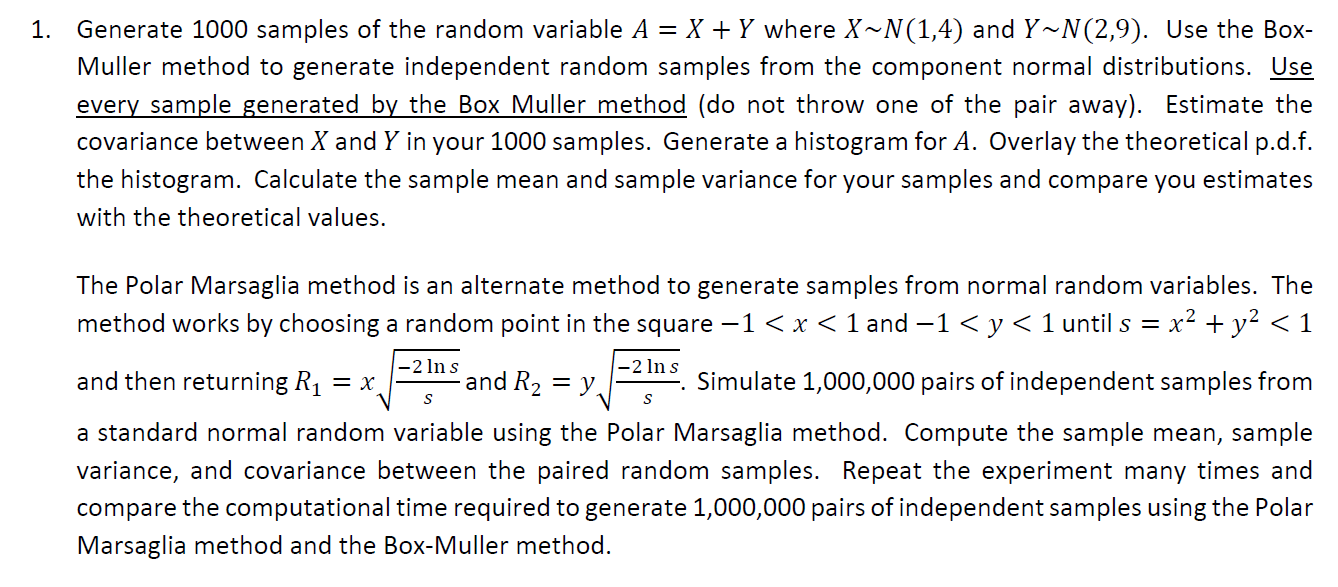
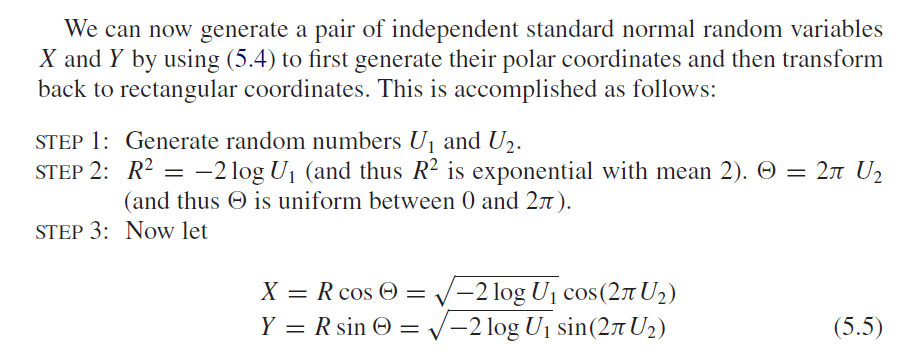
Project #6 – Continuous sampling

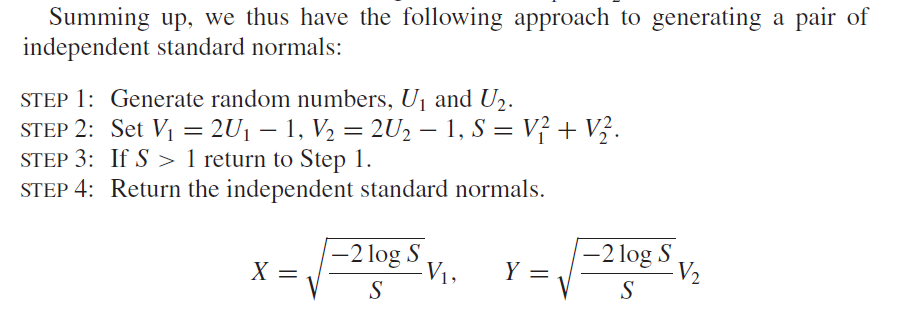
**Concept behind this problem:**

The question requires the knowledge of ***Box Muller Method*** and ***Polar Marsaglia Method***.

**Box-Muller sampling** is based on representing the joint distribution of two independent standard Normal random Cartesian variables X and Y. It is a pseudo-random number sampling method for generating pairs of independent, standard, normally distributed (zero expectation, unit variance) random numbers, given a source of uniformly distributed random numbers. It was developed as a more computationally efficient alternative to the inverse transform sampling method. Also, it can be employed for drawing from truncated bivariate Gaussian densities. This method differs from the basic method in that it is a type of rejection sampling. It discards some generated random numbers, but it is typically faster than the basic method because it is simpler to compute (provided that the random number generator is relatively fast) and is more numerically robust. It avoids the use of trigonometric functions, which are comparatively expensive in many computing environments.



**Polar Marsaglia Method** is a pseudo-random number sampling method for generating a pair of independent standard normal random variables. It is superior to Box Muller Method. One difference between Box Muller and Polar Marsaglia is that the former takes Polar form while the later deals with Cartesian Forms.



**Reference: Piazza**

**Code & Code Description:**

* Initialize variables and here number of samples and number of trails are provided by user input. Also, the question has variance and mean of X and Y.

%Initialize (Given values)

mean\_of\_X = 1;

mean\_of\_Y = 2;

var\_of\_X = 4;

var\_of\_Y = 9;

theo\_mean\_sum = mean\_of\_X + mean\_of\_Y;

theo\_var\_sum = var\_of\_X + var\_of\_Y;

* In both the cases, we start off with a tic timer to keep track of timer and end the storage of timer with a tac.
* We generate two random samples as mentioned in the algorithm and calculate the independent random variables X and Y and scale the random variables generated as per the mean.

%\*\*\*\* Code Reference from Piazza \*\*\*\*

%Box Muller Method

time\_elasped=zeros(1,No\_of\_trails);

for i=1:No\_of\_trails

tic;% Start the timer

r\_number1 = rand(No\_Of\_Samples,1);

r\_number2 = rand(No\_Of\_Samples,1);

% Generate X and Y that are N(0,1) random variables and independent

rand\_var\_X = sqrt( - 2\*log(r\_number1)).\*cos(2\*pi\*r\_number2);

rand\_var\_Y = sqrt( - 2\*log(r\_number1)).\*sin(2\*pi\*r\_number2);

% Scale them to a particular mean and variance

x = sqrt(var\_of\_X)\*rand\_var\_X + mean\_of\_X;

y = sqrt(var\_of\_Y)\*rand\_var\_Y + mean\_of\_Y;

time\_elasped(i)=toc; % Read elapsed time from stopwatch

sum=x+y;

end

cov\_x\_and\_y = cov(x,y)

disp('Summary of Simulation for BOX MULLER Method');

disp(['Calculated Mean ',num2str(mean(sum)),' Theoritical Mean ',num2str(theo\_mean\_sum)]);

disp(['Calculated Variance ',num2str(var(sum)),' Theoritical Variance ',num2str(theo\_var\_sum)]);

disp(['Sample Mean of Random Variable X is ',num2str(mean(x)),' and that of Y is ',num2str(mean(y))]);

disp(['Sample Variance of Random Variable X is ',num2str(var(x)),' and that of Y is ',num2str(var(y))]);

disp(['Covariance of Random Variables X and Y is ',num2str(cov\_x\_and\_y(1,2))]);

disp(['Time taken to generate ',num2str(No\_Of\_Samples),' samples with ',num2str(No\_of\_trails),' experiments are ',num2str(time\_elasped)]);

figure(1)

yyaxis left

hist(sum,20);

title('BOX MULLER: Random Variable X+Y overlay with PDF')

xlabel('Value of sum of Random variables')

ylabel('Frequency')

yyaxis right

t=-80:0.1:80;

theo\_pdf=normpdf(t,3,13);

plot(t,theo\_pdf,'m--\*');

ylabel('Theo Normal PDF');

* We then calculate mean, variance of X and Y followed by covariance of X and Y for each method and finally overlay the Theoritical PDF over the histogram generated by individual method.
* We then compare the performance of Box Muller method with Polar Marsaglia (calculated by tic toc in built functions)

%POLAR MARSAGLIA Method

j=0;

for i=1:No\_of\_trails

tic;% Start the timer

while(j<=No\_Of\_Samples)

r\_number1 = 2\*rand()-1;

r\_number2 = 2\*rand()-1;

s = r\_number1^2 + r\_number2^2;

if(s < 1)

j = j + 1;

X(j) = sqrt(-2\*log(s)/s)\*r\_number1;

