**PROJECT #6 – Continuous Sapling**

**Divyanshu Sahay  
USC ID: 6364938199  
(dsahay@usc.edu)**

Important Note:

Tool Used: **Python 3.7.4 and MATLAB 2018**

1. **Sampling from Normal Random Variables.**

**Problem Statement:** 1000 samples have to be generated of a random variable A defined as ***A = X + Y*** where, and using Box Muller Method and Polar Marsaglia Method. Following are the requirements:

1. Histogram of A
2. Sample Mean and Sample Variance
3. Comparison with theoretical values.
4. Covariance between X and Y
5. Computational time for both for 1,000,000 samples.

**Understanding the Problem Statement:** This problem introduces alternative methods to generate samples from Normal Random Variables, namely Box Muller and Polar Marsaglia. Both the methods have defined set of equations which it uses for the sampling. The program to be written to fulfil desired requirements will essentially have same framework except for the sampling process that utilises different set of equations. For the both the algorithm the reference was taken from class notes and the programs provided by Professor Brandon. The program written has comments which explains the methodology and approach.

**Program Written:**

#-- BOX MULLER METHOD----#

import random

import math

import numpy as np

import scipy.stats as stats

import matplotlib.pyplot as plt

import statistics

import timeit

#--- THEORETICAL VALUES GIVEN

M1 = 1

M2 = 2

V1 = 4

V2 = 9

a\_list = []

x\_list = []

y\_list = []

start = timeit.default\_timer() # START TIME

for i in range (1000000):

    #----- THE ALGORITH-------

    u1 = np.random.rand()

    u2 = np.random.rand()

    X = (math.sqrt(-2\*math.log(u1)))\*math.cos(2\*math.pi\*u2)

    Y = (math.sqrt(-2\*math.log(u1)))\*math.sin(2\*math.pi\*u2)

    x = math.sqrt(V1)\*X + M1

    x\_list.append(x)

    y = math.sqrt(V2)\*Y + M2

    y\_list.append(y)

    a = x + y

    a\_list.append(a)

stop = timeit.default\_timer() # STOP TIME

print('Time: ', stop - start)

x\_min = -20.0

x\_max = 16.0

mean = 3.0

std = 3.6

x\_plot = np.linspace(x\_min, x\_max)

y\_plot = stats.norm.pdf(x\_plot,mean,std)

cov\_mat = np.stack((x\_list, y\_list))

print(np.cov(cov\_mat))

fig = plt.figure()

plt.hist(a\_list)

plt.plot(x\_plot,3042000\*y\_plot, color='red')

plt.xlabel('Samples Taken')

plt.ylabel('Frquency of Occurance of the Sample')

plt.title('BOX MULLER METHOD')

plt.show()

print('--PRACTICAL VALUES----')

sample\_mean\_practical = statistics.mean(a\_list)

print('sample\_mean\_practical:',sample\_mean\_practical)

sample\_variance\_practical = statistics.variance(a\_list)

print('sample\_variance\_practical:',sample\_variance\_practical)

#-- POLA MARSAGLIA METHOD----#

import random

import math

import numpy as np

import scipy.stats as stats

import matplotlib.pyplot as plt

import statistics

import timeit

#--- THEORETICAL VALUES GIVEN

M1 = 1

M2 = 2

V1 = 4

V2 = 9

i = 0

a\_list = []

x\_list = []

y\_list = []

start = timeit.default\_timer() # START TIME

while(i<=999999):

    #----- THE ALGORITH-------

    u3 = 2\*np.random.rand()-1

    u4 = 2\*np.random.rand()-1

    s = u3\*u3 + u4\*u4

    if(s < 1):

        i = i + 1

        X = math.sqrt(-2\*math.log(s)/s)\*u3

        x = math.sqrt(V1)\*X + M1

        Y = math.sqrt(-2\*math.log(s)/s)\*u4

        y = math.sqrt(V2)\*Y + M2

        x\_list.append(x)

        y\_list.append(y)

        a = x + y

        a\_list.append(a)

stop = timeit.default\_timer()

print('Time: ', stop - start)  # STOP TIME

x\_min = -20.0

x\_max = 16.0

mean = 3.0

std = 3.6

x\_plot = np.linspace(x\_min, x\_max)

y\_plot = stats.norm.pdf(x\_plot,mean,std)

cov\_mat = np.stack((x\_list, y\_list))

print(np.cov(cov\_mat))

fig = plt.figure()

plt.hist(a\_list)

plt.plot(x\_plot,3042000\*y\_plot, color='red')

plt.xlabel('Samples Taken')

plt.ylabel('Frquency of Occurance of the Sample')

plt.title('POLAR MARSAGALIA METHOD')

plt.show()

print('--PRACTICAL VALUES----')

