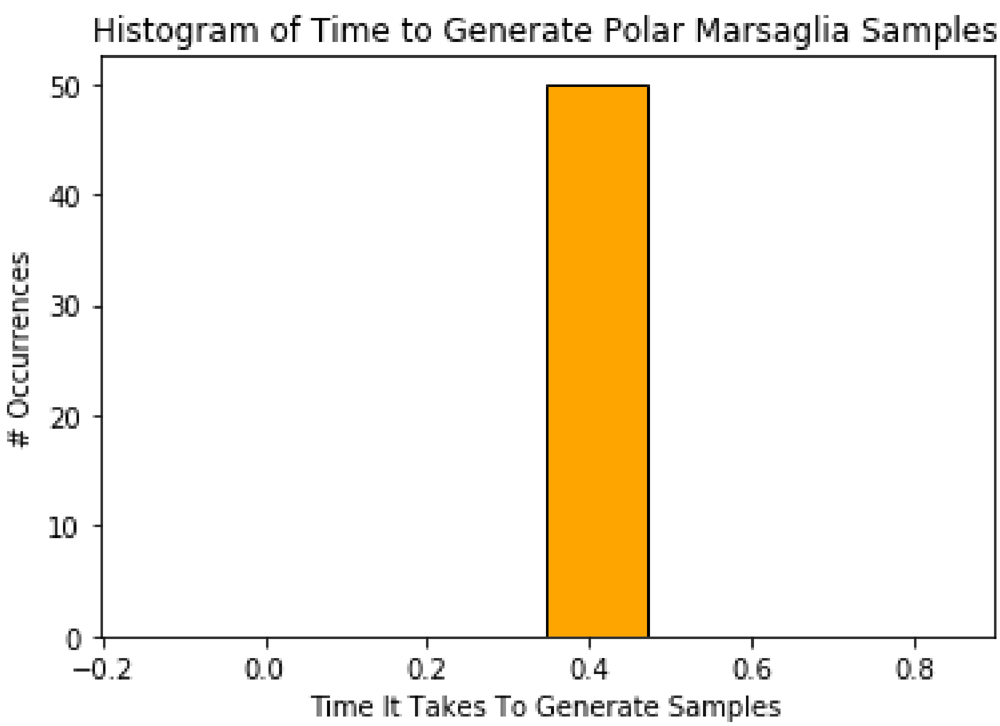
[10]: Text(0.5, 1.0, 'Histogram of Time to Generate Box-Muller Samples')



[12]:
```
box_arr = []
for i in range(0,50):
    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 independent
    while i<=999999:
        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

    # 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_p = x + y
    end_mar = time.time()
    box_arr.append(end_box - start_box)
plt.hist(box_arr, bins = 8, edgecolor = 'black', facecolor = 'orange')
plt.xlabel("Time It Takes To Generate Samples")
plt.ylabel("# Occurrences")
plt.title("Histogram of Time to Generate Polar Marsaglia Samples")
```

[12]: Text(0.5, 1.0, 'Histogram of Time to Generate Polar Marsaglia Samples')



Question 1 Analysis

- The first graph is a histogram of 1000 samples generated using the Box-Muller Method, and the theoretical pdf layed over the histogram.
  - As seen in the data, the covariance between X and Y is close to zero, which makes sense since the two variables are independently generated and theoretically independent.
  - This is a fairly precise method of generated values from a normal distribution, given that both the estimated mean and variance are within hundreths of the theoretical mean and variance.

- The second graph is a histogram of 1000 samples generated using the Polar Marsaglia Method, and the theoretical pdf layed over the histogram.
  - As seen in the data, the covariance between X and Y is close to zero, which makes sense since the two variables are independently generated and theoretically independent.
  - This is also a fairly precise method of generated values from a normal distribution, given that both the estimated mean and variance are within hundreths of the theoretical mean and variance.
  - However, the histogram for this method fits the theoretical pdf better than that of the Box-Muller Method.
- The third graph is a histogram of the time it takes to generate 1,000,000 Box-Muller samples for 100 trials.
- The fourth graph is a histogram of the time it takes to generate 1,000,000 Polar Marsaglia samples for 50 trials.
- Although the Polar Marsaglia method histogram matches the theoretical pdf in a more precise and accurate manner, the time needed to produce Box-Muller plots is on average much lower, resulting in the Box-Muller method being the more computationally efficient algorithm.