Y(j) = sqrt(-2\*log(s)/s)\*r\_number2;

end

end

x = sqrt(var\_of\_X)\*X + mean\_of\_X;

y = sqrt(var\_of\_Y)\*Y + mean\_of\_Y;

time\_elasped(i)=toc;% Read elapsed time from stopwatch

sum=x+y;

end

cov\_x\_and\_y = cov(x,y)

disp('Summary of Simulation for POLAR MARSAGLIA Method');

disp(['Calculated Mean ',num2str(mean(sum)),' Theoritical Mean ',num2str(theo\_mean\_sum)]);

disp(['Calculated Variance ',num2str(var(sum)),' Theoritical Variance ',num2str(theo\_var\_sum)]);

disp(['Sample Mean of Random Variable X is ',num2str(mean(x)),' and that of Y is ',num2str(mean(y))]);

disp(['Sample Variance of Random Variable X is ',num2str(var(x)),' and that of Y is ',num2str(var(y))]);

disp(['Covariance of Random Variables X and Y is ',num2str(cov\_x\_and\_y(1,2))]);

disp(['Time taken to generate ',num2str(No\_Of\_Samples),' samples with ',num2str(No\_of\_trails),' experiments are ',num2str(time\_elasped)]);

figure(2)

yyaxis left

hist(sum,20);

title('POLAR MARSAGLIA MULLER: Random Variable X+Y overlay with PDF')

xlabel('Value of Random variable A')

ylabel('Frequency')

yyaxis right

t=-80:0.1:80;

theo\_pdf=normpdf(t,3,13);

plot(t,theo\_pdf,'c--o');

ylabel('Theo Normal PDF');

**Results and Theoretical Values:**

**Number of samples = 1000 and Number of experiment run = 1**

>> ee511\_p6\_q1(1000,1)

Summary of Simulation for BOX MULLER Method

Calculated Mean 3.2512 Theoritical Mean 3

Calculated Variance 12.6068 Theoritical Variance 13

Sample Mean of Random Variable X is 1.1011 and that of Y is 2.1501

Sample Variance of Random Variable X is 3.7754 and that of Y is 9.0681

Covariance of Random Variables X and Y is -0.11836

Time taken to generate 1000 samples with 1 experiments are 0.00031665

Summary of Simulation for POLAR MARSAGLIA Method

Calculated Mean 3.123 Theoritical Mean 3

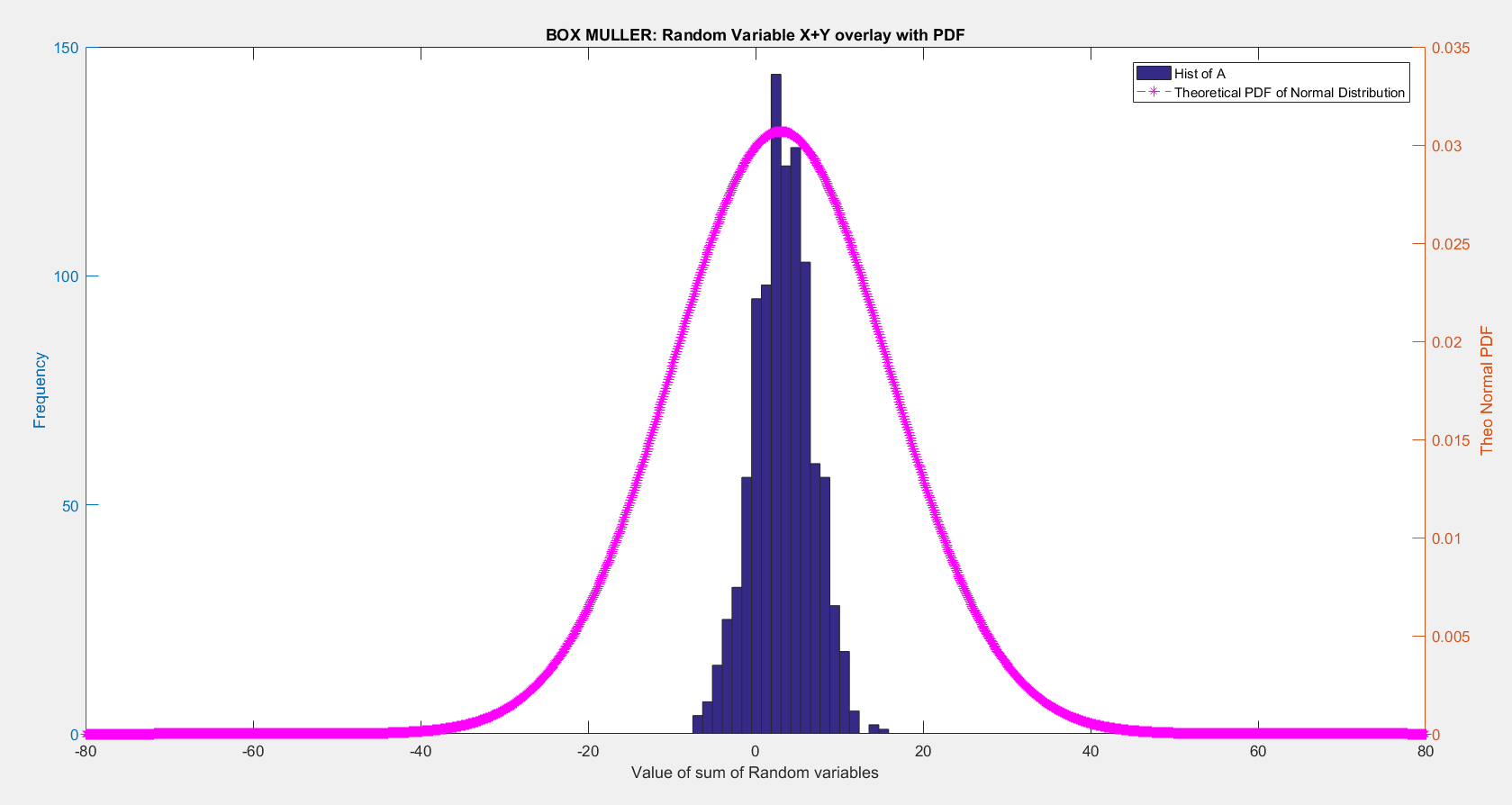
Calculated Variance 13.0521 Theoritical Variance 13

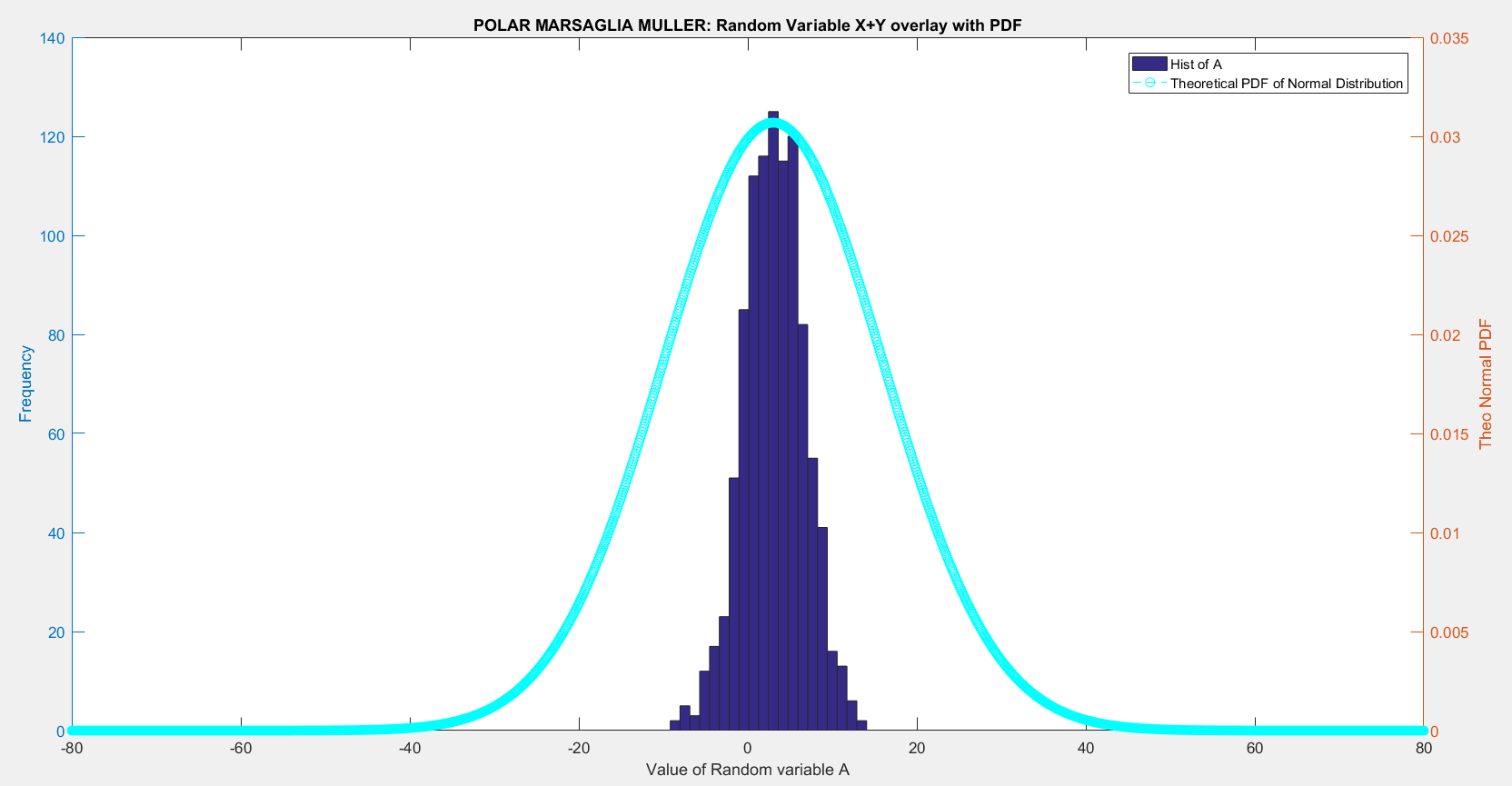
Sample Mean of Random Variable X is 1.1395 and that of Y is 1.9836

Sample Variance of Random Variable X is 3.9126 and that of Y is 9.7516

Covariance of Random Variables X and Y is -0.30606

Time taken to generate 1000 samples with 1 experiments are 0.00062836





**Number of samples = 1,000,000 and Number of experiment run = 10**

>> ee511\_p6\_q1(1000000,10)

