University of Southern California

EE511 Simulation Methods for Stochastic Systems

Project #6- Continuous sampling

BY

Mohan Krishna Thota

USC ID: 6683486728

mthota@usc.edu

PROBLEM 1:

BOX-MULLER TRANSFORM

- 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 is expressed in two different forms
- One default form from Box and Muller, maps two samples generated from the uniform distribution on the interval [0,1] to two standard normally distributed samples.
- Whereas the other, Polar form maps two samples taken from distinct interval [-1, +1] to two normally distributed samples without applying sine or cosine functions.

Let U1 and U2 be random variables that are independent and are uniformly distributed in interval (0,1).

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

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

Here in the above, Zo and Z1 be random variables which are independent and follow a standard normal distribution.

POLAR MARSAGLIA METHOD

- A pseudo-random number sampling method for generating a pair of independent standard normal random variables.
- It is superior to the Box–Muller transform.
- This method works by choosing random points (x, y) in the square −1 < x < 1, −1 < y < 1 until

$$s = x^2 + y^2 < 1.$$

and then returning the required pair of normal random variables as.

$$x\sqrt{\frac{-2\ln(s)}{s}}, \quad y\sqrt{\frac{-2\ln(s)}{s}}.$$

Or equivalently,

$$\frac{x}{\sqrt{s}}\sqrt{-2\ln(s)}, \quad \frac{y}{\sqrt{s}}\sqrt{-2\ln(s)},$$

where $x/\sqrt{s}$ and $y/\sqrt{s}$ represent the cosine and sine of the angle that the vector (x, y) makes with x axis.

ALGORITHM:

- Here U1, U2 are sampled from uniform distribution (-1,1)
- Here check for the condition, if s=(U1^2+U1^2)
- If the condition is not satisfied, we will be taking new U1, U2.
- Then value of X and Y is computed.
- X = sqrt(−2 log(s)/s)*U1
- Y = sqrt(−2 log(s)/s)*U2

CODE:

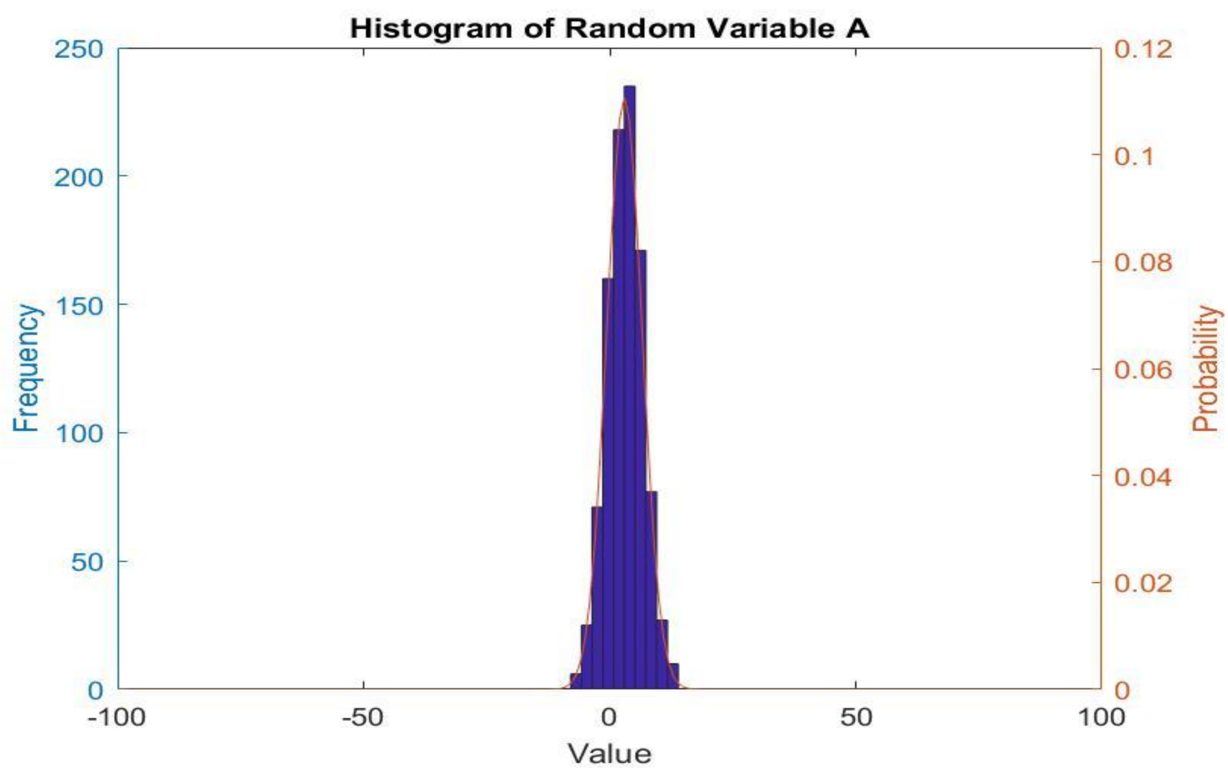
- Independent random variables are generated using Box-Muller method. BoxMuller Random Samples function is used to perform the required operation according to the formulas described above.
- Probability distribution is made using makedist function.
- By utilizing Matlab's inbuilt 'cov' function, covariance is calculated.
- Generation of Histogram is done, and the theoretical pdf is overlayed on the histogram as can be seen in the graph.
- Here Matlab's 'tic' and 'toc' commands computational time taken by above method like Box Muller is calculated for a million samples.
- Similarly, we calculate the computational time for Polar Marsaglia method and compare the results. We use PolarMarsaglia_RandomSamples function for the calculation.

CODE:

```
clc;
close all;
clear all;

%Box Muller
[M,N,A]=Box_Muller_method(1000);
covariance1=cov(M,N);
T = -100:100;
mu=3;
sigma=sqrt(13);
prob_dist=makedist('Normal',mu,sigma);%Make probability distribution
P=pdf(prob_dist,T);
%start timer
tic;
Box_Muller_method(1000000);
%stop timer
ET1=toc;

%Output
disp('Output for the Box-Muller Method:');
fprintf('\nThe Covariance of X and Y is: \n%f %f\n%f %f\n',covariance1);
yyaxis left
hist(A);
title('Histogram of Random Variable A');
xlabel('Value');
ylabel('Frequency');
yyaxis right
plot(T,P);
ylabel('Probability');
mx=mean(M);
my=mean(N);
vx=var(M);
vy=var(N);
fprintf('\nThe Observed mean and variance of X are:\nMean = %f\nVariance =
%f\n',mx,vx);
fprintf('\nThe Observed mean and variance of Y are:\nMean = %f\nVariance =
%f\n',my,vy);
%start time
```

```
%start time
tic;
[M,N]=Polarmarsaglia(1000000);
%end time
ET2=toc;

meanx1=mean(M);
meany1=mean(N);
varx1=var(M);
vary1=var(N);
cov2=cov(M,N);
disp('Polar Marsaglia method output:');
fprintf('\nObserved mean and variance of M are:\nMean = %f\nVariance =
%f\n',meanx1,varx1);
fprintf('\nObserved mean and variance of N are:\nMean = %f\nVariance =
%f\n',meany1,vary1);
fprintf('\nThe Covariance of M and N is: \n%f %f\n%f %f\n',cov2);
disp('Time required to generate 1,000,000 pairs of independent samples\n');
fprintf('Utilizing the Polar Marsaglia method :\n%f\n',ET2);
fprintf('Utilizing the Box Muller method:\n%f\n',ET1);


function [M,N,O] = Box_Muller_method(number)
%m-mean
%v-variance
mean2=2;
var2=9;
mean1=1;
var1=4;
   for j=1:number
      %uniform random number generation
      rand2=rand();
      rand1=rand();
      x_var(j)=sqrt(-2*log(rand1))*cos(2*pi*rand2);
      y_var(j)=sqrt(-2*log(rand1))*sin(2*pi*rand2);
      M(j)=sqrt(var1)*x_var(j)+mean1;
      N(j)=sqrt(var2)*y_var(j)+mean2;
      O(j)=M(j)+N(j);
   end
end
```

```
function[M,N] = Polarmarsaglia(n)
%m-mean
%v-variance
mean3 = 1;
var3 = 4;
mean4 = 2;
var4 = 9;
%algorithm generated random number
j = 0;
%Independent random variable generation
while(j<=n-1)
   rand1 = 2*rand()-1;
   rand2 = 2*rand()-1;
   sum = rand1^2 + rand2^2;
   if(sum < 1)
      j = j + 1;
      x1_var(j) = sqrt(-2*log(sum)/sum)*rand1;
      y1_var(j) = sqrt(-2*log(sum)/sum)*rand2;
   end
end
% Scaling
M = sqrt(var3)*x1_var + mean3;
N = sqrt(var4)*y1_var + mean4;
end
```



