**Final Project Report**

# Image Manipulation Via MCMC

## Summary

A Markov chain is a stochastic model describing a sequence of possible events in which the probability of each event depends only on the state attained in the previous event. It helps us to predict future outcomes based solely upon the present outcome. It has two characters: irreducibility and ergodicity. A Markov chain in which every state can be reached from every other state is called as an irreducible Markov chain. A random process is said to be ergodic if seeing one realization tells us everything about the statistics of the process. A Markov chain is said to be ergodic if it is both irreducible and aperiodic. Markov chain has several applications as statistical models of real-world processes like currency exchange rates and population dynamics etc. Markov processes forms the base for Markov chain Monte Carlo simulations.
Markov Chain Monte Carlo (MCMC) consists of a class of algorithms for sampling from a probability distribution. One of the algorithms that belongs to MCMC is the Metropolis-Hastings algorithm. The Metropolis-Hastings algorithm generates a Markov chain using a proposal density for new steps and contains a method to accept or reject the proposed new step. The MH algorithm is called random walk because it generates a new position from its current position by using a random transition logic. The two main conditions required to make this transition logic a success are: 1) $\int g(x'|x)dx' = 1 \ \forall x$ and 2) $g(x'|x) = g(x|x')$

In this project, as I am running all tasks individually. Hence, I have provided code for each task right after the explanations. It is followed by the outputs.

## Steps followed for Implementation:

### Task 1: Implementation of Metropolis-Hastings for Burr Distribution

In order to become familiar with Metropolis-Hastings algorithm, we are first going to implement it on a simple scalar random variable. And to see if the Markov Chain eventually reaches the Burr distribution, an estimated PDF for Markov chain can be plotted and compared with the real PDF.

There are many ways in which we can select conditional PDF $g(x'|x)$. And $N(x, 1)$ adds symmetric randomness with a spread of 1 standard deviation on either side. And as it is symmetric, the probability of going from state-1 to state-2 is equal to the probability of going from state-2 to state-1. Hence the conditional distribution satisfies the condition $g(x'|x) = g(x|x')$.

**In MATLAB:** The following steps have been used to implement Task 1.

1) Assigning Burr distribution function to f. As the burr distribution function exists only for values greater that equal to 0, the condition must be specified.
2) Initializing the number of samples to 10000 and creating an array to hold all the sample values. This is the Markov chain.
3) I have initialized my first element of Markov chain to 0 as it seems best. As I increase the value of s(1), the Markov chain very gradually took longer time to mix.
4) Calculate $x'$ as discussed above using the formula; = s(i-1) + randn*1 where s(i-1) is the previous value.
5) Calculate the alpha ratio $\frac{f(x\prime)}{f(x)}$ using the formula $\frac{\mathrm{f(xprime)}}{\mathrm{f(s(i-1))}}$.
6) Now, based on the ratio either accept the sample or reject the sample. The higher the alpha, higher the chances of acceptance.
7) Repeat steps 4, 5, 6 10000 times and 100000 times.
8) Plot the graph for estimated PDF of the Markov chain and the real PDF. The estimated PDF for Markov chain is obtained exactly as in Midterm Project.