Summary of Simulation for BOX MULLER Method

Calculated Mean 3.003 Theoritical Mean 3

Calculated Variance 12.9898 Theoritical Variance 13

Sample Mean of Random Variable X is 1.0037 and that of Y is 1.9992

Sample Variance of Random Variable X is 3.9994 and that of Y is 8.9968

Covariance of Random Variables X and Y is -0.0032036

Time taken to generate 1000000 samples with 10 experiments are 0.099363 0.11289 0.098821 0.098969 0.09177 0.094241 0.09695 0.095641 0.0951 0.092478

Summary of Simulation for POLAR MARSAGLIA Method

Calculated Mean 2.9985 Theoritical Mean 3

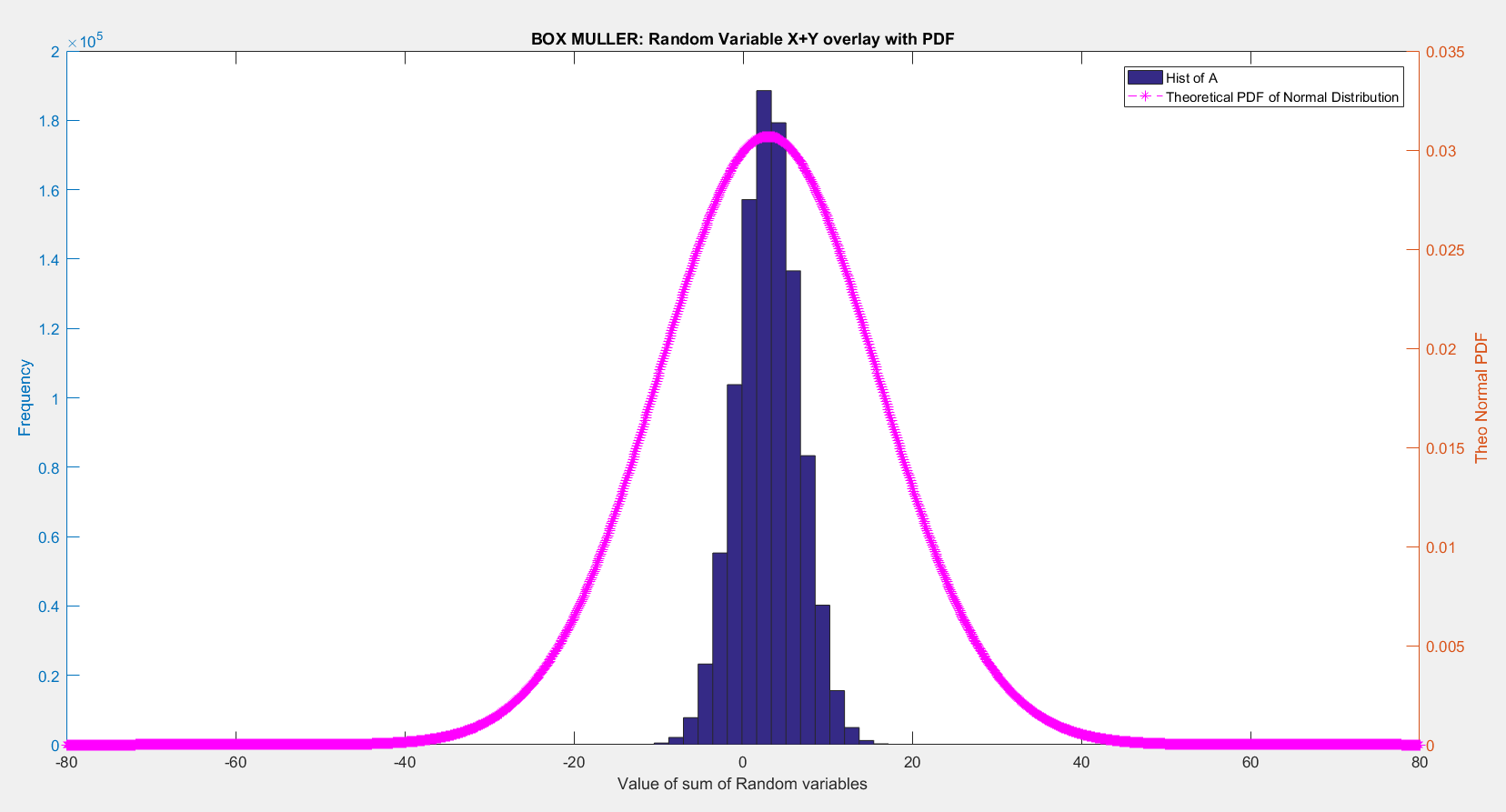
Calculated Variance 12.998 Theoritical Variance 13

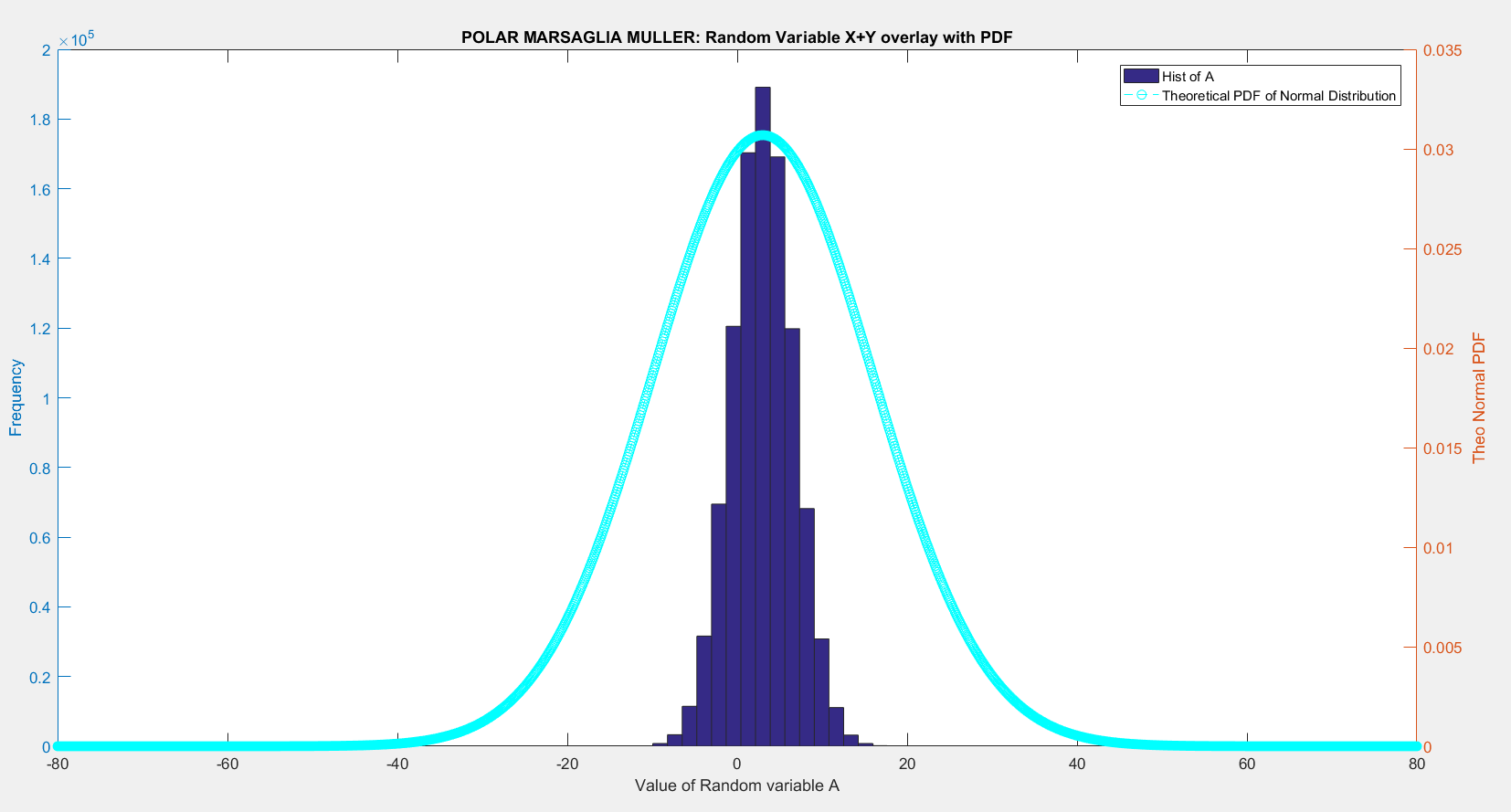
Sample Mean of Random Variable X is 0.99792 and that of Y is 2.0006

Sample Variance of Random Variable X is 4.0019 and that of Y is 8.9809

Covariance of Random Variables X and Y is 0.0076115

Time taken to generate 1000000 samples with 10 experiments are 0.36922 0.0037571 0.003347 0.0031383 0.0030706 0.0032145 0.003249 0.0031111 0.0032589 0.0030635

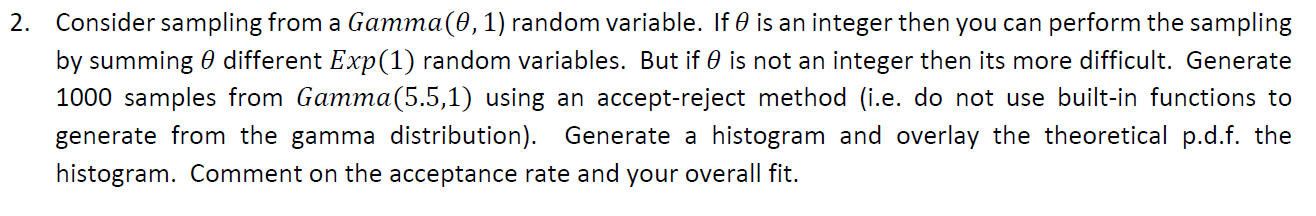




**Observations:**

* The theoretical mean, variance of X+Y random Variable is calculated using both Box Muller and Polar Marsaglia method and they are almost same as that of theoretical values.
* Covariance of X and Y are calculated as well using both the methods.
* The normal theoretical PDF is overlaid over the samples generated via both method and we find that the histogram almost fits within the curve.
* The time required in generating ‘N’ pairs of independent samples gives us a complete picture of the efficiency of Polar Marsaglia method over Box Muller method.
* We can thus see that the performance of Polar Marsaglia method is much better than Box muller method. Also, we see that the complexity of Polar Marsaglia method is less compared to Box Muller method in which the former avoids the evaluation of trigonometric functions.