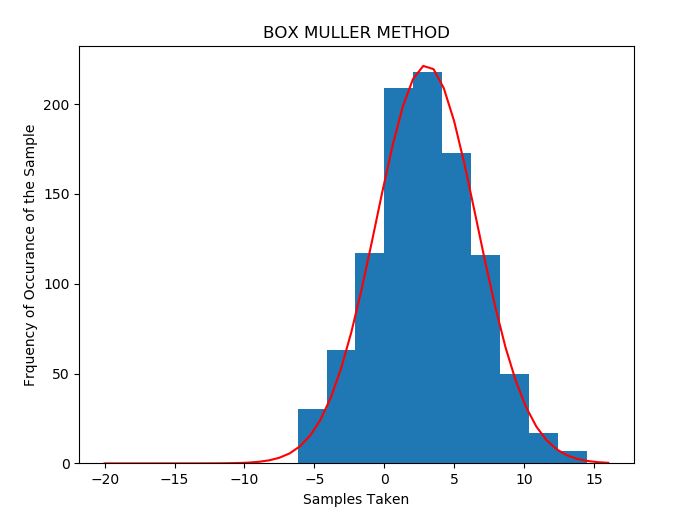
sample\_mean\_practical = statistics.mean(a\_list)

print('sample\_mean\_practical:',sample\_mean\_practical)

sample\_variance\_practical = statistics.variance(a\_list)

print('sample\_variance\_practical:',sample\_variance\_practical)

**Result and Observation:**

**  
Fig.1: Histogram overlayed with theoretical distribution**

**For 1000 samples (Box Muller Method):**

Time:  0.005665599999999937

Covariance Matrix

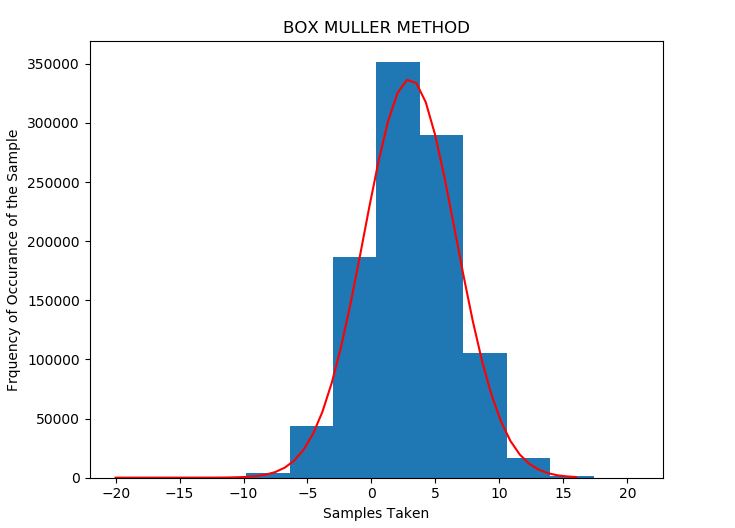
[[3.69120304 0.05531724]

 [0.05531724 9.66981839]]