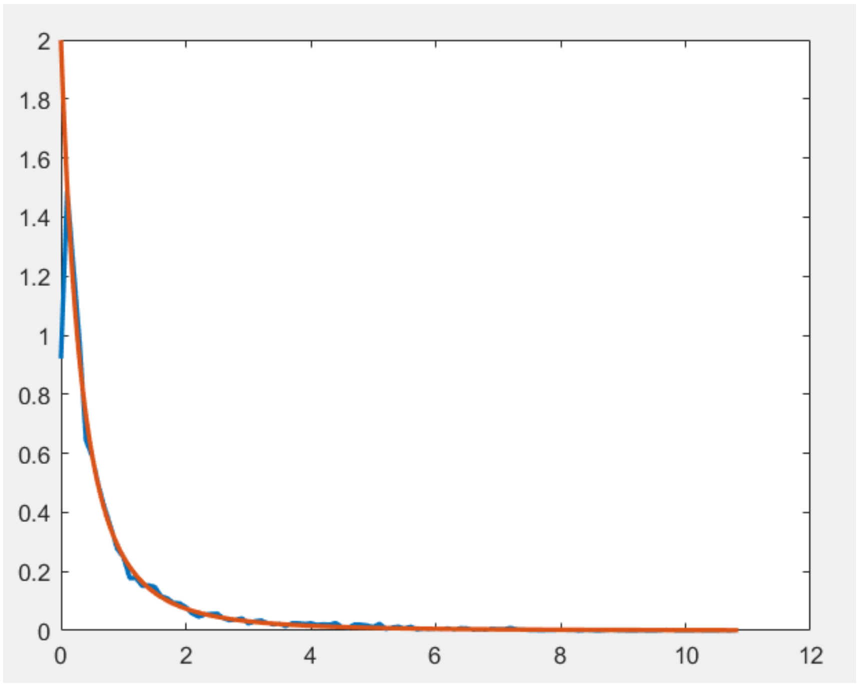
As the estimated PDF and real PDF are very similar, it proves that the two conditions(as discussed in summary) are satisfied.

**Code:**

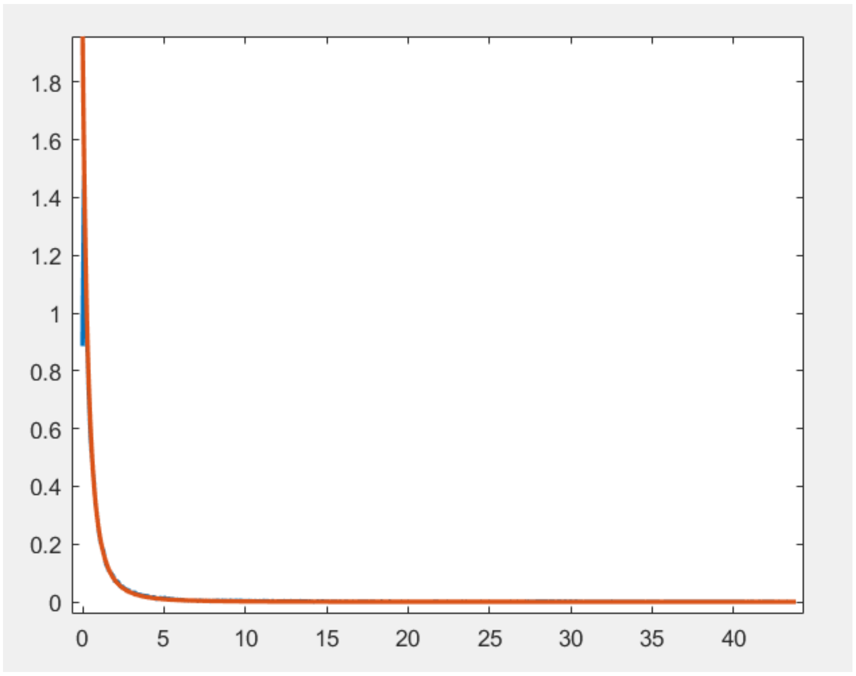
```
%TASK 1:
%Metropolis-Hastings implementation for Burr Distribution
f = @(x) 1/(1+x)^3*(x>=0); %Burr function for all x greater than or equal to 0
rng(0); %setting seed for rand generation
nsamples = 10000; %allocating memory size for 100000 samples
s = zeros(nsamples,1); %initializing sample(Markov Chain) array
s(1) = 0; %setting the value of the first element to 0
for i = 2:nsamples %Iterate from 2nd index of sample array till max number of samples
    xprime = s(i-1) + randn*1; %Find x' value by adding normally distributed random value with SD = 1 to previous
sample
    alpha = f(xprime)/f(s(i-1)); %Find the alpha ratio, based on which the sample is either accepted or rejected
    if rand < alpha %if alpha>=1, the sample will be accepted. else will be rejected
        s(i) = xprime; %if true, then set the current value to x'
    else
        s(i) = s(i-1); %else set the current value to past value
    end
end

%Plotting estimate PDF for sample s and comparing it with real PDF
del = 0.1; %delta value, a small range for displacement
smin = min(s); %minimum value from the baseline samples
smax = max(s); %maximum value from the baseline samples
bincenters = smin:del:smax; %a vector containing all the values from smin to smax with an interval of del
pdf_est = zeros(1,length(bincenters)); %initializing the vector pdf_est to the same size as that of bincenters
for i=1:length(bincenters) %each value of bincenters must be compared
 pdf_est(i) = nnz(s>bincenters(i)-del/2 & s<=bincenters(i)+del/2)/nsamples/del; %calculating the number of
nonzeros between bincenters(i)-del/2 bincenters(i)+del/2
end %Probability_est = count/nsamples. PDF est = probability_est/del
x_real = linspace(smin,smax,1000); %create a new vectore with 1000 values between smin,smax linearly
f_x = @(x)1./(1+x).^3; %To obtain normalizing constant
c = 1./integral(f_x, 0 ,Inf); %Calculation of Normalizing Constant;
```

```
pdf_real = c./((1+x_real).^3); %claculation of PDF real
plot(bincenters, pdf_est, x_real, pdf_real,'LineWidth',2); %plot graph of both real and estimated PDF for
comparision
```