Output for the Box-Muller Method:

The Covariance of X and Y is:

3.999384 -0.034594

-0.034594 9.258359

The Observed mean and variance of X are:

Mean = 1.073860

Variance = 3.999384

The Observed mean and variance of Y are:

Mean = 1.868625

Variance = 9.258359

Polar Marsaglia method output:

Observed mean and variance of M are:

Mean = 0.996184

Variance = 4.004706

Observed mean and variance of N are:

Mean = 1.998835

Variance = 8.988067

The Covariance of M and N is:

4.004706 0.002958

0.002958 8.988067

Time required to generate 1,000,000 pairs of independent samples\n

Utilizing the Polar Marsaglia method:

0.492559

Utilizing the Box Muller method:

1.012355

Time required to generate 1,000,000 pairs of independent samples\n

Utilizing the Polar Marsaglia method :

0.461763

Utilizing the Box Muller method:

0.818044

ANALYSIS:

- Polar Marsaglia method takes lesser computational time when compared to Box Muller method. This is evident from the mentioned two trials.
- From the above results it is evident that Polar Marsaglia is superior when compared to Box Muller method.

PROBLEM 2:

GAMMA RANDOM VARIABLE GENERATION

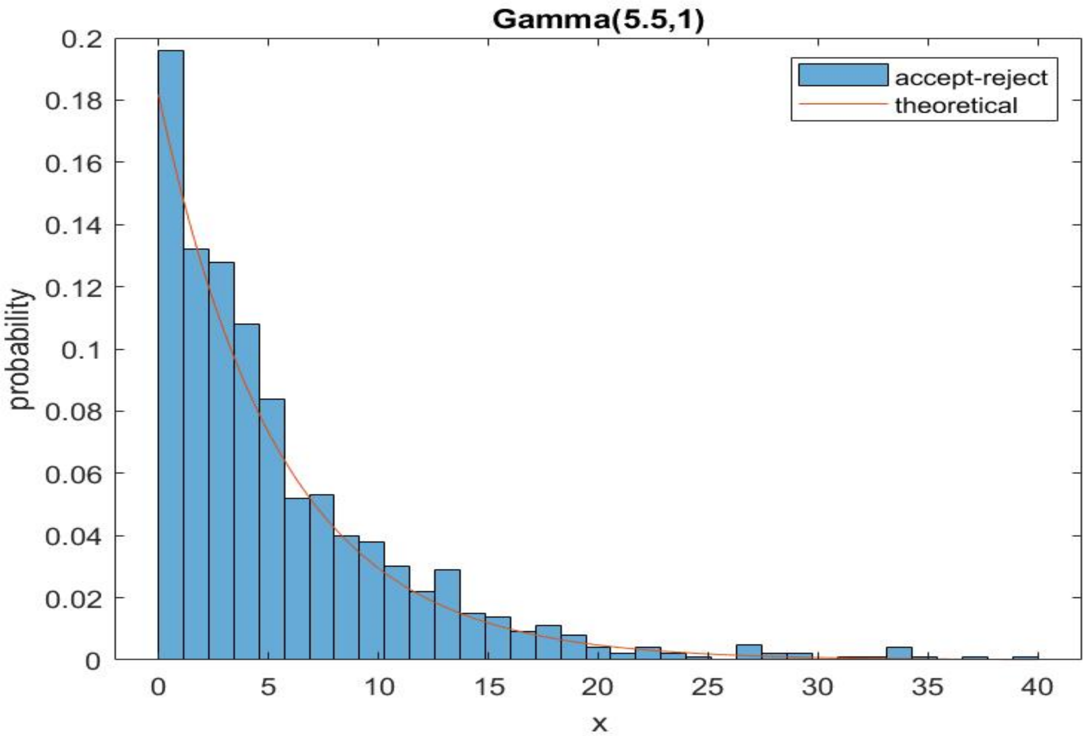
Accept-Reject Method

- Sampling values are generated using rejection sampling from a target 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)/(Mg(x)), repeating the draws from Y until a value is accepted.
- First, we derive the constant M, M need to be make sure that Mg(x) > f(x) should be true for every x, therefore, we take the max of (f(x)/g(x)) to determine the M. Then we using the accept-reject method to derive f(x)

ALGORITHM

- Set random variable S which has distribution g(x).
- Compute the constant M.
- Generate a sample s from S, generate a number u from std U [0,1].
- if u< p(s)/Mg(s), accept and records, else reject.
- Repeat 3,4 for trail budget times.

The exponential distribution with parameter 0.1 is used as my g(x), because it is easy to use inverse sampling method to generate a sample from exponential distribution.



Generation of Gamma Accept-Reject Method

```
clc;
clear all;
close all;
trial_count = 0;
X = 2;
A = 0;


while A < 1000
%Sample generation
   y = generateExpDis(0.1);
   if rand(1)<exponential5(y)/(X*0.1*exp(-0.1 * y))
      A = A + 1;
      sample(A) = y;
   end
   trial_count = trial_count + 1;
end

efficiency = 1000/trial_count;
histogram(sample',35,'BinLimits',[0,40],'Normalization','probability');
hold on
t = 0: 0.1: 40;

title('Gamma(5.5,1)');
plot(t,exp5_5(t));
legend('accept-reject','theoretical');
ylabel('probability');
xlabel('x');
```

ANALYSIS:

- From the above results we could see that a total of 2056 trials are taken to generate 1000 samples
- the acceptance rate is nearly every M sampling we get one sample.