[ ]:

```

```

1.0.2 Question 2

[34]:

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

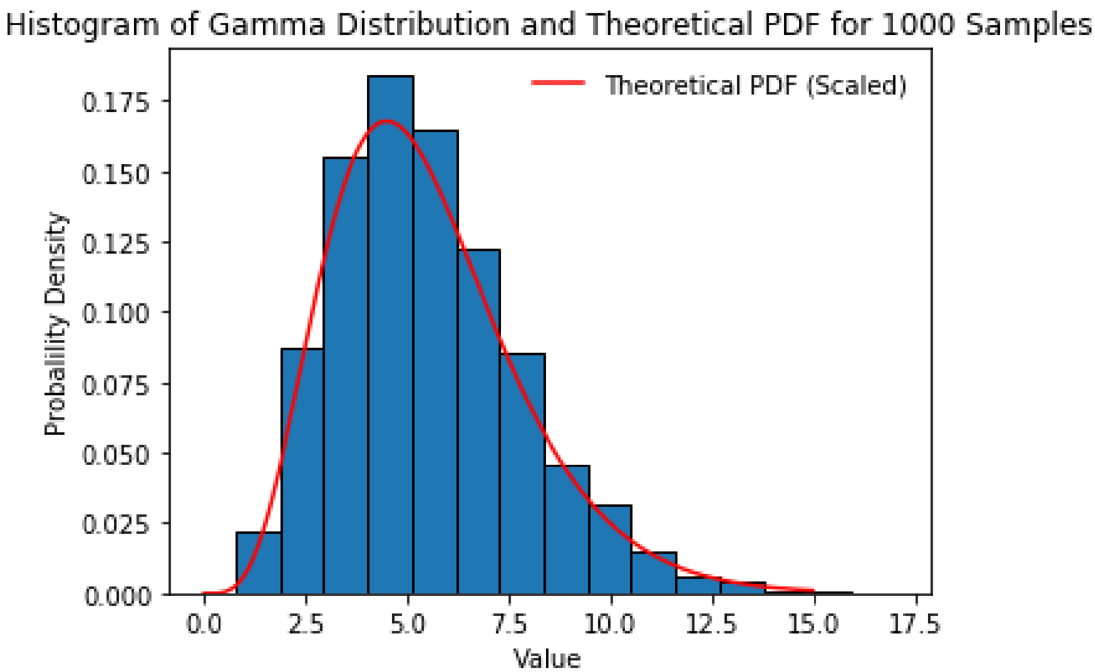
[56]:

```python
valid_outcomes = []
accepted = 0
rejected = 0
shape = 5.5
scale = 1

pdfX = lambda x: (2/(np.sqrt(np.pi)))*(np.array(x) ** (9/2))*(np.exp(-x))
pdfY = lambda y: (2/11)*(np.exp(-(2/11)*np.array(y)))

t = np.arange(0,15,0.01)
ratio = np.divide(pdfX(t),pdfY(t))
c = np.max(ratio)
fig = plt.figure(figsize=(8,6),dpi=100)

#print(ratio)
"""
plt.plot(pdfX(t),label = 'pdf of Gamma(11/2,1)')
plt.plot(pdfY(t),label = 'pdf of Exponential(11/2)')
plt.plot(pdfY(t)*c,label = 'c * pdf of Exponential(11/2)')
plt.legend()
plt.show()

```

```
"""

for i in range(0, 10000):


    y1 = np.random.exponential(5.5)
    #print(y1)

    u = np.random.uniform()
    #print(u)

    if( u >= pdfX(y1)/(c*pdfY(y1)) ):
        rejected += 1

    else:
        valid_outcomes.append(y1)
        accepted +=1

efficiency  = accepted / (accepted + rejected)

num_bins = 15
fig, ax = plt.subplots()
# the histogram of the data
n, bins, patches = ax.hist(valid_outcomes, num_bins, edgecolor = 'black',␣
 ↪density = True)

plt_range = np.linspace(-12, 15, 1000)



ax.plot(t, pdfX(t)/65, ms=8, label='Theoretical PDF (Scaled)', color = 'red')
ax.legend(loc='best', frameon=False)
plt.xlabel("Value")
plt.ylabel("Probalility Density")
plt.title("Histogram of Gamma Distribution and Theoretical PDF for 10000␣
 ↪Samples")
fig.tight_layout()
print("The efficiency is: ", str(100* efficiency), "%")
```

```
The efficiency is:  39.78 %
```

```
<Figure size 800x600 with 0 Axes>
```