## Output for 10000 runs:



## Output for 100000 runs:



## Task 2: To see how quickly the Markov chain mixes

In order to understand how quickly the Markov chain mixes, we are going to implement Metropolis-Hastings algorithm for n=1,3,5,10,30,100. And plotting an estimated PDF against real PDF for each of these n values shows us how quickly the Markov chain mixes.

**In MATLAB:** Taking an example of n = 5. It is similar for all the other cases as well.

1) Initialize the value of n = 5
2) Initialize an s array to obtain the value of X(5).
3) Initialize the s(1) to 0
4) Initialize a new_samples array that stores all the values obtained for X(5)
5) Initialize new_samples(1) = s(1)
6) Calculate $x'$ as discussed above using the formula; = s(i-1) + randn*1 where s(i-1) is the previous value.
7) Calculate the alpha ratio $\frac{f(x\prime)}{f(x)}$ using the formula $\frac{\mathrm{f(xprime)}}{\mathrm{f(s(i-1))}}$.
8) Now, based on the ratio either accept the sample or reject the sample. Higher the alpha, higher the chances of acceptance.
9) Repeat steps 6,7 and 8 n times. This will give the X(3) value.
10) Append the obtained X(3) value to new_samples. And initialize s(1) to the latest X(3) value.
11) Repeat steps 6,7,8,9 and 10 10,000 times to obtain required number of samples(10000)
12) Plot a graph of estimated PDF for new_samples MC and the read PDF.

Repeat all the above steps for all values of n except n =1. As there is no need to have another mc to calculate X(1).

As seen in the below plots, X[n] -> f as n->infinity. The estimated PDF will be very close to the real PDF as the number of samples are increased.