PROBLEM 3:

- We call X as random variable when its characteristic function can be written as

$$\phi(\omega; \alpha, \beta, c, \mu) = exp\big(i\omega\mu - |c\omega|\alpha(1 - i\beta sgn(\omega)\Phi)\big)$$

$$\phi = \{tan(\pi\alpha/2), \alpha\ != 1, (-2/\pi)\log|\omega|, \alpha = 1\}$$

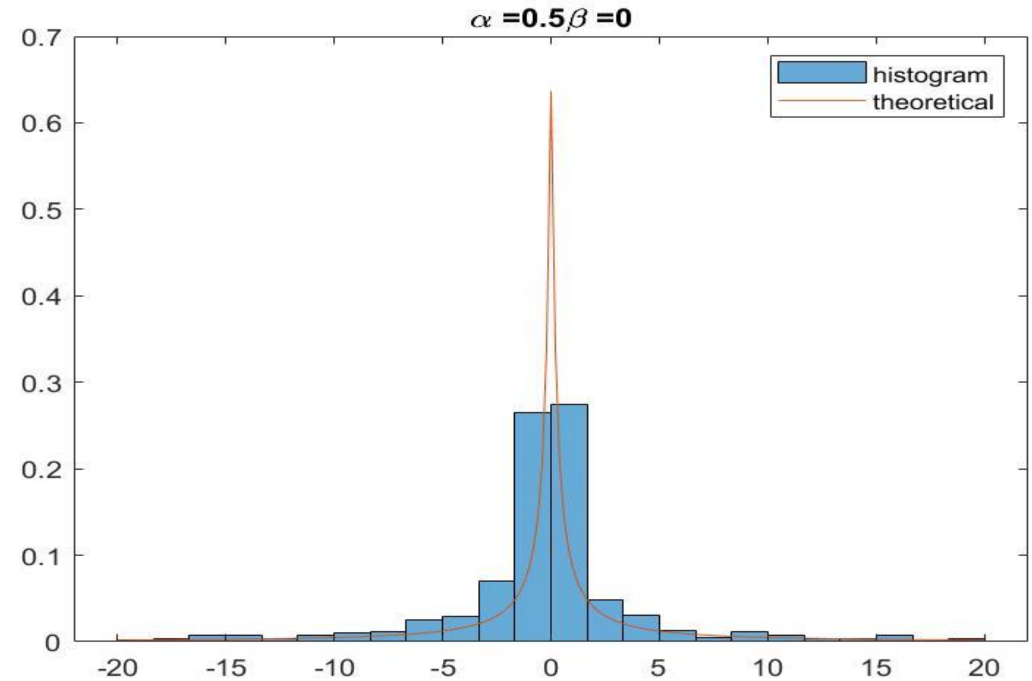
$\mu \in$ R is a shift parameter, β belongs to range [-1,1] is a skewness parameter which measures asymmetry
- In this context the usual skewness is not well defined, as for α < 2the distribution does not admit 2nd or higher moments, and the usual skewness definition is the 3rd central moment.

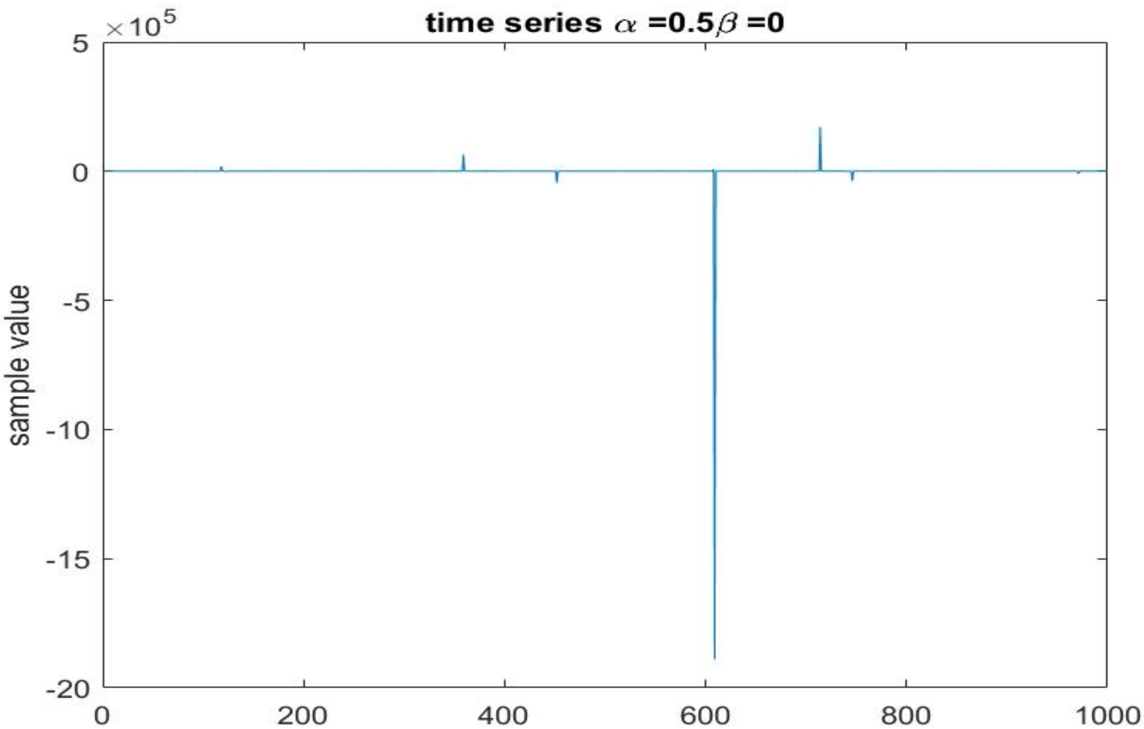
The following **program** is run for a single experiment.

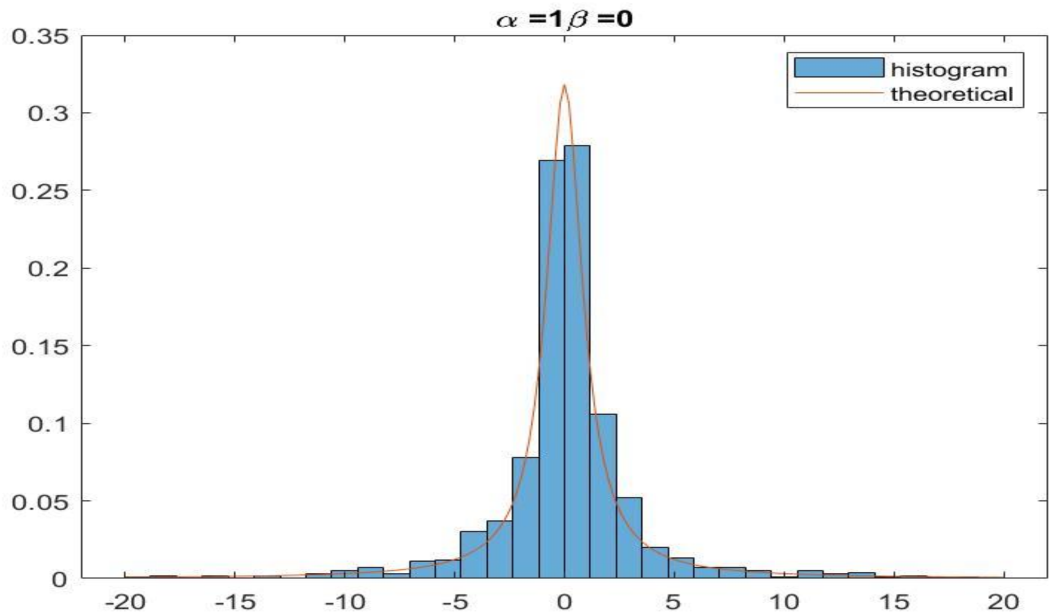
```
function experiment(alpha,beta,sampleAmount)
figure

X = stblrnd(alpha,beta,1,0,sampleAmount,1);
histogram(X,'BinLimits',[-20,20],'Normalization','probability');
t = -20:0.2:20;
hold on
y = stblpdf(t,alpha,beta,1,0);
plot(t,y)
title(strcat('\alpha =', num2str(alpha) , '\beta =', num2str(beta)));
legend('histogram','theoretical')
hold off
%time series
figure
plot(X)
ylabel('sample value')
title(strcat('time series \alpha =', num2str(alpha) , '\beta =', num2str(beta)))
end
```

Program:

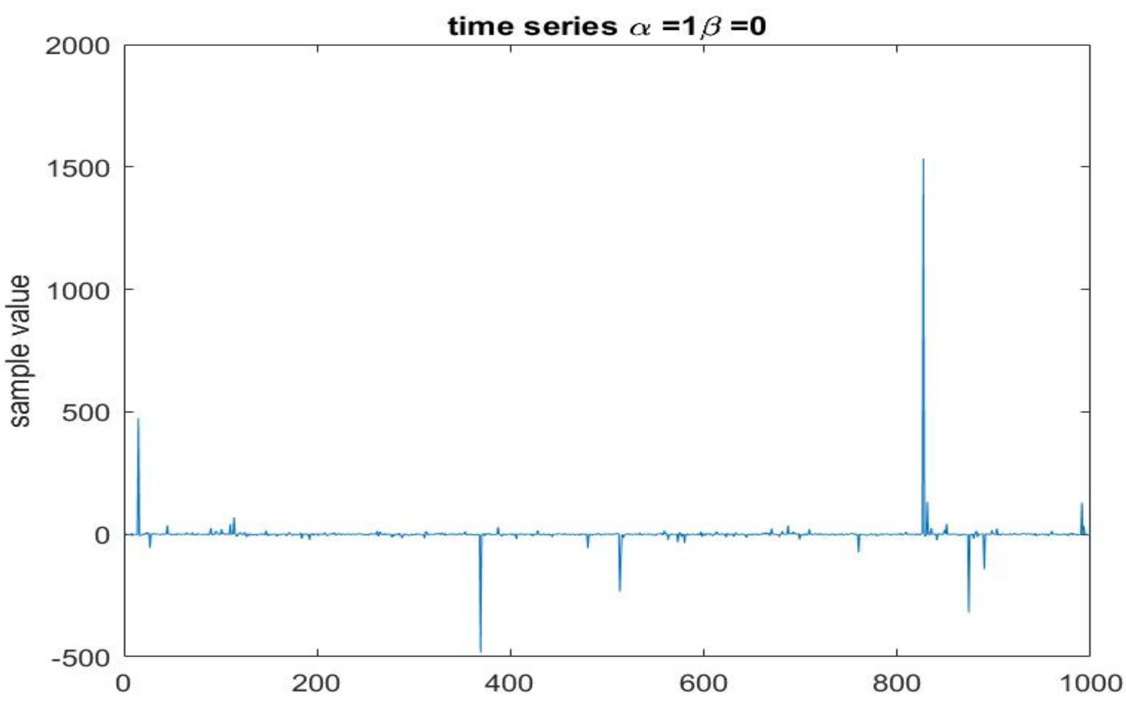
```
experiment(0.5,0,1000);
experiment(1,0,1000);
experiment(1.8,0,1000);
experiment(2.0,0,1000);
experiment(0.5,0.75,1000);
experiment(1.0,0.75,1000);
experiment(1.8,0.75,1000);
experiment(2.0,0.75,1000);
```


α =0.5β =0
histogram
theoretical
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
-20
-15
-10
-5
0
5
10
15
20

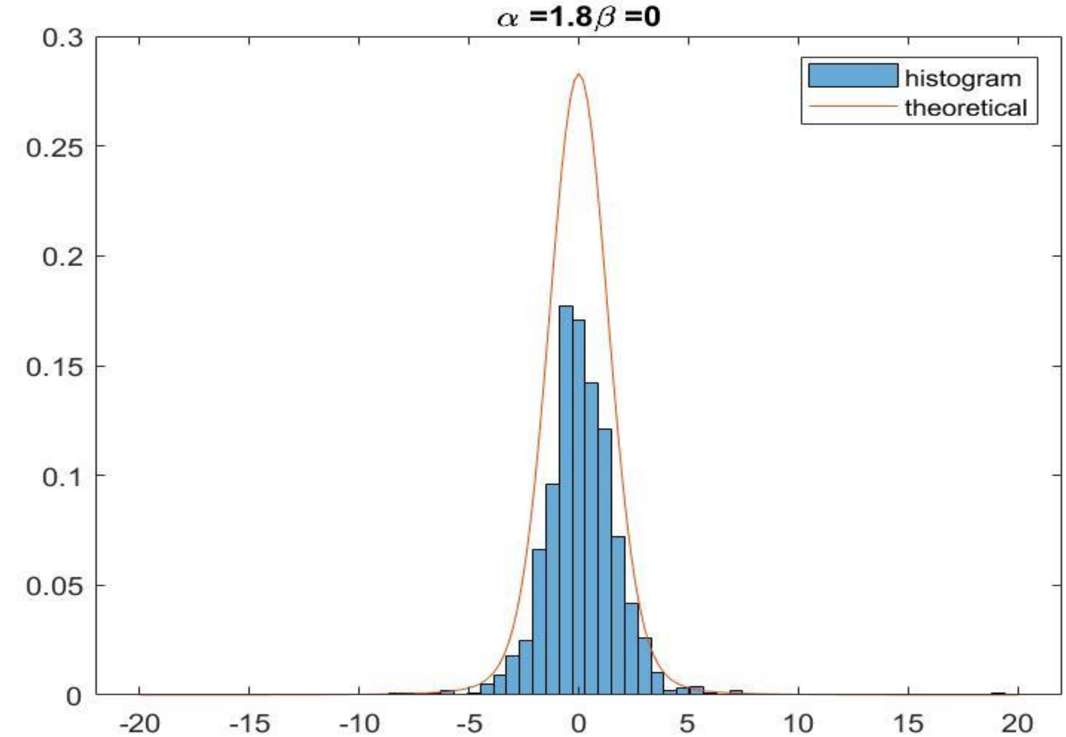

time series α =0.5β =0
$\times10^5$
sample value
5
0
-5
-10
-15
-20
0
200
400
600
800
1000