Question 2 Analysis

- In this graph, the Theoretical PDF is scaled so that it fits on the histogram for comparison purposes.
- This histogram was generated using 10000 samples.
- However, this is a relatively inefficient method. The efficiency of this method to generate values from the gamma distribution is just under 40%.

[ ]:

```

```

1.0.3 Question 3

[1]:

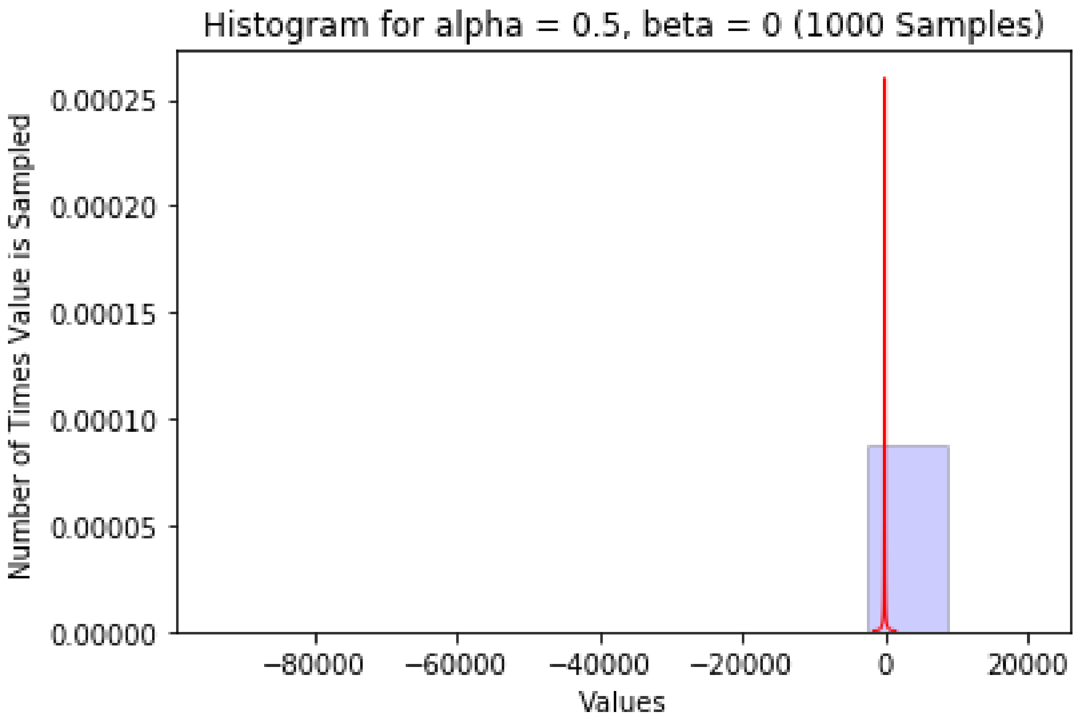
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import levy_stable
import math
```

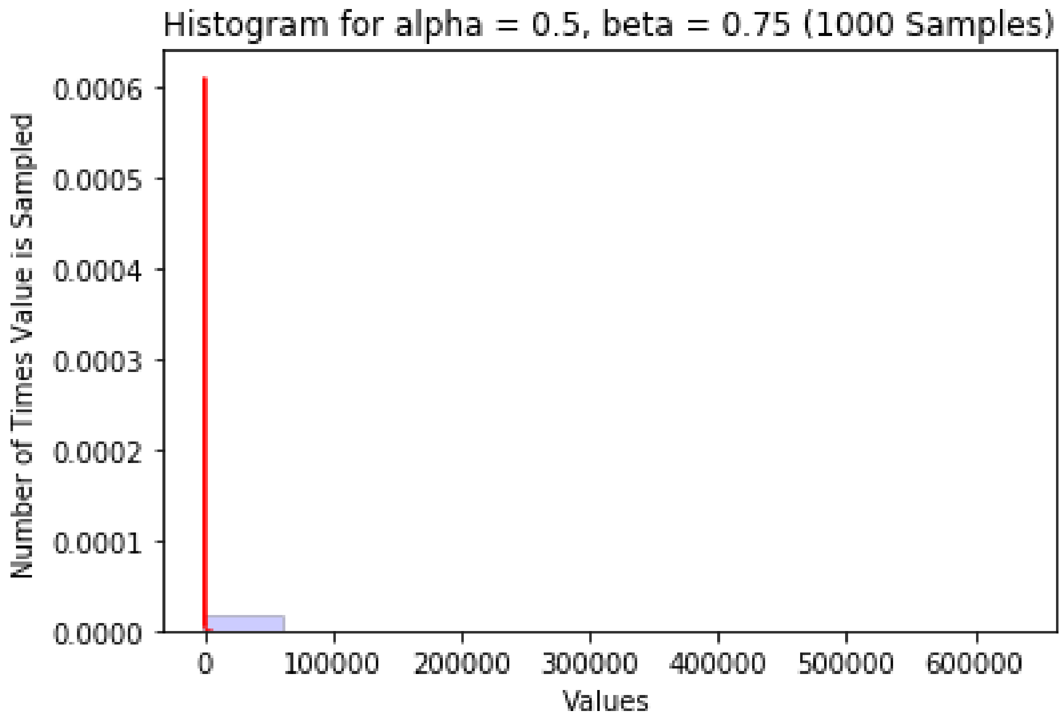
[5]:

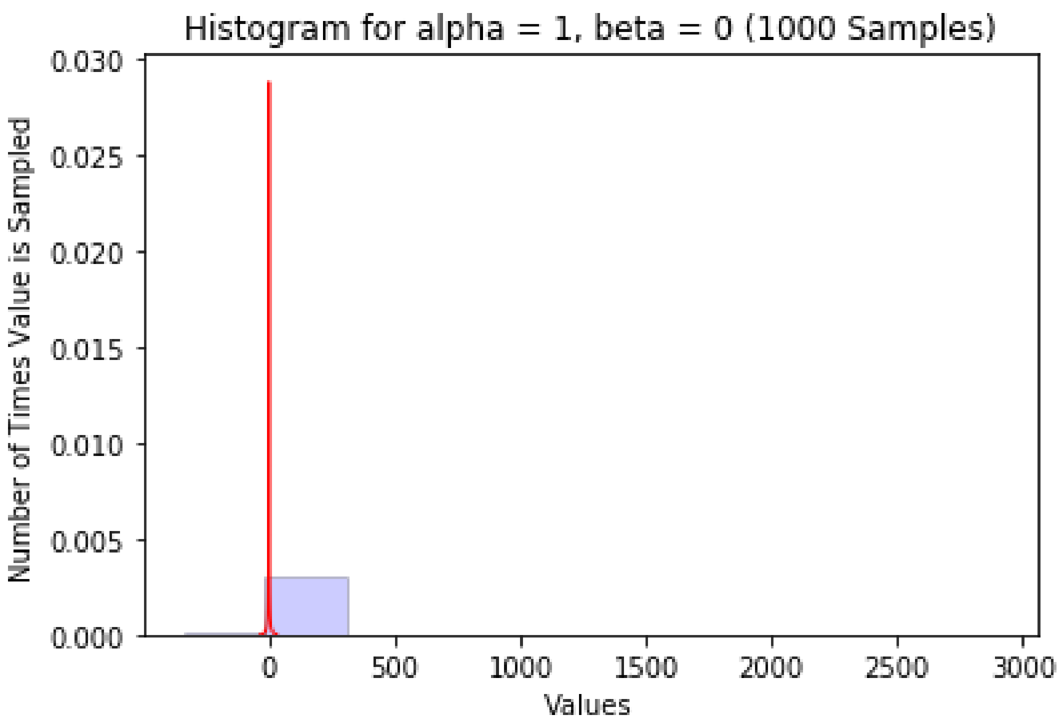
```
alphas = [.5, 1, 1.8, 2]
betas = [0, 0.75]