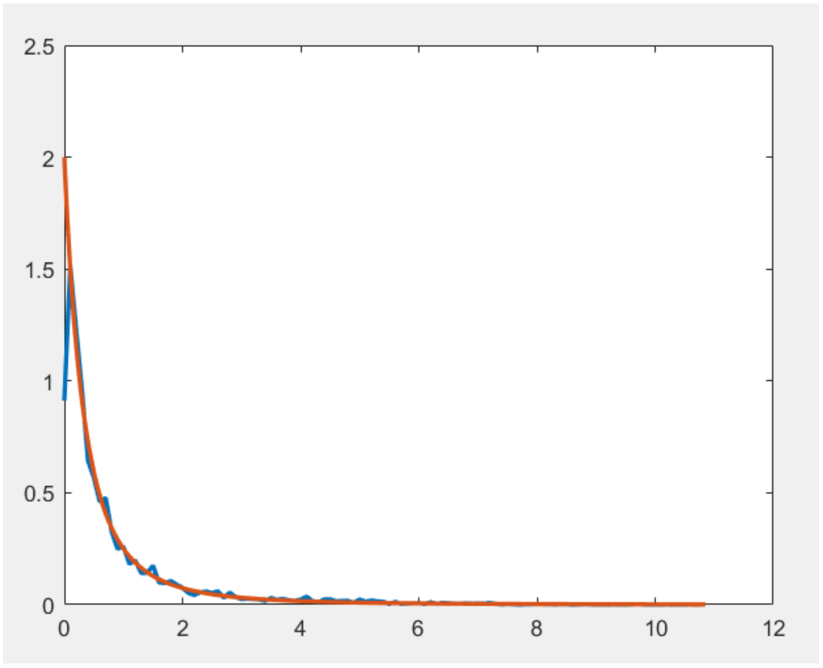
For n=5, each value in mc is 5 steps ahead of its previous element. Similarly, when n = 30, each value in its mc are 30 steps ahead of its previous element. Hence as n increases, the Markov chain mixes faster.

### Code for n = 1:

```
%TASK 2

f = @(x) 1/(1+x)^3*(x>=0); %Burr function for all x greater than or equal to 0
rng(0); %setting seed for rand generation

%------X(1)
n = 1; %assigning the value of n
s = zeros(n,1); %initializing sample array(MC) size to n
s(1) = 0; %setting the value of the sample to 0
new_samples = zeros(10000,1); %Initializing new markov chain to hold values of X(1)
for j = 1:length(new_samples) %Generating X(1) value 10000 times.
   xprime = s(1) + randn*1; %Find x' value by adding normally distributed random value with SD = 1 to previous
sample
```

```
    alpha = f(xprime)/f(s(1)); %Find the alpha ratio, based on which the sample is either accepted or rejected
    if rand < alpha %if alpha>=1, the sample will be accepted. else will be rejected
        s(1) = xprime; %if true, then set the current value to x' (also re-initializes first element to new value)
    else
        s(1) = s(1); %else set the current value to past value
    end
    new_samples(j) = s(1); %Append the new value to new_samples(MC)
end %Generate X(1) 10000 times

%Plotting estimate PDF for sample s and comparing it with real PDF
del = 0.1; %delta value, a small range for displacement
smin = min(new_samples); %minimum value from the new_samples
smax = max(new_samples); %maximum value from the new_samples
bincenters = smin:del:smax; %a vector containing all the values from smin to smax with an interval of del
pdf_est = zeros(1,length(bincenters)); %initializing the vector pdf_est to the same size as that of bincenters
for i=1:length(bincenters) %each value of bincenters must be compared
 pdf_est(i) = nnz(new_samples>bincenters(i)-del/2 & new_samples<=bincenters(i)+del/2)/length(new_samples)/del;
%calculating the number of nonzeros between bincenters(i)-del/2 bincenters(i)+del/2
end %Probability_est = count/nsamples. PDF est = probability_est/del
x_real = linspace(smin,smax,1000); %create a new vector with 1000 values between smin,smax linearly
f_x = @(x)1./(1+x).^3; %To obtain normalizing constant
c = 1./integral(f_x, 0 ,Inf); %Calculation of Normalizing Constant;
pdf_real = c./((1+x_real).^3); %calculation of PDF real
plot(bincenters, pdf_est, x_real, pdf_real,'LineWidth',2); %plot graph of both real and estimated PDF for comparison
```