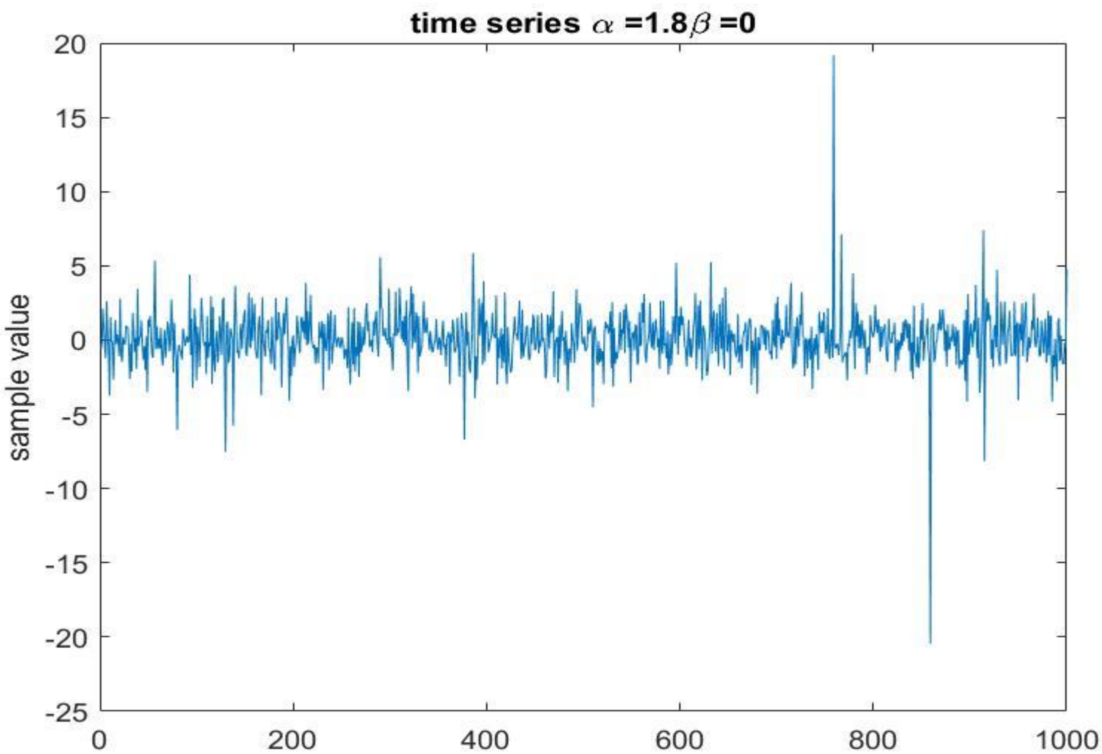


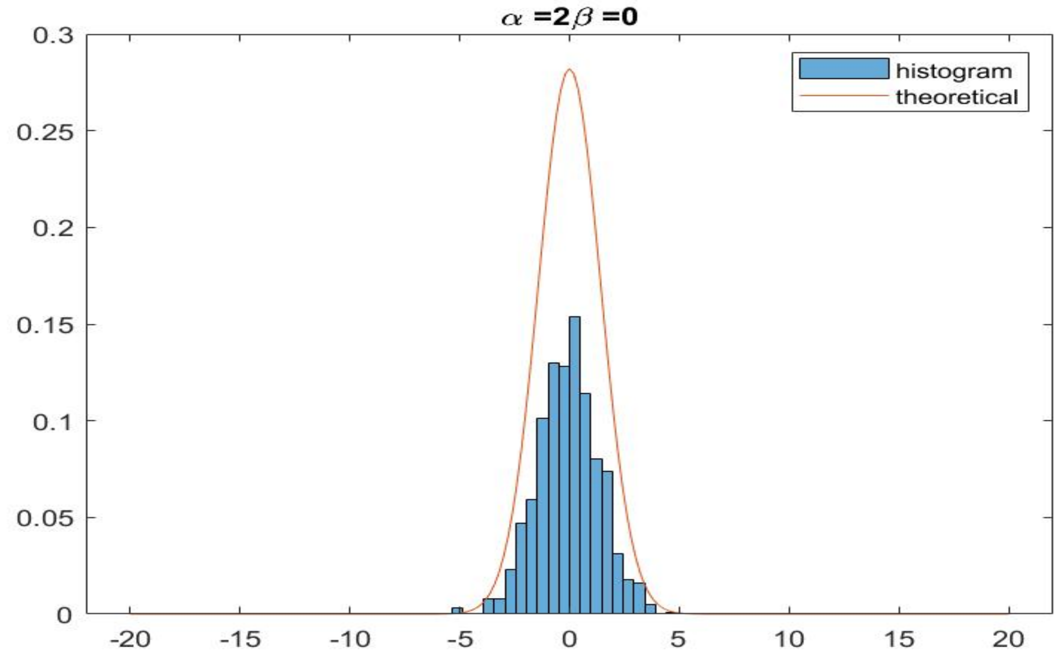
α =1β =0
histogram
theoretical
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
-20
-15
-10
-5
0
5
10
15
20

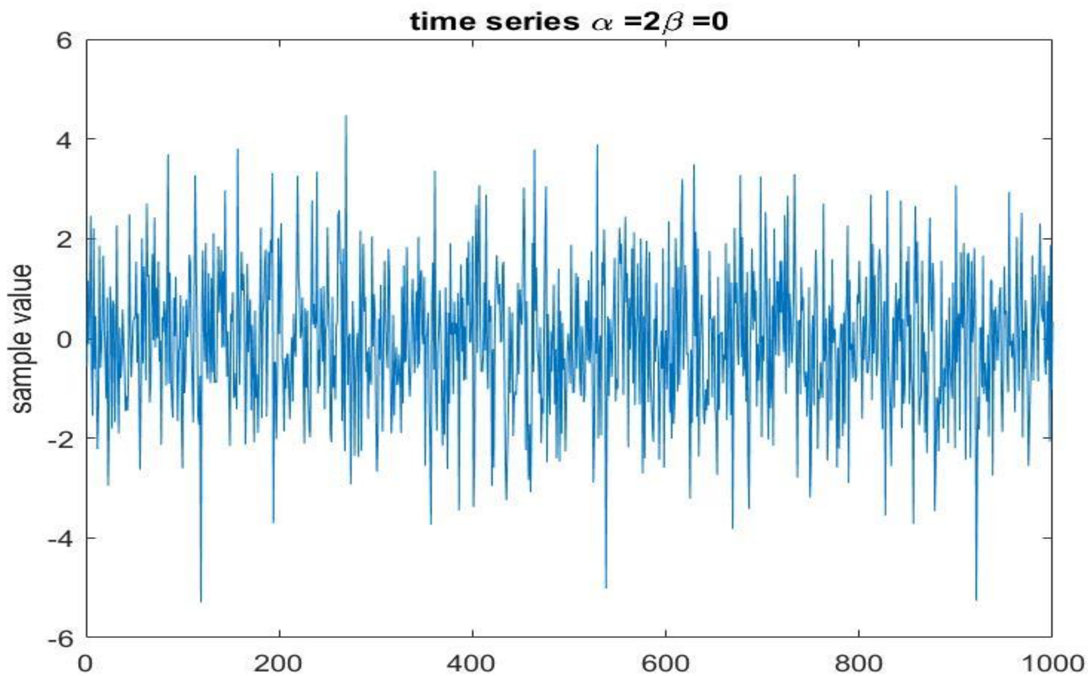


time series α =1β =0
sample value
2000
1500
1000
500
0
-500
0
200
400
600
800
1000


α =1.8β =0
histogram
theoretical
0.3
0.25
0.2
0.15
0.1
0.05
0
-20
-15
-10
-5
0
5
10
15
20

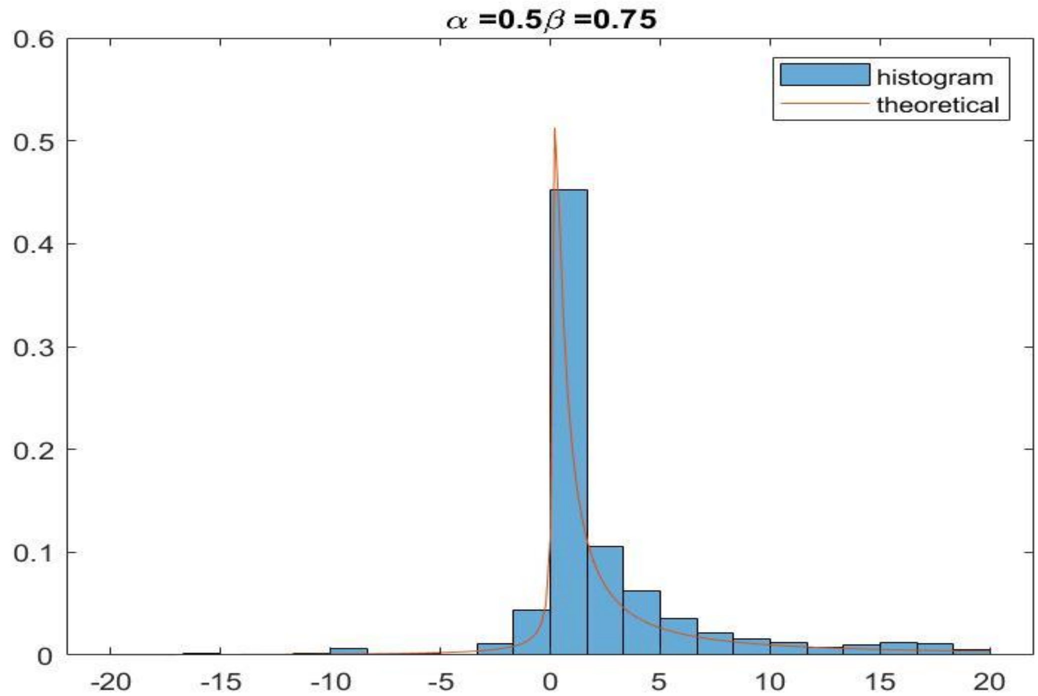

time series α =1.8β =0
sample value
20
15
10
5
0
-5
-10
-15
-20
-25
0
200
400
600
800
1000