for alpha in alphas:
    for beta in betas:
        X = np.empty(1000)
        if (alpha != 1):
```

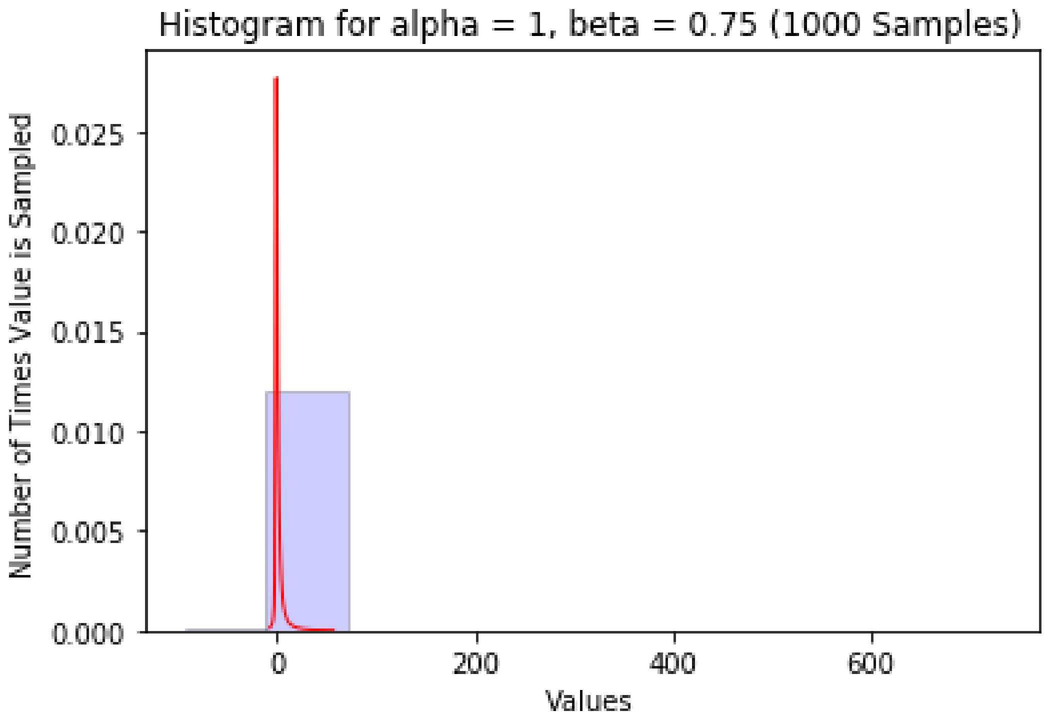
```
            B_ab = np.arctan(beta * np.tan(math.pi*alpha/2))/alpha
            #print(B_ab)
            S_ab =  math.pow( 1 + (math.pow(beta,2) * np.tan(math.pi*alpha/2)),␣
↪1/(2*alpha) )
            #print(S_ab)
            W = np.random.exponential()
            #print(W)
            for i in range(0, len(X)):
                V = np.random.uniform(-1*math.pi/2,math.pi/2);
                #print(V)
                X[i] = S_ab * (math.sin(alpha*(V+B_ab))/ math.pow(math.cos(V), 1/
↪alpha)) \
                    * math.pow((math.cos(V - alpha*(V+B_ab))/W), (1 -␣
↪alpha)*alpha)
                #print(X[i])
        else:
            for i in range(0, len(X)):
                V = np.random.uniform(-1*math.pi/2,math.pi/2);
                W = np.random.exponential()
                #print(W)
                X[i] = (2/math.pi)*( (math.pi/2 + beta* V)* np.tan(V) \
                        - (beta * math.log(W*math.cos(V)/(math.pi/2 + beta*V))) )
                #print(X[i])

        fig, ax = plt.subplots(1, 1)
        x_1 = np.linspace(levy_stable.ppf(0.01, alpha, beta), levy_stable.ppf(0.
↪99, alpha, beta), 100)
        ax.plot(x_1, levy_stable.pdf(x_1, alpha, beta)/10, 'r-', lw=1,␣
↪label='levy_stable pdf')
        ax.hist(X, density=True, edgecolor = 'black', facecolor = 'blue',␣
↪alpha=0.2)

        plt.xlabel("Values")
        plt.ylabel("Number of Times Value is Sampled")
        plt.title("Histogram for alpha = {a}, beta = {b} (1000 Samples)".
↪format(a=alpha, b = beta ))
        plt.show()