## Output for n = 1:

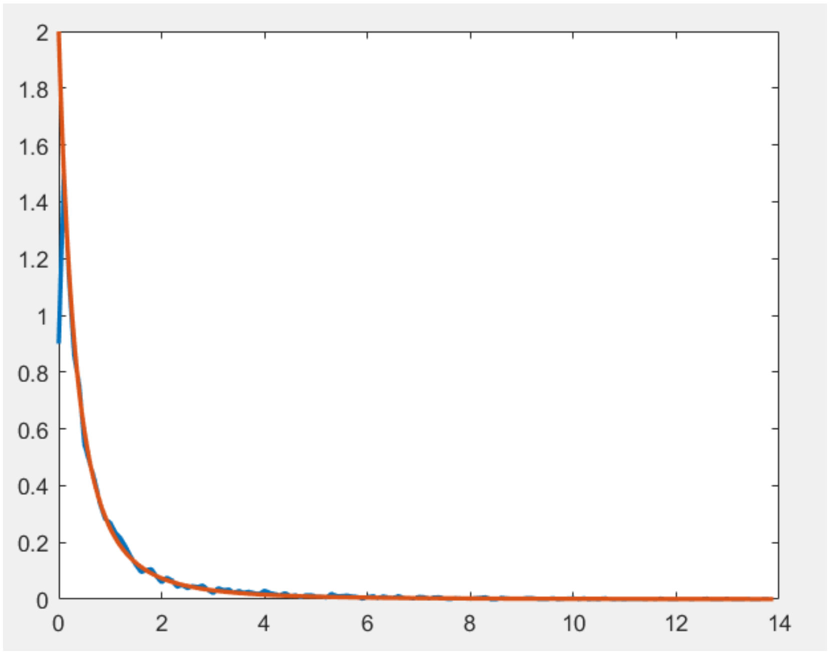


### Code for n = 3:

```
%------X(3)

n = 3; %assigning the value of n
```

```
s = zeros(n,1); %initializing sample array(MC) size to n
s(1) = 0; %setting the value of the sample to 0
new_samples = zeros(10000,1); %Initializing new markov chain to hold values of X(1)
for j = 1:length(new_samples) %Generating X(3) value 10000 times.
  for i = 2:n %run 3 times to obtain X(3)
    xprime = s(i-1) + randn*1; %Find x' value by adding normally distributed random value with SD = 1 to
previous sample
    alpha = f(xprime)/f(s(i-1)); %Find the alpha ratio, based on which the sample is either accepted or rejected
    if rand < alpha %if alpha>=1, the sample will be accepted. else will be rejected
      s(i) = xprime; %if true, then set the current value to x'
    else
      s(i) = s(i-1); %else set the current value to past value
    end
  end
  new_samples(j) = s(i); %Assign X(3) value obtained from above to MC
  s(1) = s(i); %Reinitialize the first element to the latest X(3) value
end
%Plotting estimate PDF for sample s and comparing it with real PDF
del = 0.1; %delta value, a small range for displacement
smin = min(new_samples); %minimum value from the baseline samples
smax = max(new_samples); %maximum value from the baseline samples
bincenters = smin:del:smax; %a vector containing all the values from smin to smax with an interval of del
pdf_est = zeros(1,length(bincenters)); %initializing the vector pdf_est to the same size as that of bincenters
for i=1:length(bincenters) %each value of bincenters must be compared
 pdf_est(i) = nnz(new_samples>bincenters(i)-del/2 & new_samples<=bincenters(i)+del/2)/length(new_samples)/del;
%calculating the number of nonzeros between bincenters(i)-del/2 bincenters(i)+del/2
end %Probability_est = count/nsamples. PDF est = probability_est/del
x_real = linspace(smin,smax,1000); %create a new vector with 1000 values between smin,smax linearly
f_x = @(x)1./(1+x).^3; %To obtain normalizing constant
c = 1./integral(f_x, 0 ,Inf); %Calculation of Normalizing Constant
pdf_real = c./((1+x_real).^3); %calculation of PDF real
plot(bincenters, pdf_est, x_real, pdf_real,'LineWidth',2); %plot graph of both real and estimated PDF for comparison
```