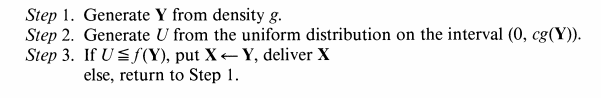
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*



**Concept behind this problem:**

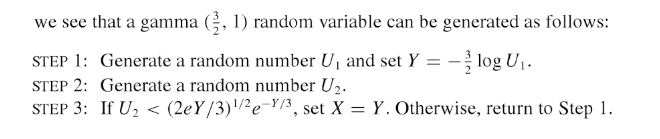
The problem uses the concept of **Acceptance Rejection Method**.

Acceptance Rejection Method or accept-reject algorithm is a type of Monte Carlo method. The rejection sampling method generates sampling values from a target distribution X with arbitrary probability density function f(x) by using a proposal distribution Y with probability density g(x). The idea is that one can generate a sample value from X by instead sampling from Y and accepting the sample from Y with probability f(x)/cg(x), repeating the draws from Y until a value is accepted. c here is a constant, finite bound on the likelihood ratio f(x)/g(x), satisfying 1<c<infinity over the support of X. To perform Acceptance Rejection method, we need to perform the below



**Reference:**

**Chp-5 Example 5e from Simulation 5th Edition by Sheldon Ross**



**Code and Description:**

* My implementation uses trying out acceptance rejection method for exponential distribution since it suits best to try it out for a given Gamma function. Hence f(x) is Gamma while g(x) is Exponential function having the same mean as the gamma.
* We first obtain the pdf of Gamma and Exponential function i.e. of f(x) and g(x).
* Using these values, we calculate the constant c such that c\*g(x)>f(x).
* As mentioned above, we calculate a random Variable and iterate it so that its value is within the threshold and set X=Y in case of success.
* After the entire process, we calculate the theoPDF of Gamma (5.5) and overlay it over the histogram of the generated samples.
* Finally, we calculate the Acceptance Rate.

%Initialize Variables

rej\_samp = 0;

value = 11/2;

x = 0:0.05:10;%to make histogram clearer on x-axis

%Calculate the value of C so that u<=f(y)/c\*g(y)

% function f

func\_X = @(x) 1/gamma(value) .\* x.^(value-1) .\* exp(-x);

% function g

func\_Y = @(y) (1/value) \* exp(-y/value);

X\_by\_Y = func\_X(x)./func\_Y(x);

const\_c = max(X\_by\_Y);

c\_value=(value/(const\_c\*gamma(5.5)));

for iter=1:No\_of\_samples

while(1)

r\_number=rand;

uniform\_i=-value\*log(r\_number);

if rand<((c\_value)\*(uniform\_i^(value-1))\*exp(-uniform\_i\*(1-(1/value))))

accept(iter)=uniform\_i;

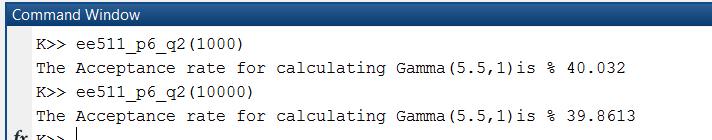
break;

else

rej\_samp = rej\_samp+1;

end end end

**Results and Plots:**



yyaxis left;

hist(accept,15);

title('Overlaying of theoretical pdf and the histogram of the generated samples');

xlabel('Accept Values');

ylabel('Frequency of the generated samples');

yyaxis right;

x=0:0.1:40;

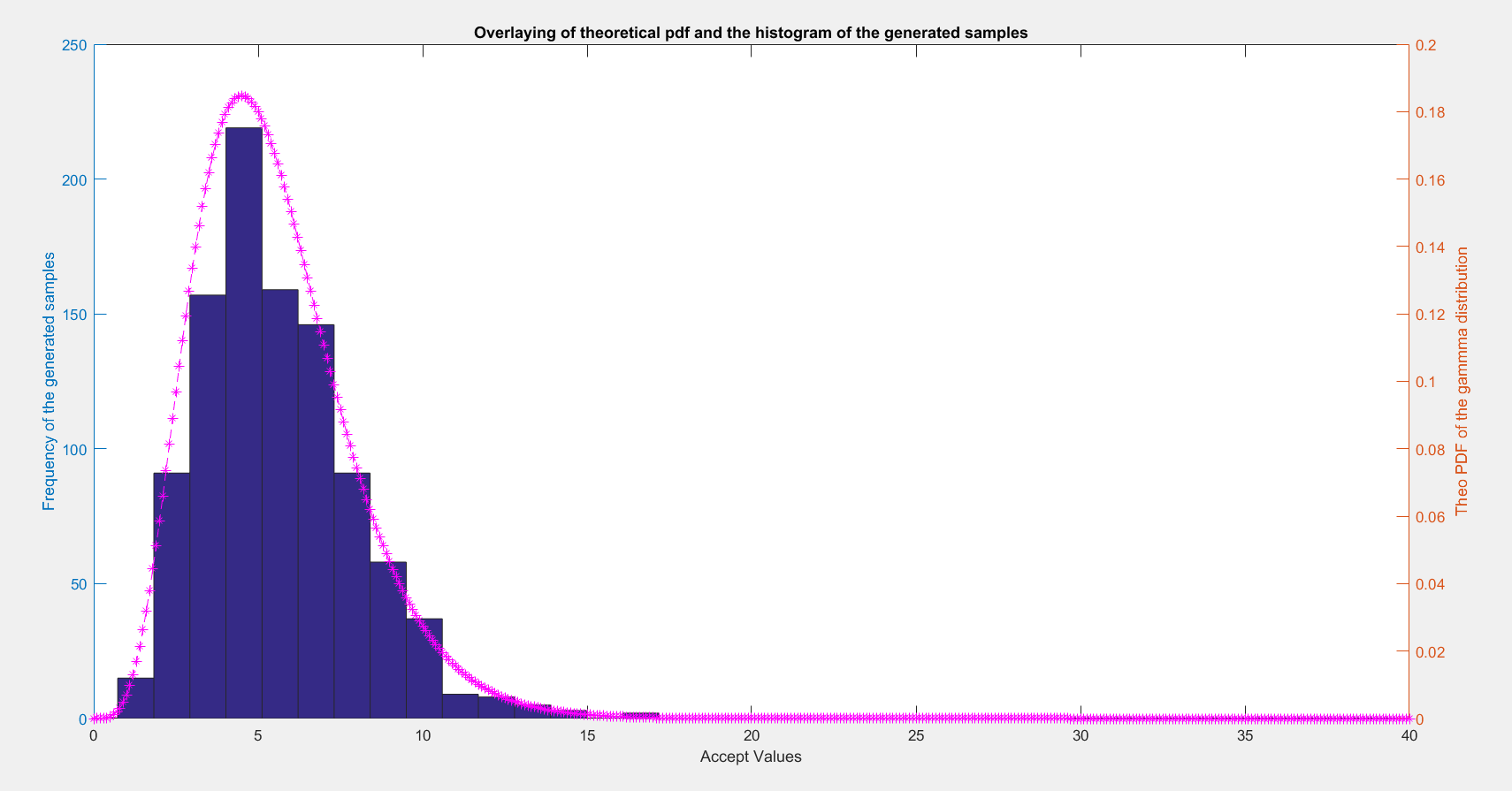
theo\_gamma=gampdf(x,value,1);

plot(x,theo\_gamma,'m--\*');

ylabel('Theo PDF of the gammma distribution');

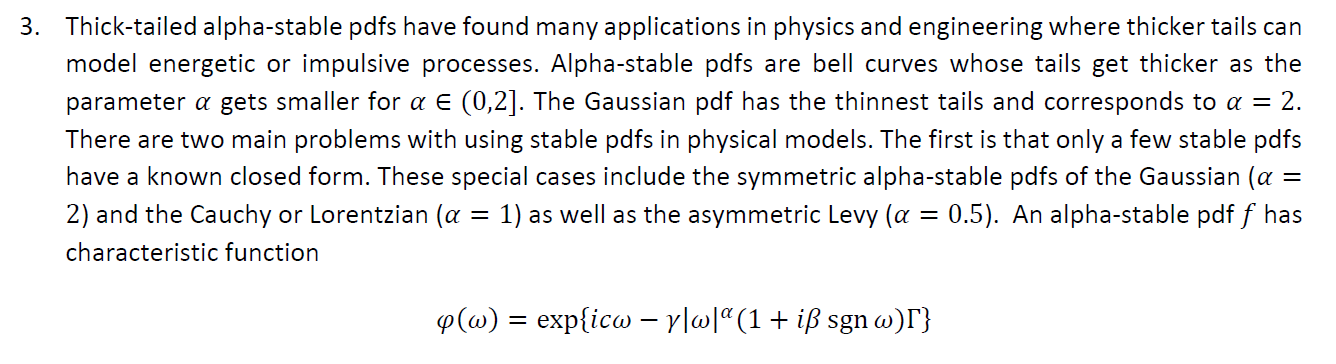
accept\_rate=(No\_of\_samples/(rej\_samp+No\_of\_samples))\*100;

disp(['The Acceptance rate for calculating Gamma(5.5,1)is % ',num2str(accept\_rate)]);