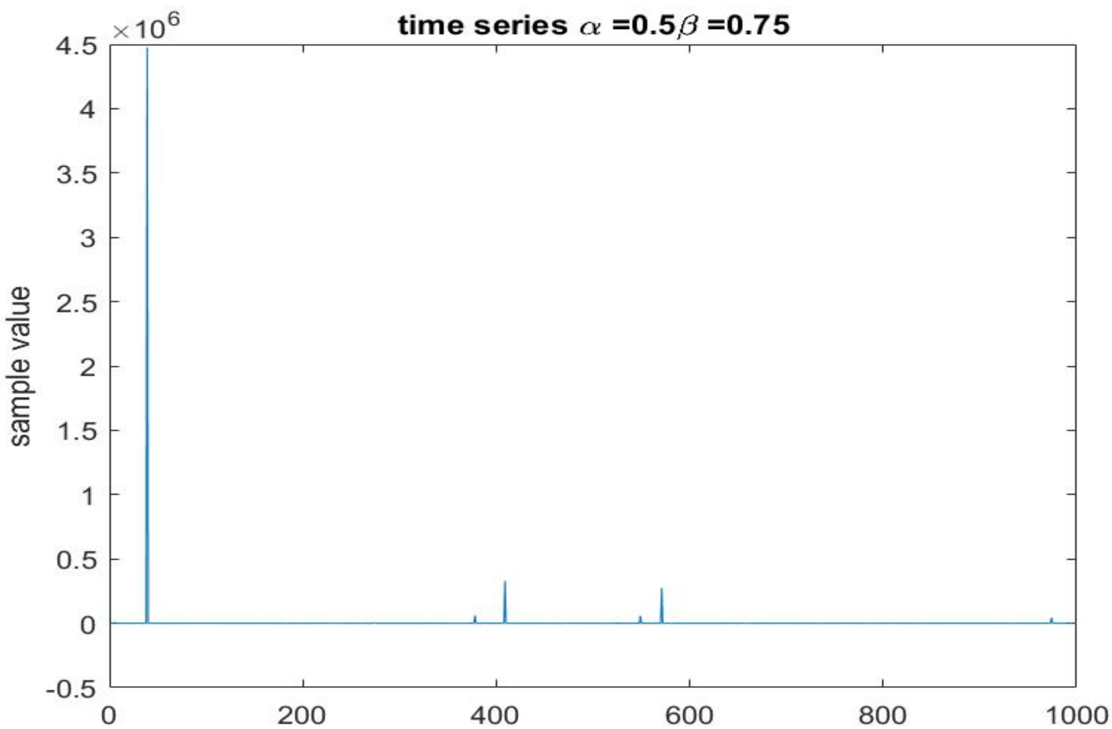


From the graphs we could see that, with the decrease in value of alpha, a less oscillating and the magnitude of the sample is large,

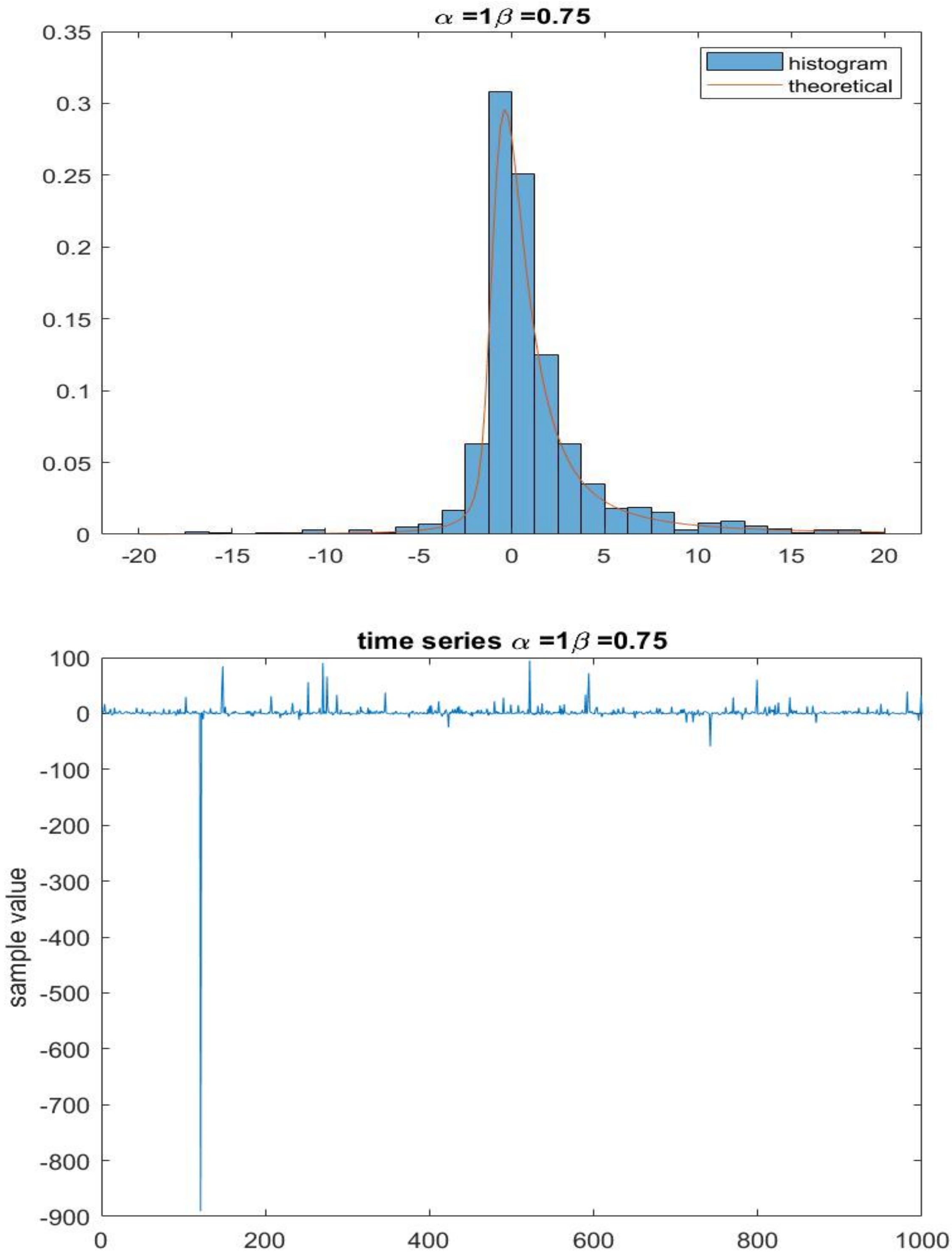
When the value of alpha is 2, a gaussian distribution and the white noise is simulated by samples.