--PRACTICAL VALUES----

sample\_mean\_practical: 2.899336343633929

sample\_variance\_practical: 13.471655906759493

  
**Fig.2: Histogram overlayed with theoretical distribution**

**For 1,000,000 samples (Box Muller Method):**

Time:  4.4583966

Covariance Matrix

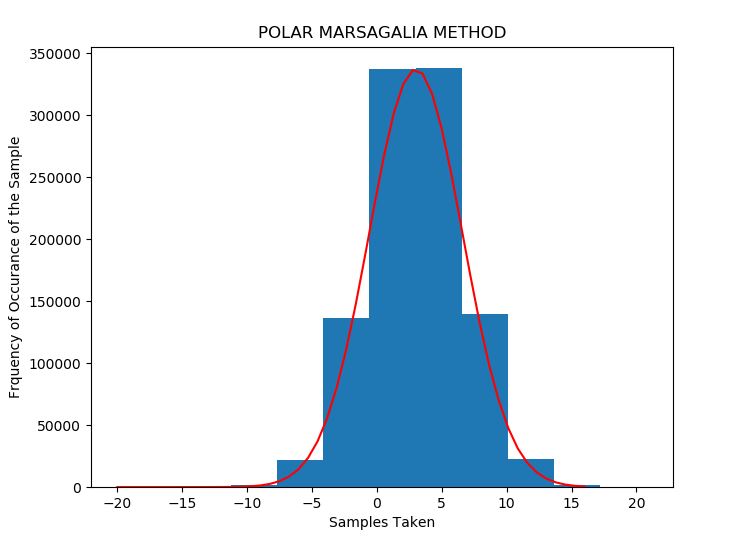
[[ 3.99698746e+00 -4.01092062e-03]

 [-4.01092062e-03  8.98858927e+00]]

--PRACTICAL VALUES----

sample\_mean\_practical: 3.002586460312552

sample\_variance\_practical: 12.977554889442366

**  
Fig.3: Histogram overlayed with theoretical distribution**

**For 1,000,000 samples (Polar Marsaglia Method):**

Time:  4.386356

Covariance Matrix

[[3.99079631e+00 5.73742502e-03]

 [5.73742502e-03 8.99659451e+00]]

--PRACTICAL VALUES----

sample\_mean\_practical: 3.0025396257752583

sample\_variance\_practical: 12.99886567239947

**Repeating the experiment 5 more times for 1,000,000 samples: (Time in Secs)**

|  |  |
| --- | --- |
| **Box Muller Method** | **Polar Marsaglia Method** |
| Time:  4.37816 | Time:  4.237645 |
| Time:  4.580231599999999 | Time:  4.4133192 |
| Time:  4.7033094 | Time:  4.354315199999999 |
| Time:  4.3258602 | Time:  4.3176808 |
| Time:  4.6366715 | Time:  4.3892712000000005 |

Theoretically, we know that the mean and variance of ***A = X + Y*** will be the individual sum of the mean and variance of X and Y. It can be seen that the mean and variance obtained matches the expected values of 3 and 13. Thus, the result obtained is correct. The covariance in both the cases can be seen in the covariance matrix at the (1,2) = (2,1) position. Evidently, it is very small. This suggest that the variables have same trend of behaviour. It is interesting to note that the Covariance reduced considerable when the number of samples are increased. Furthermore, the computational time was calculated for both the methods. It can be seen that both have almost the same computational duration for 1,000,000 samples.

1. **Sampling Gamma Random Variable.**

**Problem Statement:** A gamma random variable ***Gamma(Ɵ,1)***, has to be sampled where ***Ɵ*** is not an integer using Accept-Reject method. Following are the requirements:

1. Histogram for the Samples with the theoretical p.d.f overlayed.
2. Observation on the acceptance rate.

**Understanding the Problem Statement:** For ***Ɵ = 11/2*** the pdf of this Gamma random variable is:

We can use the exponential distribution:

In order to apply the accept-reject method to generate samples  according to , we need to generate samples  from  and accept it with probability

**Program Written:**

import random

import math

import numpy as np

import scipy.stats as stats

import matplotlib.pyplot as plt

import statistics

c\_list = []

sample\_data = []

i=0

j=0

accept = 0

reject = 0

def pdfx(x): # the gamma pdf

    p = 32/(945\*math.sqrt(math.pi))

    q = x\*\*(9/2)

    return p\*q\*math.exp(-x)

def pdfy(y): # the exponenetial pdf

    return (2/11)\*math.exp(-2\*y/11)

while(i<10.0): #generating C

    c = pdfx(i)/pdfy(i)

    c\_list.append(c)

    i+=0.01

C = max(c\_list)

for j in range (1000):

    Y = random.expovariate(2/11)

    u = np.random.rand()

    value\_test = C\*u

    if value\_test <= pdfx(Y)/pdfy(Y): #the accept reject algorithm

        sample\_data.append(Y)

        accept += 1

    else:

        reject += 1

x= np.arange(0,15)

fig = plt.figure()

plt.hist(sample\_data)

plt.plot(550\*(stats.gamma.pdf(x,5.5,1)))