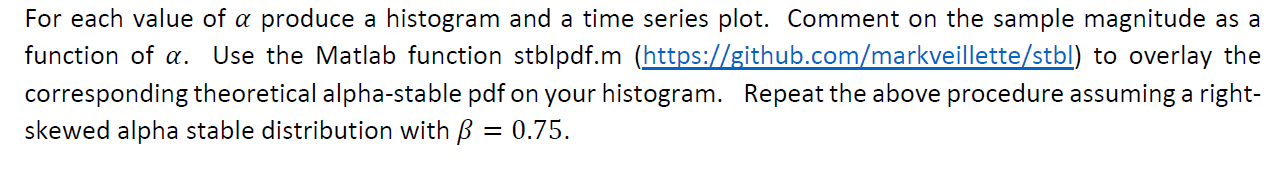


**Observations:**

* The samples plotted in histogram lie within the theoretical PDF (except for few cases which go outside the theo PDF).
* The disadvantage of the rejection method is that it is difficult to find a good hat function g(x) which is easy to calculate and at the same time approximates f(x) so good that the rejection rate will be low. A bad hat function will lead to a high rejection rate or less acceptance rate as in our case.
* Using number of samples as 1000 or more than that, we always get the acceptance rate of 40%. Since the acceptance rate of is very less, the Accept-reject method might not be suitable for generating samples for a Gamma Distribution.
* The current method can lead to a lot of unwanted samples being taken if the function being sampled is highly concentrated in a certain region, for example a function that has a spike at some location. Hence for many distributions, this problem maybe solved using an adaptive extension. Furthermore, as the dimensions of the problem get larger, a lot of rejections can take place before a useful sample is generated, thus making the algorithm inefficient and impractical.



…..



**Reference:**

**[1] Piazza**

**[2] Chambers, John M., Colin L. Mallows, and B. W. Stuck. "A method for simulating stable random variables." Journal of the american statistical association 71.354 (1976): 340‐344.**

**[3] Weron, Rafał. "On the Chambers‐Mallows‐Stuck method for simulating skewed stable random variables." Statistics & probability letters 28.2 (1996): 165‐171.**

**Concept behind this problem:**

The problem uses the concept of ***Alpha Stable Distribution***

The **alpha-stable distribution** is a four-parameter family of distributions and is (usually) denoted by $S(\alpha,\beta,\gamma,\delta)$. The first parameter $\alpha \in (0,2]$ is called the *characteristic exponent*, and describes the tail of the distribution. The second, $\beta \in [-1,1]$ is the *skewness*, and as the name implies, specifies if the distribution is right- ($\beta > 0$) or left- ($\beta < 0$) skewed. The last two parameters are the *scale*, $\gamma > 0$, and the *location* $\delta \in R$. One can think of these two as being similar to the variance and mean in the normal distribution in the following sense - if $Z \sim S(\alpha,\beta,1,0)$, then if $\alpha \neq 1$, $\gamma Z + \delta \sim S(\alpha,\beta,\gamma,\delta)$. The variable $Z$ is usually called a *standard* alpha-stable random variable.

Since the normal distribution, the Cauchy distribution, and the Lévy distribution all have the above property, it follows that they are special cases of stable distributions. In probability theory, a distribution or a random variable is said to be stable if a linear combination of two independent copies of a random sample has the same distribution, up to location and scale parameters. With the exception of the above 3 cases, the density function of an alpha-stable random variable cannot be given in closed form. However, the characteristic function can always be given. Let . The parameterization we use here corresponds to the following form of the characteristic function. For α ≠1, we have

http://math.bu.edu/people/mveillet/html/alphastablepub_eq01861.png.

and for α =1,

http://math.bu.edu/people/mveillet/html/alphastablepub_eq67165.png.

**Code & Code Description:**

* In general, if a random variable is obtained in closed form, we could have used Inverse Transform method which would ease the implementation.
* However, in this case, the random variables are not in closed form and hence to generate samples, we use a method called Chambers‐Mallows‐Stuck method which describes how to generate samples from an arbitrary alpha stable distribution
* Variables are initialized.
* **First step** is to use STBLRND (an alpha-stable random number generator) which draws a sample from the Levy alpha-stable distribution with characteristic exponent Alpha, % skewness Beta, scale parameter Gamma and location parameter Delta.
* This produces a characteristic of stable distributions, and is the reason they are useful. We see here a lot of samples falling near the origin, and a few rare samples which are large in magnitude.

function [] = ee511\_p6\_q3(alpha,beta,gamma,delta,varargin)

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

%==== 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

The **second step** is to generate pdf of the stable distribution with characteristic exponent Alpha, skewness Beta, scale % parameter Gamma, and location parameter Delta, at the values in X using the STBLPDF. For doing this, We use the parameterization of stable distribution used in- The characteristic function phi(t) of a S (Alpha, Beta, Gamma, Delta).

* The STBLPDF skips the step of locating the peak in the integrand, and thus is faster, but is less accurate deep into the tails of the pdf. This option is useful for plotting. In place of 'quick', STBLPDF also excepts a logical true or false

if nargin < 5

error('stblpdf:TooFewInputs','Requires at least five input arguments.');

end

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

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

end

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

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

end

if gam < 0 || ~isscalar(gam)

error('stblpdf:BadInputs',' "gam" must be a non-negative scalar');

end

if ~isscalar(delta)

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

end

% Warn if alpha is very close to 1 or 0

if ( 1e-5 < abs(1 - alpha) && abs(1 - alpha) < .02) || alpha < .02

warning('stblpdf:ScaryAlpha',...

'Difficult to approximate pdf for alpha close to 0 or 1')

end

% Check and initialize additional inputs

quick = false;

tol = [];

for i=1:length(varargin)

if strcmp(varargin{i},'quick')

quick = true;

elseif islogical(varargin{i})

quick = varargin{end};

elseif isscalar(varargin{i})

tol = varargin{i};

end end

if isempty(tol)

if quick

tol = 1e-8;

else

tol = 1e-12;

end end

%Compute pdf:Check to see if you are in a simple case, if so be quick, if not do general algorithm

if alpha == 2 % Gaussian distribution

x = (x - delta)/gam; % Standardize

p = 1/sqrt(4\*pi) \* exp( -.25 \* x.^2 ); % ~ N(0,2)

p = p/gam; %rescale

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

x = (x - delta)/gam; % Standardize

p = (1/pi) \* 1./(1 + x.^2);

p = p/gam; %rescale

elseif alpha == .5 && abs(beta) == 1 % Levy distribution

x = (x - delta)/gam; % Standardize

p = zeros(size(x));

if beta ==1

p( x <= 0 ) = 0;

p( x > 0 ) = sqrt(1/(2\*pi)) \* exp(-.5./x(x>0)) ./...

x(x>0).^1.5;

else

p(x >= 0) = 0;

p(x < 0 ) = sqrt(1/(2\*pi)) \* exp(.5./x(x<0) ) ./...

( -x(x<0) ).^1.5;

end

p = p/gam; %rescale

elseif abs(alpha - 1) > 1e-5 % Gen. Case, alpha ~= 1

xold = x; % Save for later

% Standardize in (M) parameterization ( See equation (2) in [1] )

x = (x - delta)/gam - beta \* tan(alpha\*pi/2);

% Compute pdf

p = zeros(size(x));

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

theta0 = (1/alpha) \* atan(beta\*tan(pi\*alpha/2));

A1 = alpha\*theta0;

A2 = cos(A1)^(1/(alpha-1));

exp1 = alpha/(alpha-1);

alpham1 = alpha - 1;

c2 = alpha ./ (pi \* abs(alpha - 1) \* ( x(x>zeta) - zeta) );

V = @(theta) A2 \* ( cos(theta) ./ sin( alpha\*(theta + theta0) ) ).^exp1.\*...

cos( A1 + alpham1\*theta ) ./ cos(theta);

if any(x(:) > zeta)

xshift = (x(x>zeta) - zeta) .^ exp1;

if beta == -1 && alpha < 1

p(x > zeta) = 0;

elseif ~quick % Locate peak in integrand and split up integral

g = @(theta) xshift(:) .\* V(theta) - 1;

R = repmat([-theta0, pi/2 ],numel(xshift),1);

if abs(beta) < 1

theta2 = bisectionSolver(g,R,alpha);

else

theta2 = bisectionSolver(g,R,alpha,beta,xshift);

end

theta2 = reshape(theta2,size(xshift));

% change variables so the two integrals go from 0 to 1/2 and 1/2 to 1.

theta2shift1 = 2\*(theta2 + theta0);

theta2shift2 = 2\*(pi/2 - theta2);

g1 = @(theta) xshift .\* ...

V(theta2shift1 \* theta - theta0);

g2 = @(theta) xshift .\* ...

V(theta2shift2 \* (theta - .5) + theta2);

zexpz = @(z) max(0,z .\* exp(-z)); % use max incase of NaN

p(x > zeta) = c2 .\* ...

(theta2shift1 .\* quadv(@(theta) zexpz( g1(theta) ,...

0 , .5, tol) ... + theta2shift2 .\* quadv(@(theta) zexpz( g2(theta) ),...

.5 , 1, tol) );

else % be quick - calculate integral without locating peak. Use a default tolerance of 1e-6

g = @(theta) xshift \* V(theta);

zexpz = @(z) max(0,z .\* exp(-z)); % use max incase of NaN

p( x > zeta ) = c2 .\* quadv(@(theta) zexpz( g(theta) ),...