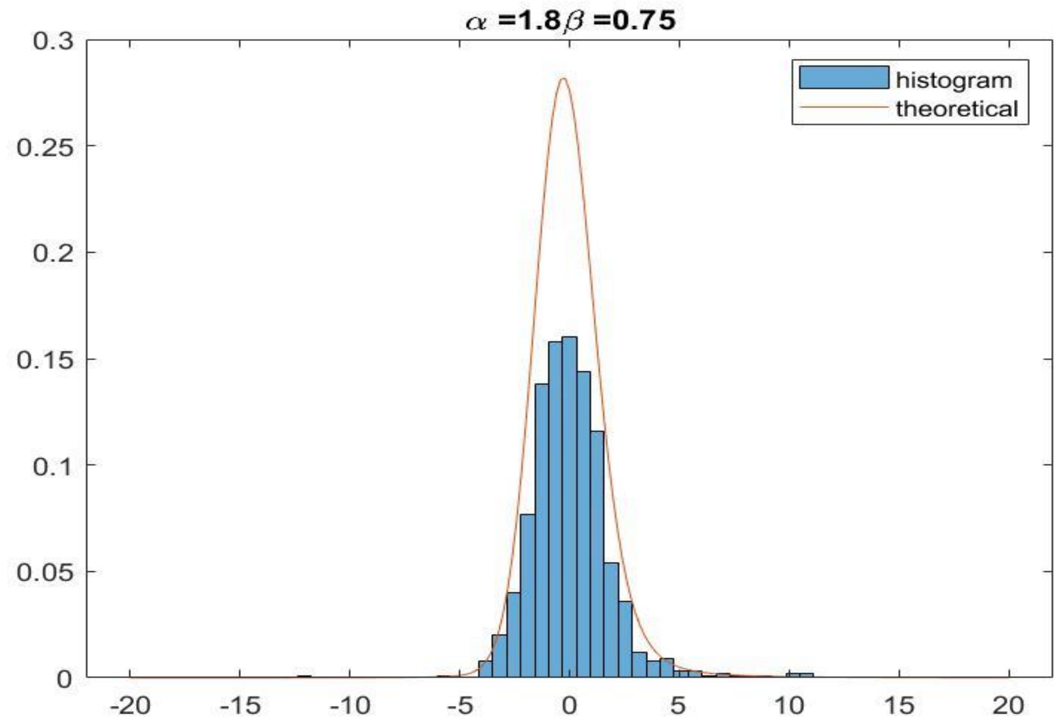
α =0.5β =0.75
histogram
theoretical
0.6
0.5
0.4
0.3
0.2
0.1
0
-20
-15
-10
-5
0
5
10
15
20

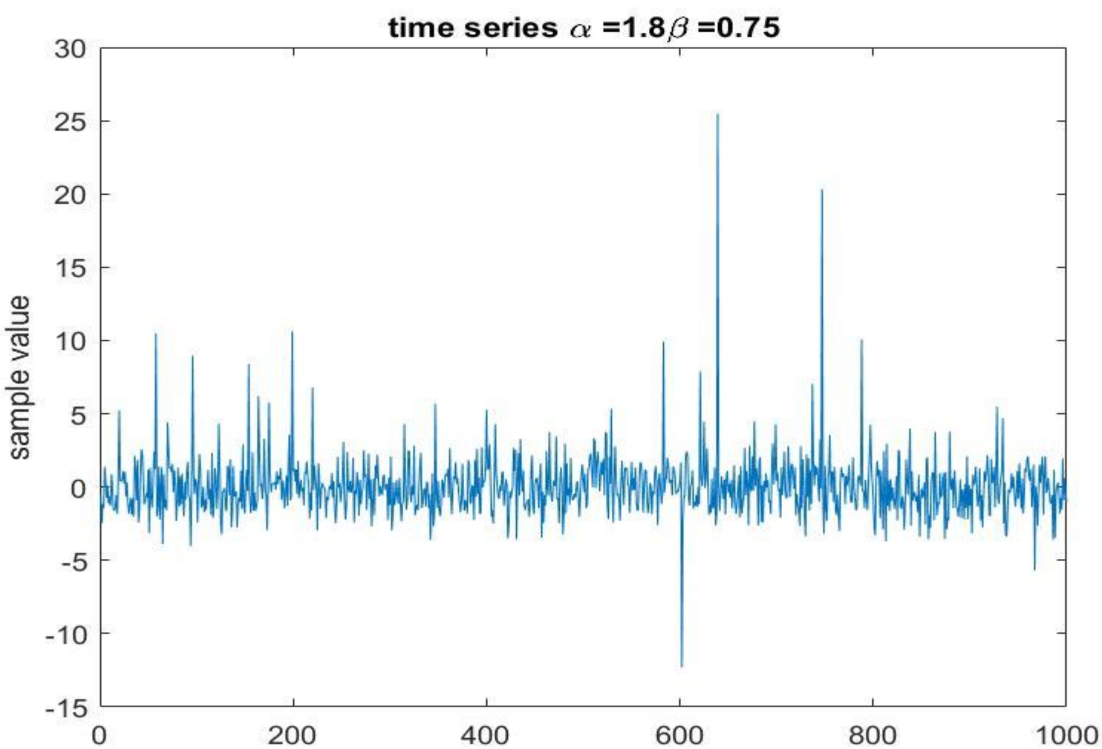


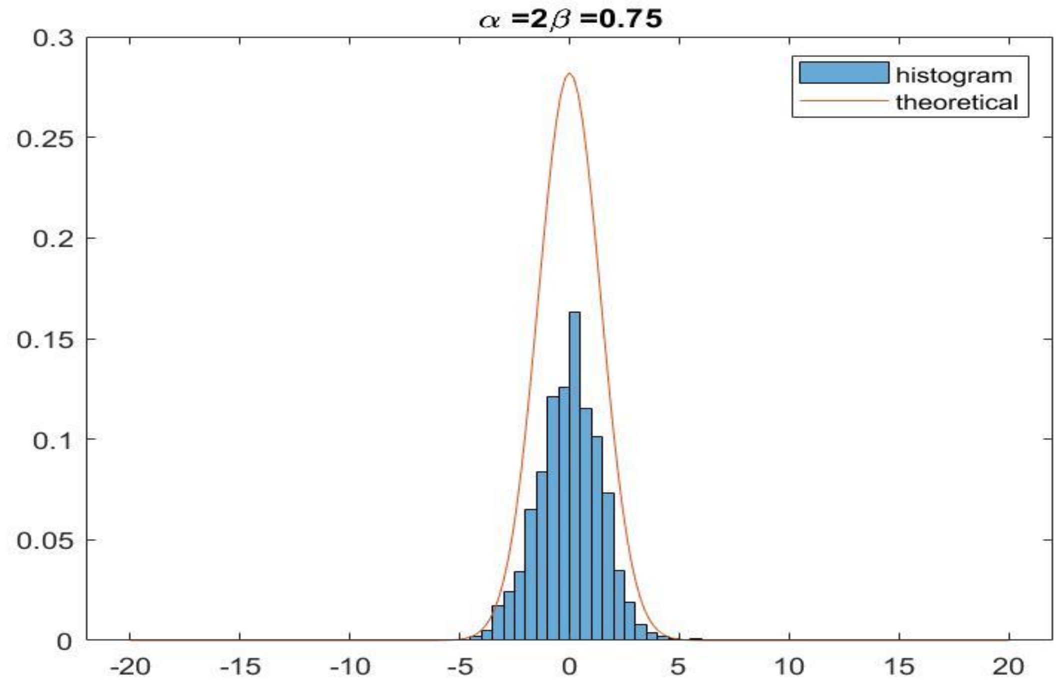
time series α =0.5β =0.75
$\times 10^6$
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
-0.5
sample value
0
200
400
600
800
1000