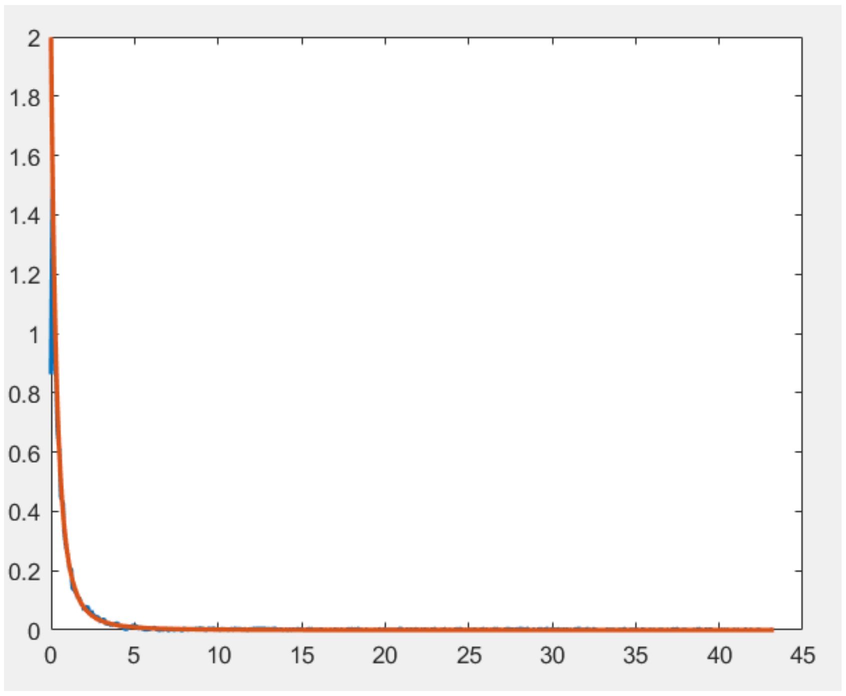
## Output for n = 3:



## Code for n = 5:

```
%------X(5)

n = 5; %assigning the value of n
s = zeros(n,1); %initializing sample array(MC) size to n
s(1) = 0; %setting the value of the sample to 0
new_samples = zeros(10000,1); %Initializing new markov chain to hold values of X(1)
for j = 1:length(new_samples) %Generating X(3) value 10000 times.
  for i = 2:n %run 3 times to obtain X(3)
      xprime = s(i-1) + randn*1; %Find x' value by adding normally distributed random value with SD = 1 to
previous sample
      alpha = f(xprime)/f(s(i-1)); %Find the alpha ratio, based on which the sample is either accepted or rejected
      if rand < alpha %if alpha>=1, the sample will be accepted. else will be rejected
        s(i) = xprime; %if true, then set the current value to x'
      else
        s(i) = s(i-1); %else set the current value to past value
      end
  end
  new_samples(j) = s(i); %Assign X(3) value obtained from above to MC
  s(1) = s(i); %Reinitialize the first element to the latest X(3) value
end

%Plotting estimate PDF for sample s and comparing it with real PDF
del = 0.1; %delta value, a small range for displacement
smin = min(new_samples); %minimum value from the baseline samples
smax = max(new_samples); %maximum value from the baseline samples
bincenters = smin:del:smax; %a vector containing all the values from smin to smax with an interval of del
pdf_est = zeros(1,length(bincenters)); %initializing the vector pdf_est to the same size as that of bincenters
for i=1:length(bincenters) %each value of bincenters must be compared
 pdf_est(i) = nnz(new_samples>bincenters(i)-del/2 & new_samples<=bincenters(i)+del/2)/length(new_samples)/del;
%calculating the number of nonzeros between bincenters(i)-del/2 bincenters(i)+del/2
end %Probability_est = count/nsamples. PDF est = probability_est/del
x_real = linspace(smin,smax,1000); %create a new vector with 1000 values between smin,smax linearly
f_x = @(x)1./(1+x).^3; %To obtain normalizing constant
c = 1./integral(f_x, 0 ,Inf); %Calculation of Normalizing Constant
pdf_real = c./((1+x_real).^3); %calculation of PDF real
plot(bincenters, pdf_est, x_real, pdf_real,'LineWidth',2); %plot graph of both real and estimated PDF for comparison
```