-theta0 , pi/2, tol );

end

p(x > zeta) = p(x>zeta)/gam; %rescale

end

if any( abs(x(:) - zeta) < 1e-8 )

p( abs(x - zeta) < 1e-8 ) = max(0,gamma(1 + 1/alpha)\*...

cos(theta0)/(pi\*(1 + zeta^2)^(1/(2\*alpha))));

p( abs(x - zeta) < 1e-8 ) = p( abs(x - zeta) < 1e-8 )/gam; %rescale

end

% x < zeta, recall function with -xold, -beta, -delta(No rescale needed)

if any(x(:) < zeta)

p( x < zeta ) = stblpdf( -xold( x<zeta ),alpha,-beta,...

gam , -delta , tol , quick); end

else % Gen case, alpha = 1

x = (x - (2/pi) \* beta \* gam \* log(gam) - delta)/gam; % Standardize

% Compute pdf

piover2 = pi/2;

twooverpi = 2/pi;

oneoverb = 1/beta;

theta0 = piover2;

% Use logs to avoid overflow/underflow

logV = @(theta) log(twooverpi \* ((piover2 + beta \*theta)./cos(theta))) + ...

( oneoverb \* (piover2 + beta \*theta) .\* tan(theta) );

c2 = 1/(2\*abs(beta));

xterm = ( -pi\*x/(2\*beta));

if ~quick % Locate peak in integrand and split up integral

logg = @(theta) xterm(:) + logV(theta) ;

R = repmat([-theta0, pi/2 ],numel(xterm),1);

theta2 = bisectionSolver(logg,R,1-beta);

theta2 = reshape(theta2,size(xterm));

theta2shift1 = 2\*(theta2 + theta0);

theta2shift2 = 2\*(pi/2 - theta2);

logg1 = @(theta) xterm + ...

logV(theta2shift1 \* theta - theta0);

logg2 = @(theta) xterm + ...

logV(theta2shift2 \* (theta - .5) + theta2);

zexpz = @(z) max(0,exp(z) .\* exp(-exp(z))); % use max incase of NaN

p = c2 .\* ...

(theta2shift1 .\* quadv(@(theta) zexpz( logg1(theta) ),...

0 , .5, tol) ...

+ theta2shift2 .\* quadv(@(theta) zexpz( logg2(theta) ),...

.5 , 1, tol) );

else % be quick - calculate integral without locating peak

% Use a default tolerance of 1e-6

logg = @(theta) xterm + logV(theta);

zexpz = @(z) max(0,exp(z) .\* exp(-exp(z))); % use max incase of NaN

p = c2 .\* quadv(@(theta) zexpz( logg(theta) ),-theta0 , pi/2, tol );

end

p = p/gam; %rescale

end

p = real(p); % just in case a small imaginary piece crept in

% This might happen when (x - zeta) is really small

end

* One hurdle where we end up in the above calculation is to solve the equation g(theta)-1=0. This is where we use vectorized bisection method (**Third Step**).
* The solution to this equation is used to increase accuracy in the calculation of a numerical integral.

if nargin < 2

error('bisectionSolver:TooFewInputs','Requires at least two input arguments.');

end

noSolution = false(size(R,1));

tol = 1e-6;

maxiter = 30;

[N M] = size(R);

if M ~= 2

error('bisectionSolver:BadInput',...

'"R" must have 2 columns'); end

a = R(:,1);

b = R(:,2);

X = (a+b)/2;

try

val = f(X);

catch ME

error('bisectionSolver:BadInput',...

'Input function inconsistint with rectangle dimension') end

if size(val,1) ~= N

error('bisectionSolver:BadInput',...

'Output of function must be a column vector with dimension of input');

end

% Main loop

val = inf;

iter = 0;

while( max(abs(val)) > tol && iter < maxiter )

X = (a + b)/2;

val = f(X);

l = (val > 0);

if alpha > 1

l = 1-l; end

a = a.\*l + X.\*(1-l);

b = X.\*l + b.\*(1-l);

iter = iter + 1; end

if any(noSolution(:))

X(noSolution) = (R(1,1) + R(1,2))/2; end

**Graphs and Plots:**

xlength =-80:1:80;

figure(1)

hist(r);

title('The histogram to plot the number of samples');

xlabel('The range of X ');

ylabel('The frequency of the samples lying in the range of X');

figure(2)

plot( xlength , stblpdf(xlength,alpha,beta,gamma,delta,'quick'));

title('The Theo PDF of the alpha stable random variable');

xlabel('The range of X ');

ylabel('The probability(PDF) for different values of x');

figure(3)

title('The graph shows the Theo pmf overlayed on the histogram of random variable generated by M ');

yyaxis left

y=hist(r,xlength);

bar(xlength,y);

ylabel('The frequency of the number of samples generated');

yyaxis right

plot(xlength,stblpdf(xlength,alpha,beta,gamma,delta,'quick'),'g');

ylabel('The PDF of the Alpha Stable Distribution');

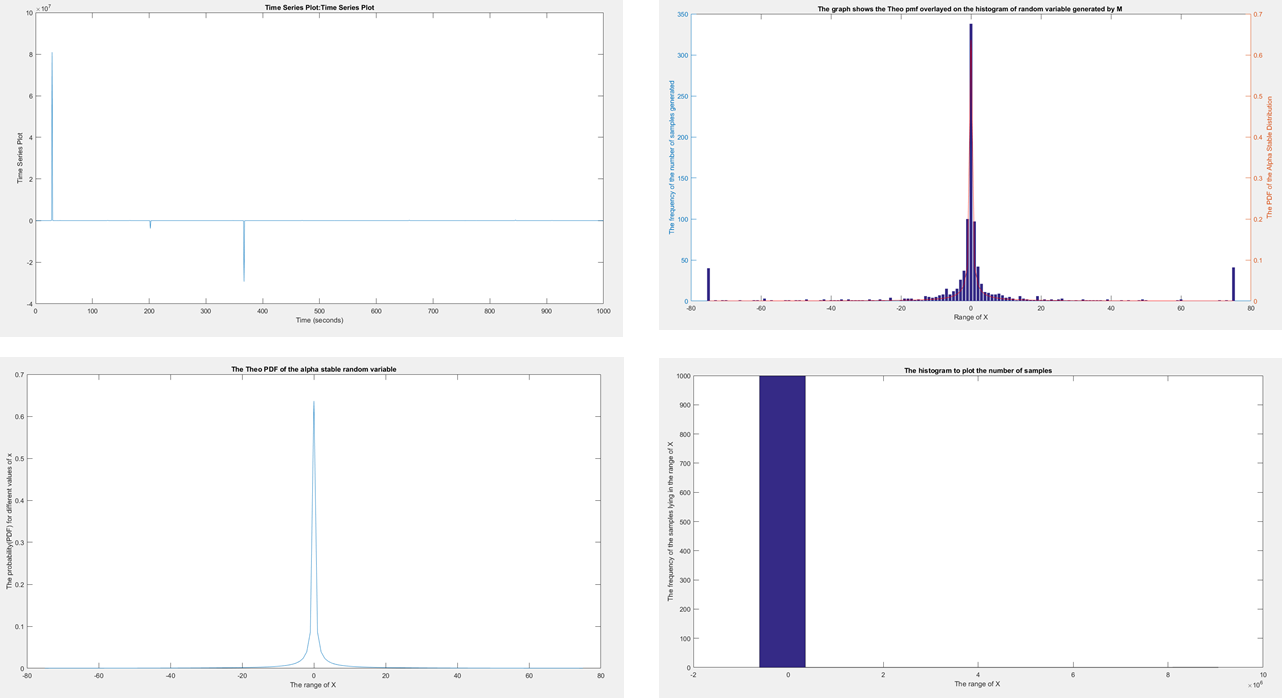
figure(4)

time\_plot = timeseries(r,1:1000);

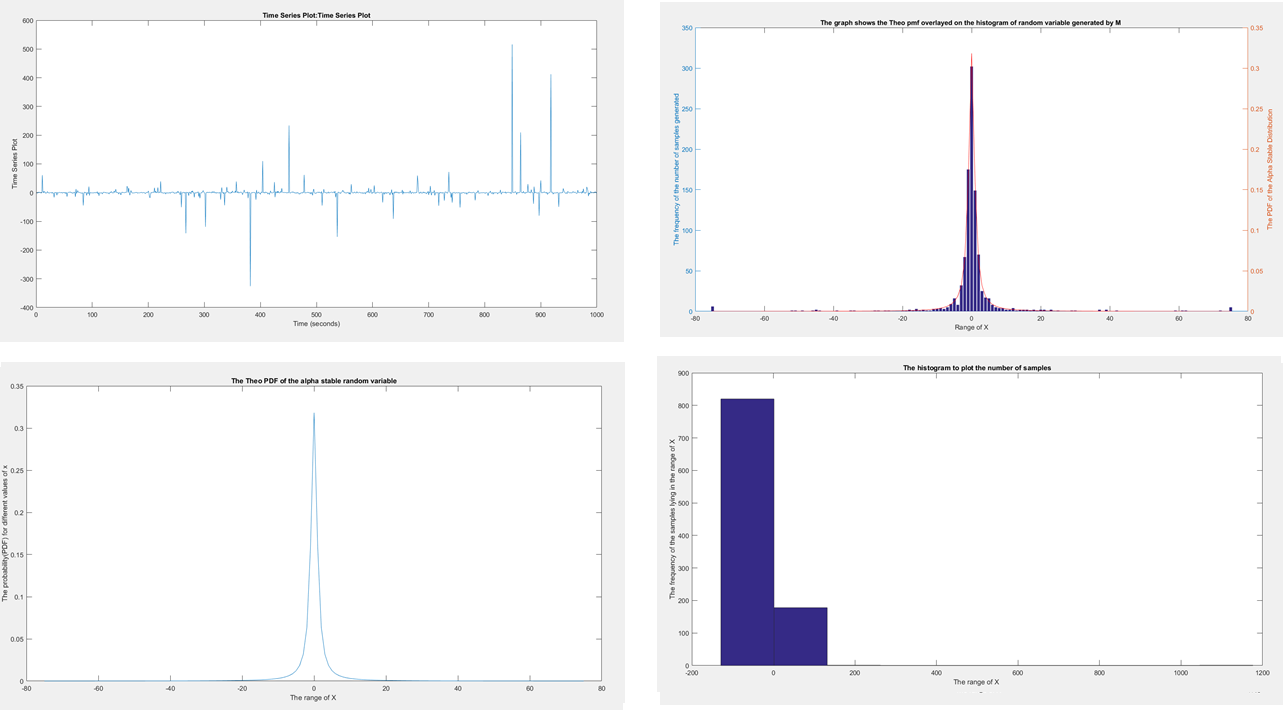
time\_plot.Name = 'Time Series Plot';

plot(time\_plot)