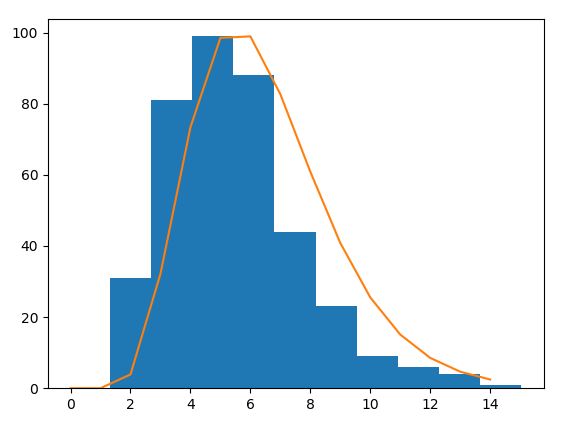
plt.show()

print('--ACCEPTANCE RATE---')

rate = accept/ (accept + reject)

print(rate)

**Result and Observation:**

**  
Fig.4: Histogram overlayed with theoretical distribution (X: Samples, Y:Frequency)**

--ACCEPTANCE RATE---

0.391

It can be seen that the fit is almost perfect and the acceptance rate is around 40%.

1. **Sampling alpha-stable random variable:**

**Problem Statement:** It is required to sample from a symmetric alpha stable pdfs using 4 different values of α = {0.5,1,1.8,2.0}. Following are the requirements:

1. Histogram and time series plot for each alpha
2. Repeat the experiment for β = 0.75

**Understanding Problem Statement:** *Taken from Piazza*- *The alpha-stable distribution is a four-parameter family of distributions and is (usually) denoted by  http://math.bu.edu/people/mveillet/html/alphastablepub_eq07061.png. The first parameter  http://math.bu.edu/people/mveillet/html/alphastablepub_eq67893.png  is called the characteristic exponent, and describes the tail of the distribution. The second, http://math.bu.edu/people/mveillet/html/alphastablepub_eq32153.png is the skewness, and as the name implies, specifies if the distribution is right- (http://math.bu.edu/people/mveillet/html/alphastablepub_eq76915.png) or left- (http://math.bu.edu/people/mveillet/html/alphastablepub_eq75308.png) skewed. The last two parameters are the scale, http://math.bu.edu/people/mveillet/html/alphastablepub_eq08866.png, and the location http://math.bu.edu/people/mveillet/html/alphastablepub_eq88041.png. One can think of these two as being similar to the variance and mean in the normal distribution in the following sense - if http://math.bu.edu/people/mveillet/html/alphastablepub_eq27343.png, then if http://math.bu.edu/people/mveillet/html/alphastablepub_eq47709.png,  http://math.bu.edu/people/mveillet/html/alphastablepub_eq68478.png. The variable http://math.bu.edu/people/mveillet/html/alphastablepub_eq14833.png is usually called a standard alpha-stable random variable (but keep in mind the word "standard" depends on the choice of parameterization!).* The sample code provided by Prof Brandon in MATLAB provides a function that generates random numbers from levy alpha-stable random numbers for specified alpha values. The same used along with a inbuilt function to overlay it.

**Program Written:**

X = stblrnd(2,1,0,1000,1);

hist(X);

hold on

x = -5:.1:5;

pd\_levy = makedist('Stable','alpha',2,'beta',1,'gam',1,'delta',0);

pdf\_levy = pdf(pd\_levy,x);

xlabel('samples')

ylabel('magnitude')

plot(x,1000\*pdf\_levy,'r');

%------ CODE TAKEN FORM PROF BRANDON'S TUTORIAL--------

function r = stblrnd(alpha,beta,gamma,delta,varargin)

%STBLRND alpha-stable random number generator.

% R = STBLRND(ALPHA,BETA,GAMMA,DELTA) draws a sample from the Levy

% alpha-stable distribution with characteristic exponent ALPHA,

% skewness BETA, scale parameter GAMMA and location parameter DELTA.

% ALPHA,BETA,GAMMA and DELTA must be scalars which fall in the following

% ranges :

% 0 < ALPHA <= 2

% -1 <= BETA <= 1

% 0 < GAMMA < inf

% -inf < DELTA < inf

%

%

% R = STBLRND(ALPHA,BETA,GAMMA,DELTA,M,N,...) or

% R = STBLRND(ALPHA,BETA,GAMMA,DELTA,[M,N,...]) returns an M-by-N-by-...

% array.