        """
        fig, ax = plt.subplots()
        x = np.linspace(-20, 20,10000)
        rv = levy_stable(alpha, beta)
        ax.hist( X, edgecolor = 'black', facecolor = 'blue')
        ax.legend(loc='best', frameon=False)
```

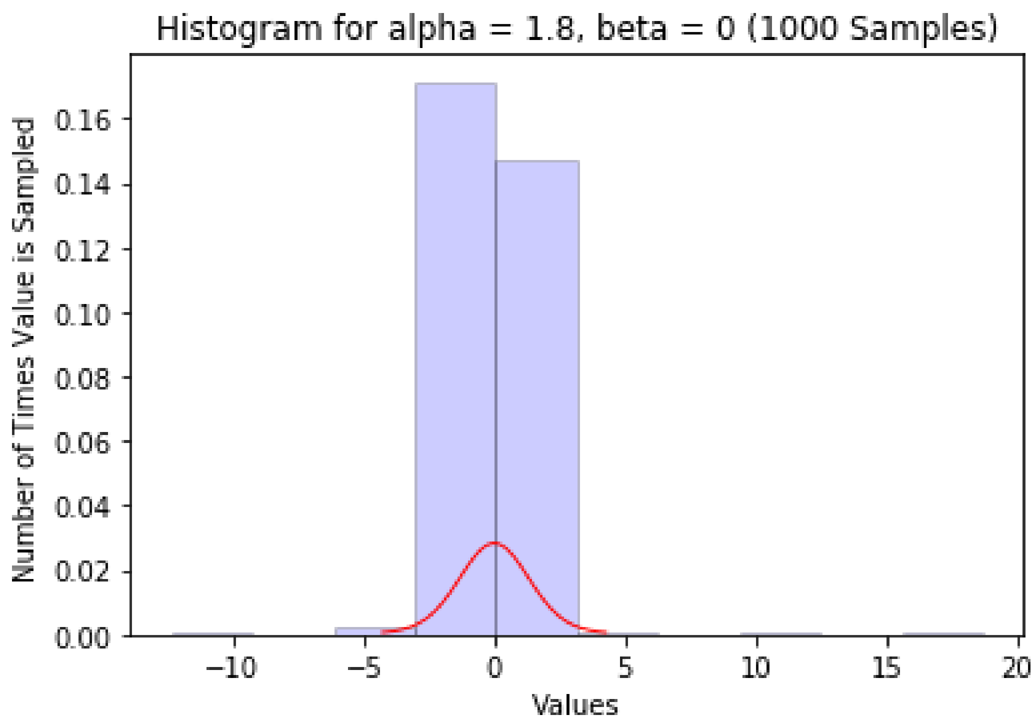
```
        ax.plot(x, rv.pdf(x), ms=8, label='Theoretical PDF (Scaled)', color =␣
↪'red')
        plt.xlabel("Values")
        plt.ylabel("Number of Times Value is Sampled")
        plt.title("Histogram for alpha = {a}, beta = {b} (1000 Samples)".
↪format(a=alpha, b = beta ))
        #plt.xlim(0, 10000)
        plt.show()
        """