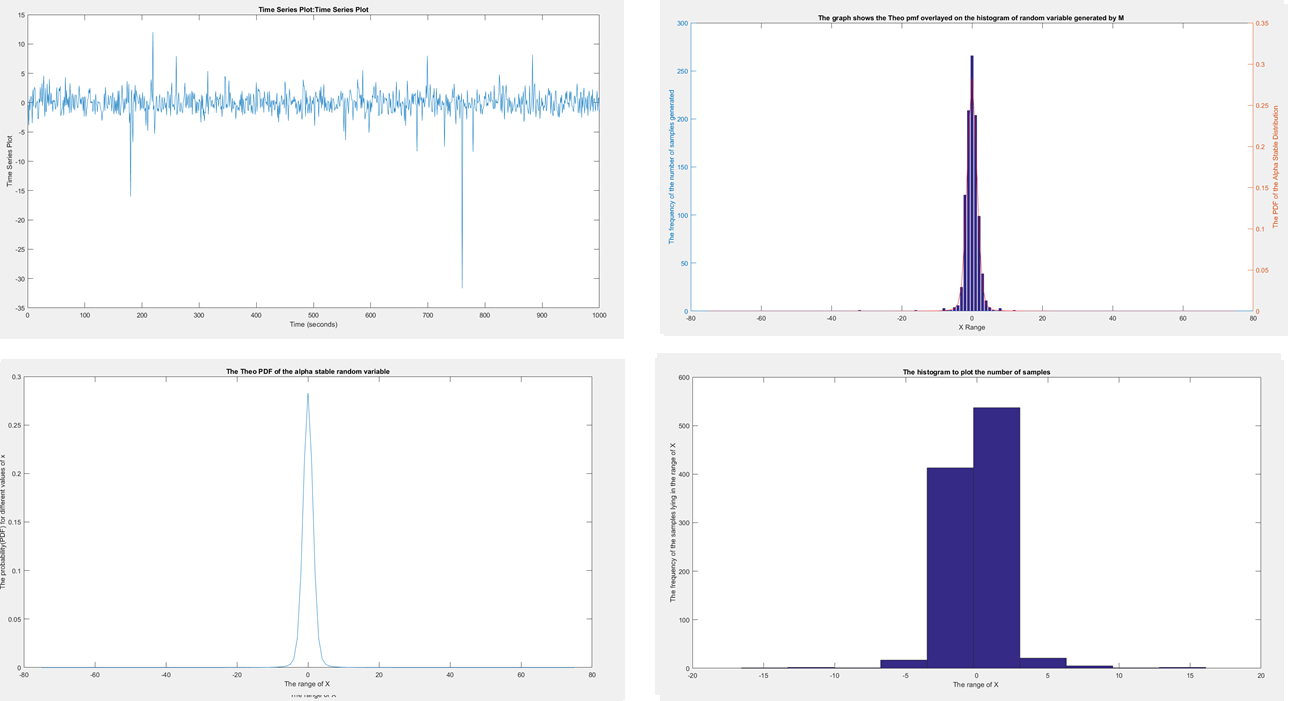
**ALPHA =0.5, BETA =0, GAMMA=1, DELTA=0, SAMPLES=1000**

****

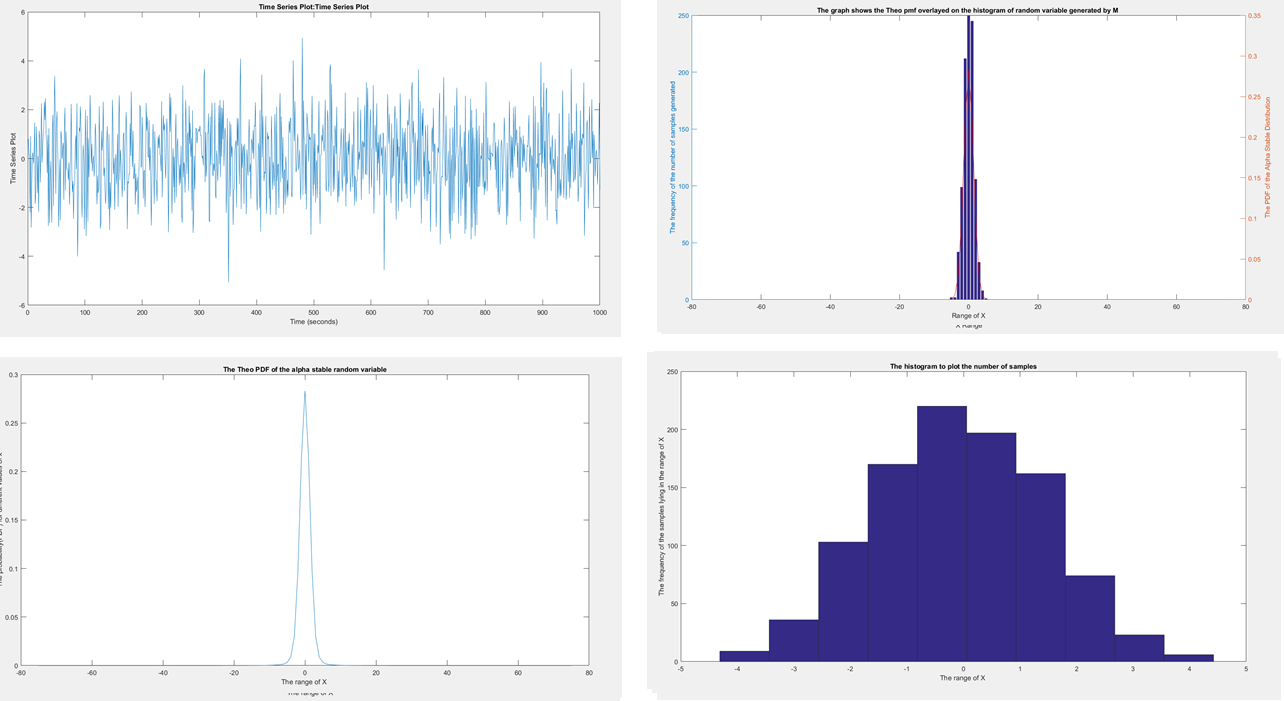
**ALPHA =1 BETA =0, GAMMA=1, DELTA=0, SAMPLES=1000**

****

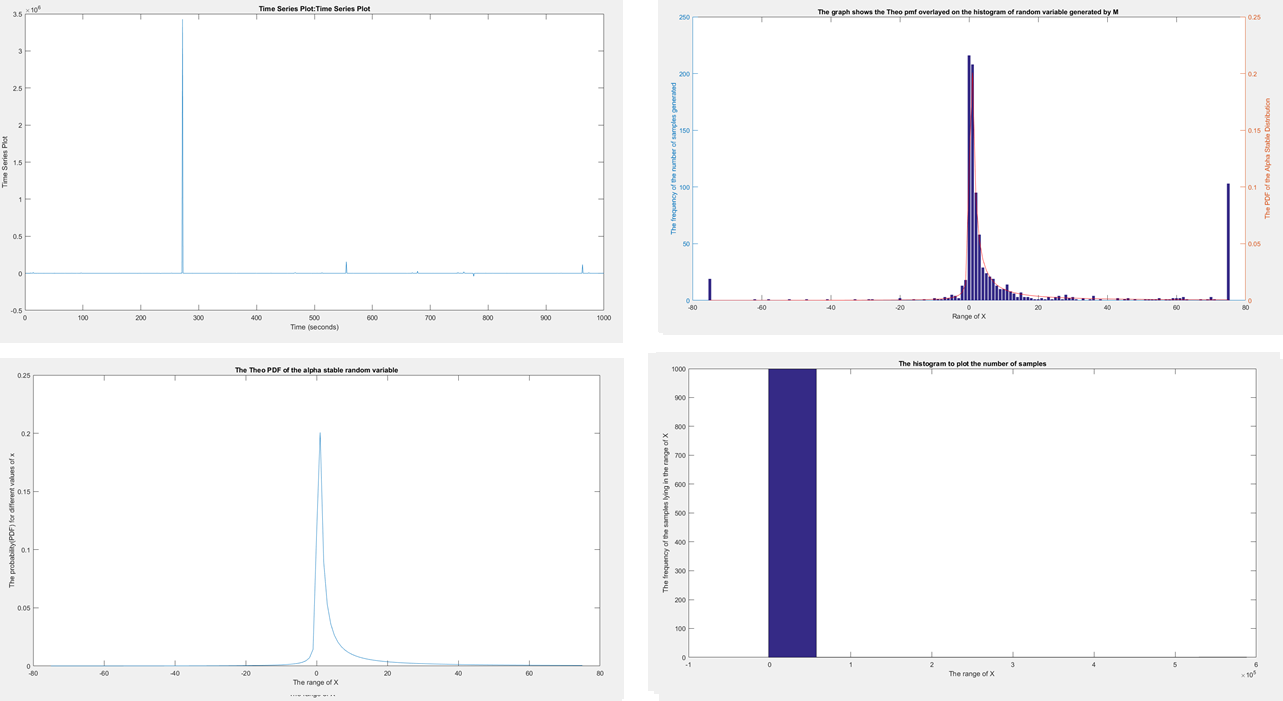
**ALPHA =1.8 BETA =0, GAMMA=1, DELTA=0, SAMPLES=1000**