%

%

% References:

% [1] J.M. Chambers, C.L. Mallows and B.W. Stuck (1976)

% "A Method for Simulating Stable Random Variables"

% JASA, Vol. 71, No. 354. pages 340-344

%

% [2] Aleksander Weron and Rafal Weron (1995)

% "Computer Simulation of Levy alpha-Stable Variables and Processes"

% Lec. Notes in Physics, 457, pages 379-392

%

if nargin < 4

error('stats:stblrnd:TooFewInputs','Requires at least four input arguments.');

end

% Check parameters

if alpha <= 0 || alpha > 2 || ~isscalar(alpha)

error('stats:stblrnd:BadInputs',' "alpha" must be a scalar which lies in the interval (0,2]');

end

if abs(beta) > 1 || ~isscalar(beta)

error('stats:stblrnd:BadInputs',' "beta" must be a scalar which lies in the interval [-1,1]');

end

if gamma < 0 || ~isscalar(gamma)

error('stats:stblrnd:BadInputs',' "gamma" must be a non-negative scalar');

end

if ~isscalar(delta)

error('stats:stblrnd:BadInputs',' "delta" must be a scalar');

end

% Get output size

[err, sizeOut] = genOutsize(4,alpha,beta,gamma,delta,varargin{:});

if err > 0

error('stats:stblrnd:InputSizeMismatch','Size information is inconsistent.');

end

%---Generate sample----

% See if parameters reduce to a special case, if so be quick, if not

% perform general algorithm

if alpha == 2 % Gaussian distribution

r = sqrt(2) \* randn(sizeOut);

elseif alpha==1 && beta == 0 % Cauchy distribution

r = tan( pi/2 \* (2\*rand(sizeOut) - 1) );

elseif alpha == .5 && abs(beta) == 1 % Levy distribution (a.k.a. Pearson V)

r = beta ./ randn(sizeOut).^2;

elseif beta == 0 % Symmetric alpha-stable

V = pi/2 \* (2\*rand(sizeOut) - 1);

W = -log(rand(sizeOut));

r = sin(alpha \* V) ./ ( cos(V).^(1/alpha) ) .\* ...

( cos( V.\*(1-alpha) ) ./ W ).^( (1-alpha)/alpha );

elseif alpha ~= 1 % General case, alpha not 1

V = pi/2 \* (2\*rand(sizeOut) - 1);

W = - log( rand(sizeOut) );

const = beta \* tan(pi\*alpha/2);

B = atan( const );

S = (1 + const \* const).^(1/(2\*alpha));

r = S \* sin( alpha\*V + B ) ./ ( cos(V) ).^(1/alpha) .\* ...

( cos( (1-alpha) \* V - B ) ./ W ).^((1-alpha)/alpha);

else % General case, alpha = 1

V = pi/2 \* (2\*rand(sizeOut) - 1);

W = - log( rand(sizeOut) );

piover2 = pi/2;

sclshftV = piover2 + beta \* V ;

r = 1/piover2 \* ( sclshftV .\* tan(V) - ...

beta \* log( (piover2 \* W .\* cos(V) ) ./ sclshftV ) );

end

% Scale and shift

if alpha ~= 1

r = gamma \* r + delta;

else

r = gamma \* r + (2/pi) \* beta \* gamma \* log(gamma) + delta;

end

end

%==== function to find output size ======%

function [err, commonSize, numElements] = genOutsize(nparams,varargin)

try

tmp = 0;

for argnum = 1:nparams

tmp = tmp + varargin{argnum};

end

if nargin > nparams+1

tmp = tmp + zeros(varargin{nparams+1:end});

end

err = 0;

commonSize = size(tmp);

numElements = numel(tmp);

catch

err = 1;

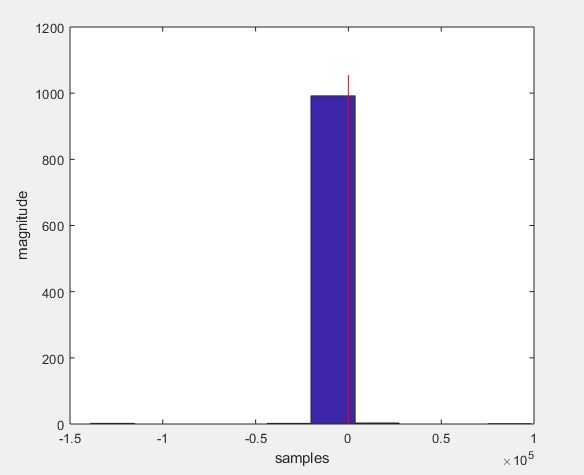
commonSize = [];

numElements = 0;