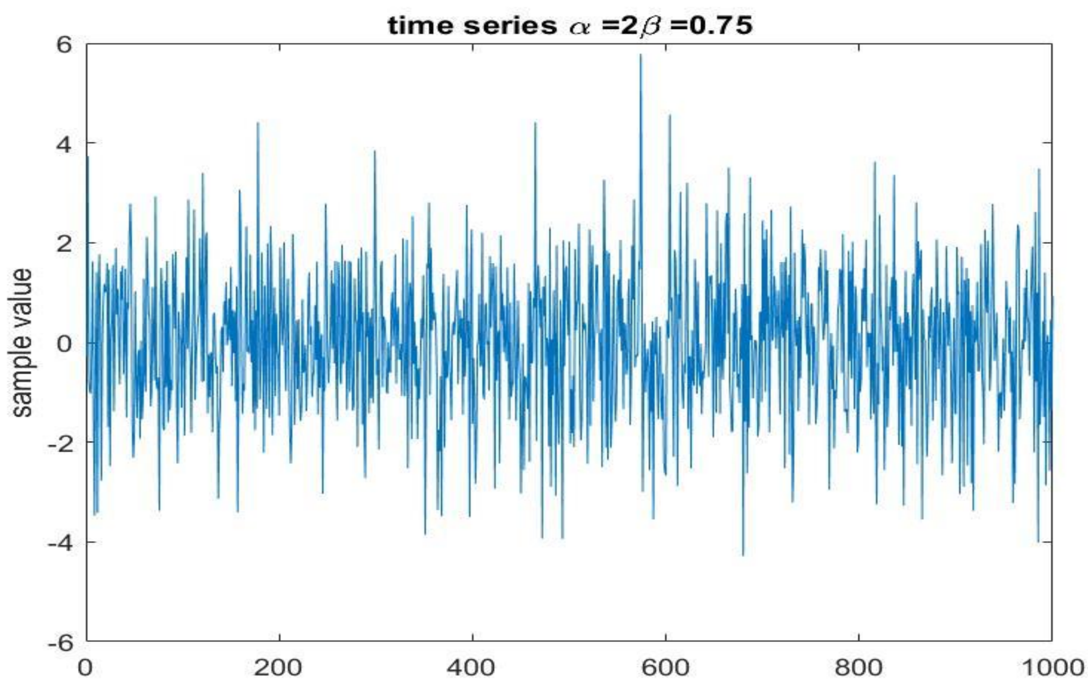


α =1β =0.75
histogram
theoretical
0.35 0.3 0.25 0.2 0.15 0.1 0.05 0
-20 -15 -10 -5 0 5 10 15 20
time series α =1β =0.75
sample value
100 0 -100 -200 -300 -400 -500 -600 -700 -800 -900
0 200 400 600 800 1000


α =1.8β =0.75
histogram
theoretical
0.3
0.25
0.2
0.15
0.1
0.05
0
-20
-15
-10
-5
0
5
10
15
20


time series α =1.8β =0.75
sample value
30
25
20
15
10
5
0
-5
-10
-15
0
200
400
600
800
1000





From the above figures, we could observe that when the value of α is 0.5 and α is 1 there is a thick tail because of skewness. We could see that, with the decrease in α, we got less oscillating, and the sample magnitude can be really large. Moreover, because of the change of skewness, it seems that the oscillation starts earlier.

Sample generation for Alpha-Stable

```
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
```

PDF Generation for alpha stable

```
function p = stblpdf(x,alpha,beta,gam,delta,varargin)
%P = STBLPDF(X,ALPHA,BETA,GAM,DELTA) returns the pdf of the stable
% distribtuion with characteristic exponent ALPHA, skewness BETA, scale
% parameter GAM, and location parameter DELTA, at the values in X.  We use
% the parameterization of stable distribtuions used in [2] - The
% characteristic function phi(t) of a S(ALPHA,BETA,GAM,DELTA)
% random variable has the form
%
% phi(t) = exp(-GAM^ALPHA |t|^ALPHA [1 - i BETA (tan(pi ALPHA/2) sign(t)]
%               + i DELTA t )  if alpha ~= 1
%
% phi(t) = exp(-GAM |t| [ 1 + i BETA (2/pi) (sign(t)) log|t|] + i DELTA t
%                            if alpha = 1
%
% The size of P is the size of X.  ALPHA,BETA,GAM and DELTA must be scalars
%
%P = STBLPDF(X,ALPHA,BETA,GAM,DELTA,TOL) computes the pdf to within an
% absolute error of TOL.
%
% The algorithm works by computing the numerical integrals in Theorem
% 1 in [1] using MATLAB's QUADV function.  The integrands
% are smooth non-negative functions, but for certain parameter values
% can have sharp peaks which might be missed.  To avoid this, STBLEPDF
% locates the maximum of this integrand and breaks the integral into two
% pieces centered around this maximum (this is exactly the idea suggested
% in [1] ).
%
% If abs(ALPHA - 1) < 1e-5,  ALPHA is rounded to 1.
%
%P = STBLPDF(...,'quick') 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 (for quick or not quick)
%
% See also: STBLRND, STBLCDF, STBLINV, STBLFIT
%
% References:
%
% [1] J. P. Nolan (1997)
%     "Numerical Calculation of Stable Densities and Distribution
%     Functions"  Commun. Statist. - Stochastic Modles, 13(4), 759-774
%
% [2] G Samorodnitsky, MS Taqqu (1994)
%     "Stable non-Gaussian random processes: stochastic models with
%      infinite variance"  CRC Press
%

if nargin < 5
    error('stblpdf:TooFewInputs','Requires at least five input arguments.');
end

% Check parameters
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

% warnings will happen during call to QUADV, and it's okay
warning('off');

% 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);


    % x > zeta, calculate integral using QUADV
    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

    % x = zeta, this is easy
    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
    % This doesn't need to be rescaled.
    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
            % Use a default tolerance of 1e-12
        logg = @(theta) xterm(:) + logV(theta) ;
        R = repmat([-theta0, pi/2 ],numel(xterm),1);
        theta2 = bisectionSolver(logg,R,1-beta);
        theta2 = reshape(theta2,size(xterm));
        % 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);
        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




function X = bisectionSolver(f,R,alpha,varargin)
% Solves equation g(theta) - 1 = 0 in STBLPDF using a vectorized bisection
% method and a tolerance of 1e-5.  The solution to this
% equation is used to increase accuracy in the calculation of a numerical
% integral.
%
% If alpha ~= 1 and 0 <= beta < 1, the equation always has a solution
```

```
%
% If alpha > 1 and beta <= 1, then g is monotone decreasing
%
% If alpha < 1 and beta < 1, then g is monotone increasing
%
% If alpha = 1,  g is monotone increasing if beta > 0 and monotone
% decreasing is beta < 0.  Input alpha = 1 - beta to get desired results.
%
%


if nargin < 2
    error('bisectionSolver:TooFewInputs','Requires at least two input arguments.');
end

noSolution = false(size(R,1));
% if ~isempty(varargin)
%     beta = varargin{1};
%     xshift = varargin{2};
%     if abs(beta) == 1
%         V0=(1/alpha)^(alpha/(alpha-1))*(1-alpha)*cos(alpha*pi/2)*xshift;
%         if alpha > 1
%             noSolution = V0 - 1 %>= 0;
%         elseif alpha < 1
%             noSolution = V0 - 1 %<= 0;
%         end
%     end
% end

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

end
```