****

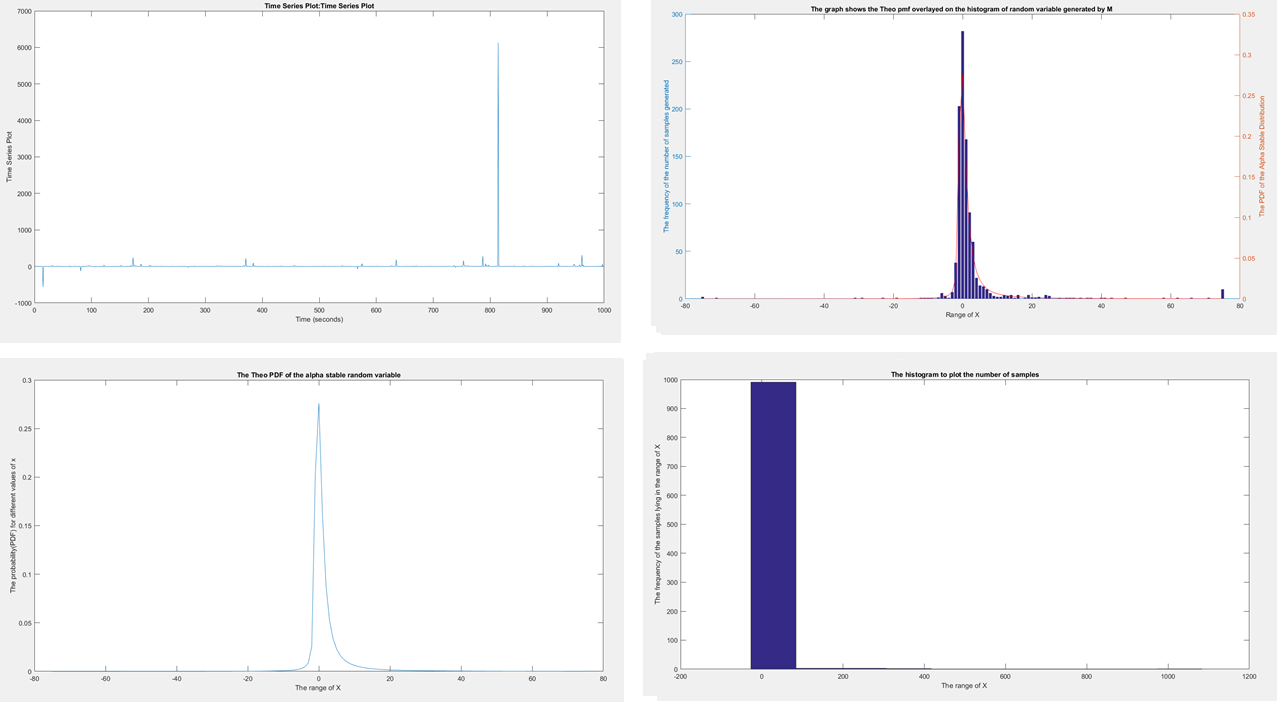
**ALPHA =2 BETA =0, GAMMA=1, DELTA=0, SAMPLES=1000**



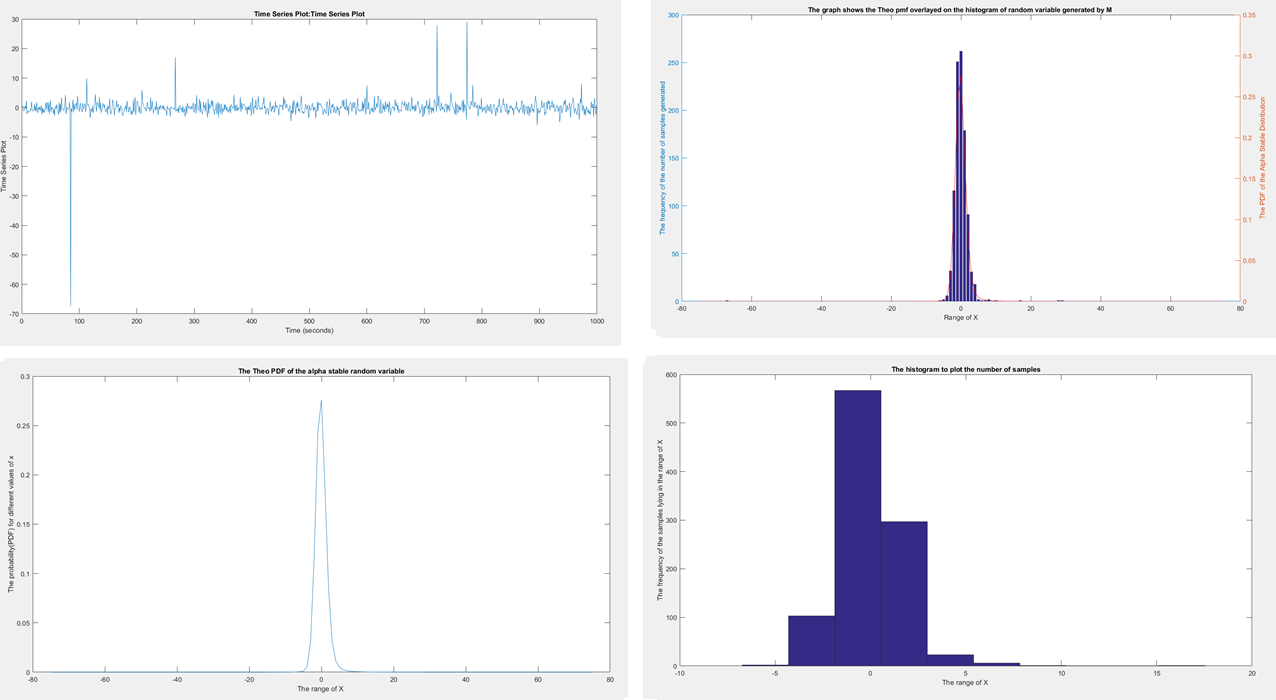
**ALPHA =0.5 BETA =0.75, GAMMA=1, DELTA=0, SAMPLES=1000**

****

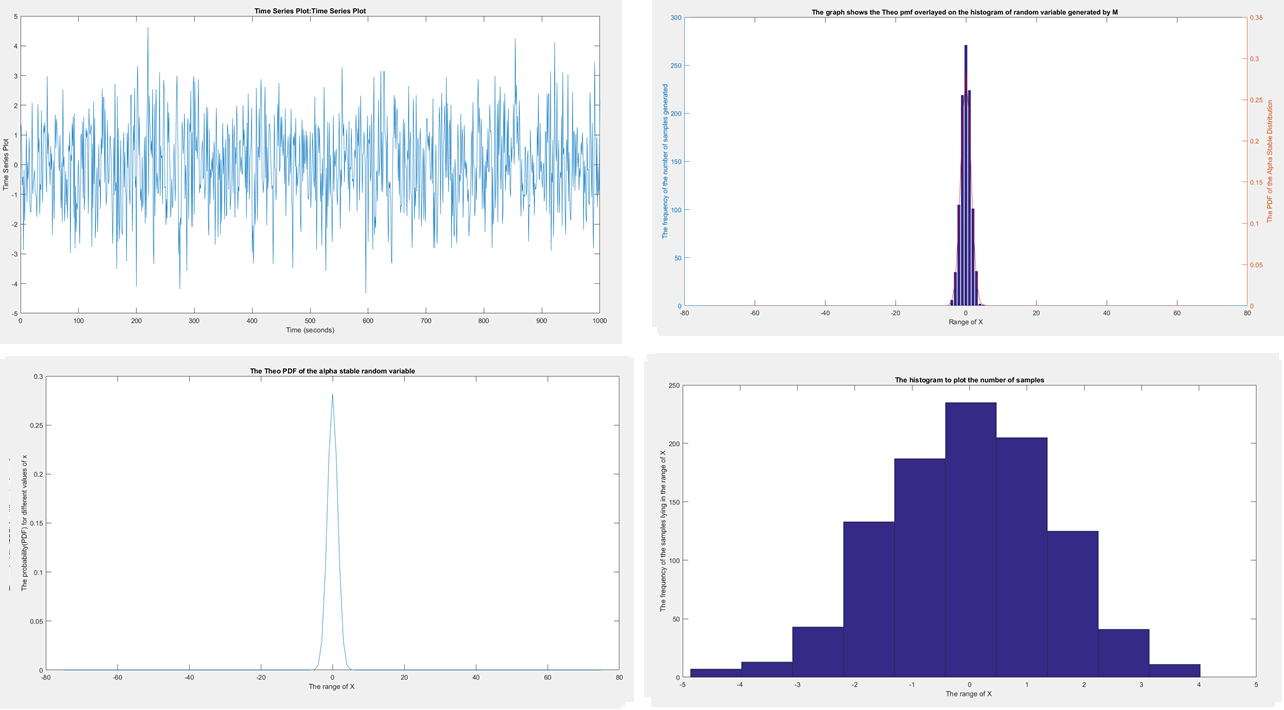
**ALPHA =1 BETA =0.75, GAMMA=1, DELTA=0, SAMPLES=1000**

****

**ALPHA =1.8 BETA =0.75, GAMMA=1, DELTA=0, SAMPLES=1000**



**ALPHA =2 BETA =0.75, GAMMA=1, DELTA=0, SAMPLES=1000**

****

**Observations:**

* We observe that the samples generated by Mallow Stuck method for alpha distributions (0.5, 1, 1.8 and 2) fit well within the theoretical PDF.
* As alpha increases from 0.5 to 2 (for a given Beta), we see that the tail of the distribution increases.
* The time series which denotes the noise (Gaussian) is more significant as we move from alpha 0.5 to 2.
* The histogram to plot the number of samples are distributed differently as alpha changes from 0.5 to 1.
* The PDF of alpha stable random variable is constant even if alpha changes for a constant beta value.
* Symmetry of the distribution is affected with the value of beta. (a positive beta shifts the symmetry of graph to positive X axis).