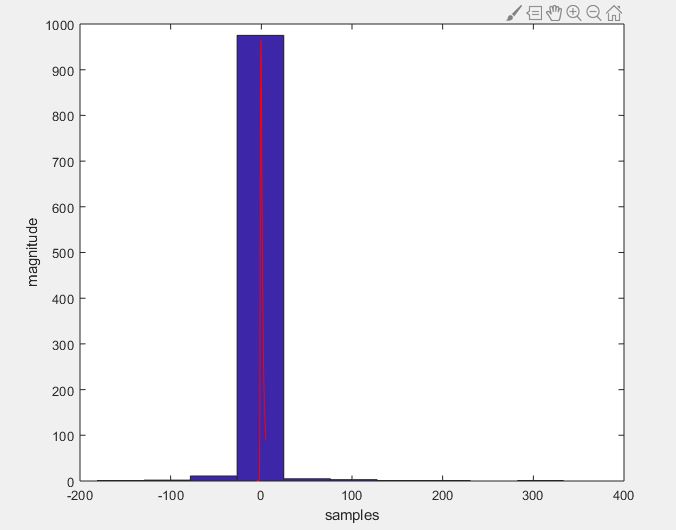
end

end

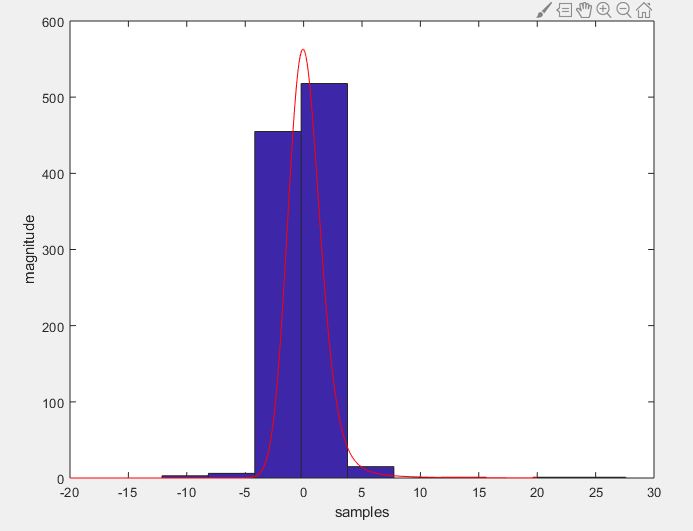
**Results and Observation:**

**  
Fig.5. alpha = 0.5 with levy stable overlayed in red**

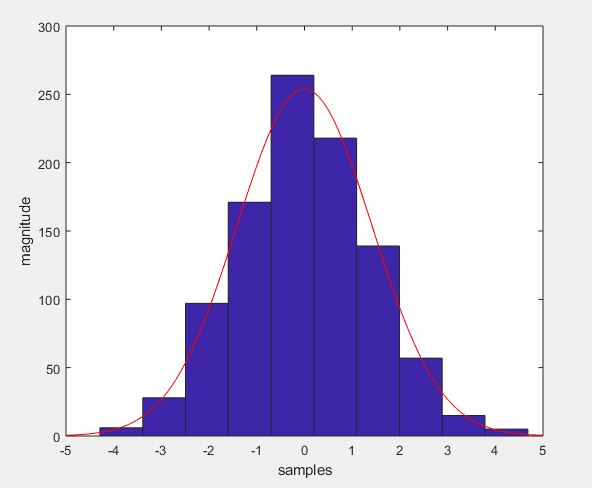
It can be seen that the tail of the distribution of the overlayed distribution is very steep. The same can be observed that the histogram just have one dominant value at 0.