        #print(max(X))
```

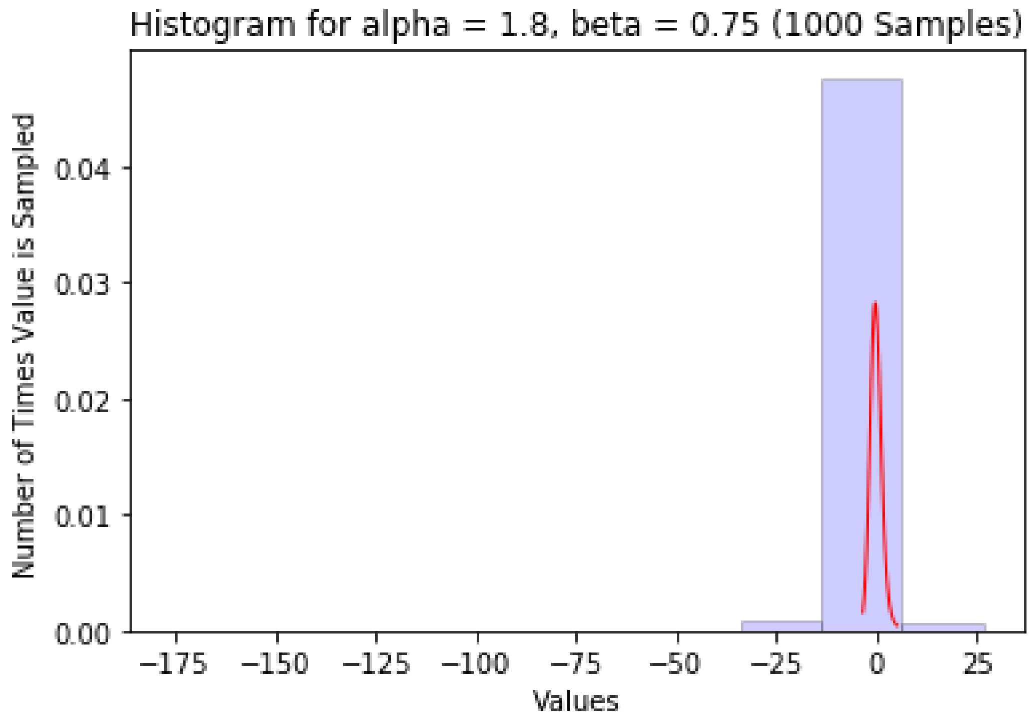


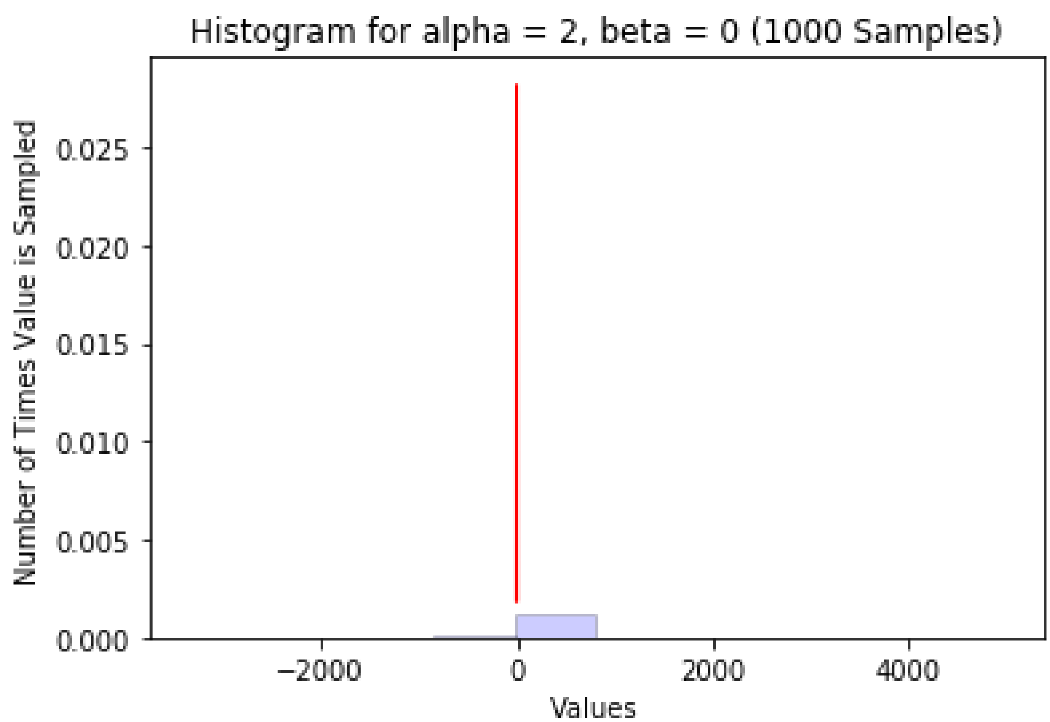

Histogram for alpha = 0.5, beta = 0.75 (1000 Samples)
Number of Times Value is Sampled
0.0000
0.0001
0.0002
0.0003
0.0004
0.0005
0.0006
0
100000
200000
300000
400000
500000
600000
Values

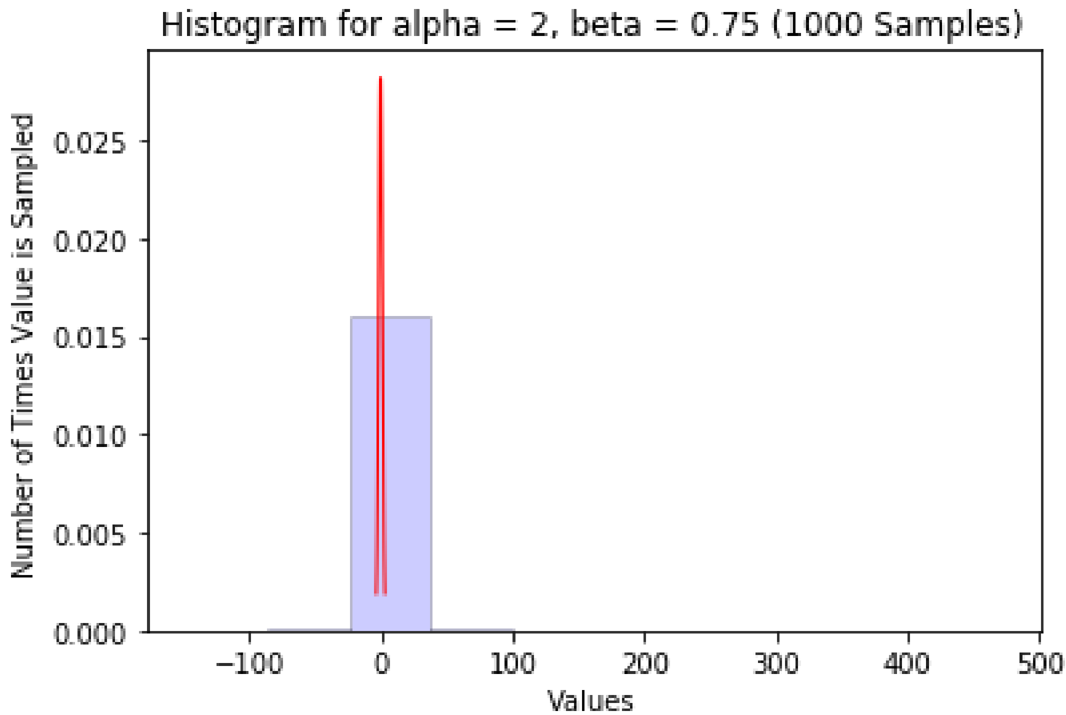

Histogram for alpha = 1, beta = 0 (1000 Samples)
Number of Times Value is Sampled
0.000
0.005
0.010
0.015
0.020
0.025
0.030
0
500
1000
1500
2000
2500
3000
Values


Histogram for alpha = 1, beta = 0.75 (1000 Samples)
Number of Times Value is Sampled
0.000
0.005
0.010
0.015
0.020
0.025
0
200
400
600
Values


Histogram for alpha = 1.8, beta = 0 (1000 Samples)
Number of Times Value is Sampled
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
0.16
−10
−5
0
5
10
15
20
Values


Histogram for alpha = 1.8, beta = 0.75 (1000 Samples)
Number of Times Value is Sampled
0.04
0.03
0.02
0.01
0.00
−175
−150
−125
−100
−75
−50
−25
0
25
Values


Histogram for alpha = 2, beta = 0 (1000 Samples)
Number of Times Value is Sampled
0.025
0.020
0.015
0.010
0.005
0.000
−2000
0
2000
4000
Values



Question 3 Analysis

- The 8 histograms above show the different combinations of alpha and beta values and that affects the different alpha stable distributions.
- For values of alpha between one and two, the distribution is a little wider and follows more of a curve.
- For values of alpha less than one, the values generated by X are larger.

```
[ ]:
```