  
**Fig.6. alpha = 1 with levy stable overlayed in red**

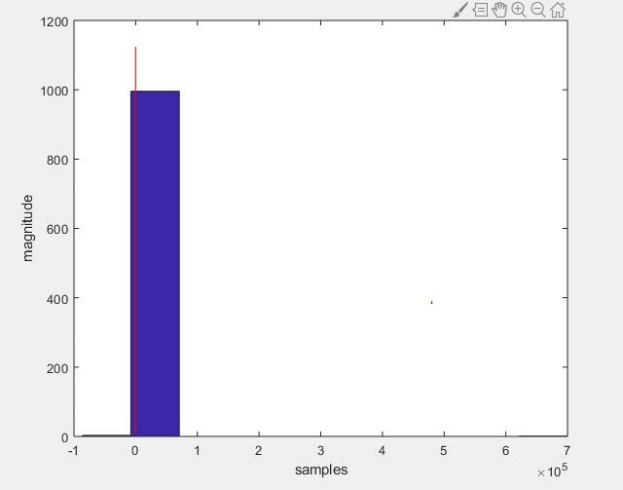
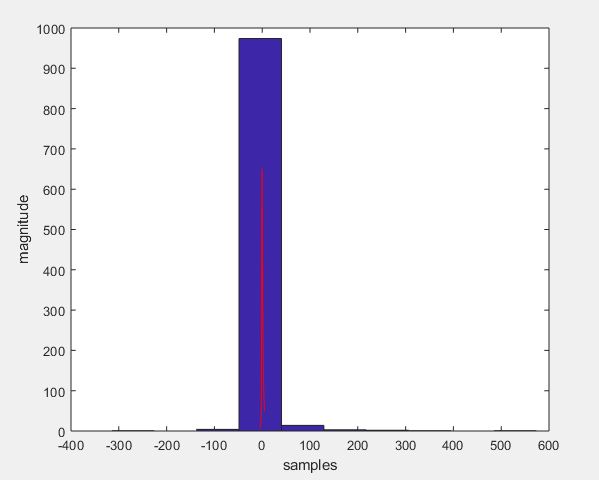
It can be seen that the tail spreads out a little as we increase the value of alpha. Let’s see for alpha = 1.5.

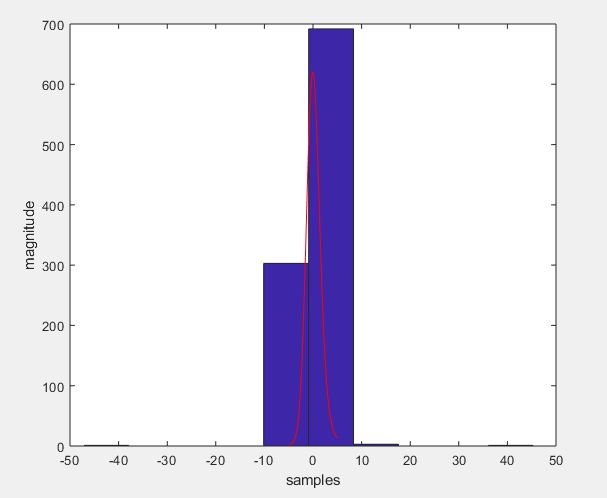
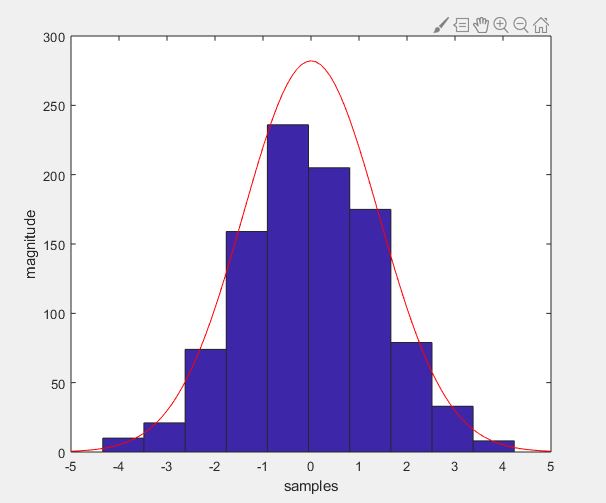
  
**Fig.7. alpha = 1.8 with levy stable overlayed in red**

As expected the curve spreads out more and so does the histogram.

  
**Fig.8. alpha = 2 with levy stable overlayed in red**

Finally for alpha = 2, it can be seen that the spread is the maximum. Now the experiment was repeated with β = 0.75 and we can see the skew to right-skewed alpha distribution as follows:

   
 **Fig.9. alpha = 0.5, beta = 0.75 Fig.10. alpha = 1, beta = 0.75**

   
 **Fig.9. alpha = 1.8, beta = 0.75 Fig.10. alpha = 2, beta = 0.75**