## Output for n = 5:

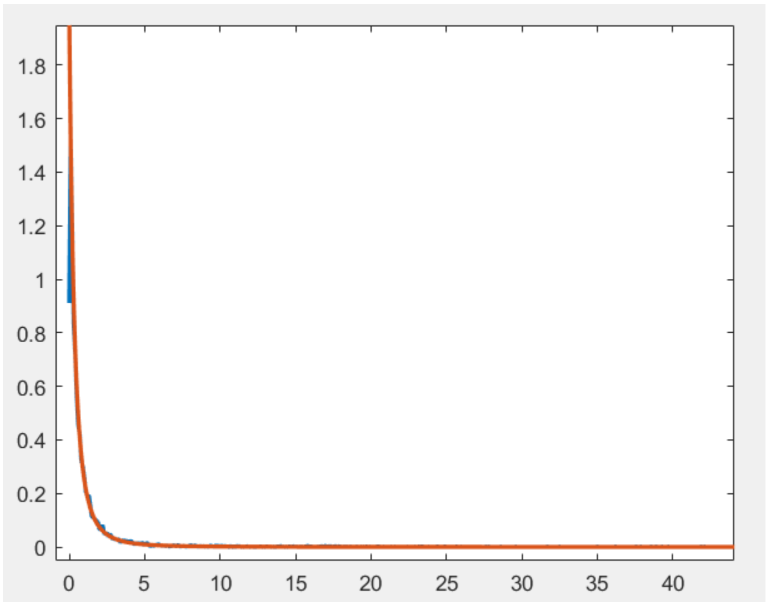


## Code for n = 10:

```
%------X(10)

n = 10; %assigning the value of n
s = zeros(n,1); %initializing sample array(MC) size to n
s(1) = 0; %setting the value of the sample to 0
new_samples = zeros(10000,1); %Initializing new markov chain to hold values of X(1)
for j = 1:length(new_samples) %Generating X(3) value 10000 times.
    for i = 2:n %run 3 times to obtain X(3)
        xprime = s(i-1) + randn*1; %Find x' value by adding normally distributed random value with SD = 1 to
previous sample
        alpha = f(xprime)/f(s(i-1)); %Find the alpha ratio, based on which the sample is either accepted or rejected
        if rand < alpha %if alpha>=1, the sample will be accepted. else will be rejected
            s(i) = xprime; %if true, then set the current value to x'
        else
            s(i) = s(i-1); %else set the current value to past value
        end
    end
    new_samples(j) = s(i); %Assign X(3) value obtained from above to MC
    s(1) = s(i); %Reinitialize the first element to the latest X(3) value
end
%Plotting estimate PDF for sample s and comparing it with real PDF
del = 0.1; %delta value, a small range for displacement
smin = min(new_samples); %minimum value from the baseline samples
smax = max(new_samples); %maximum value from the baseline samples
bincenters = smin:del:smax; %a vector containing all the values from smin to smax with an interval of del
pdf_est = zeros(1,length(bincenters)); %initializing the vector pdf_est to the same size as that of bincenters
for i=1:length(bincenters) %each value of bincenters must be compared
```

```
 pdf_est(i) = nnz(new_samples>bincenters(i)-del/2 & new_samples<=bincenters(i)+del/2)/length(new_samples)/del;
%calculating the number of nonzeros between bincenters(i)-del/2 bincenters(i)+del/2
end %Probability_est = count/nsamples. PDF est = probability_est/del
x_real = linspace(smin,smax,1000); %create a new vector with 1000 values between smin,smax linearly
f_x = @(x)1./(1+x).^3; %To obtain normalizing constant
c = 1./integral(f_x, 0 ,Inf); %Calculation of Normalizing Constant
pdf_real = c./((1+x_real).^3); %calculation of PDF real
plot(bincenters, pdf_est, x_real, pdf_real,'LineWidth',2); %plot graph of both real and estimated PDF for comparison
```

## Output for n = 10:

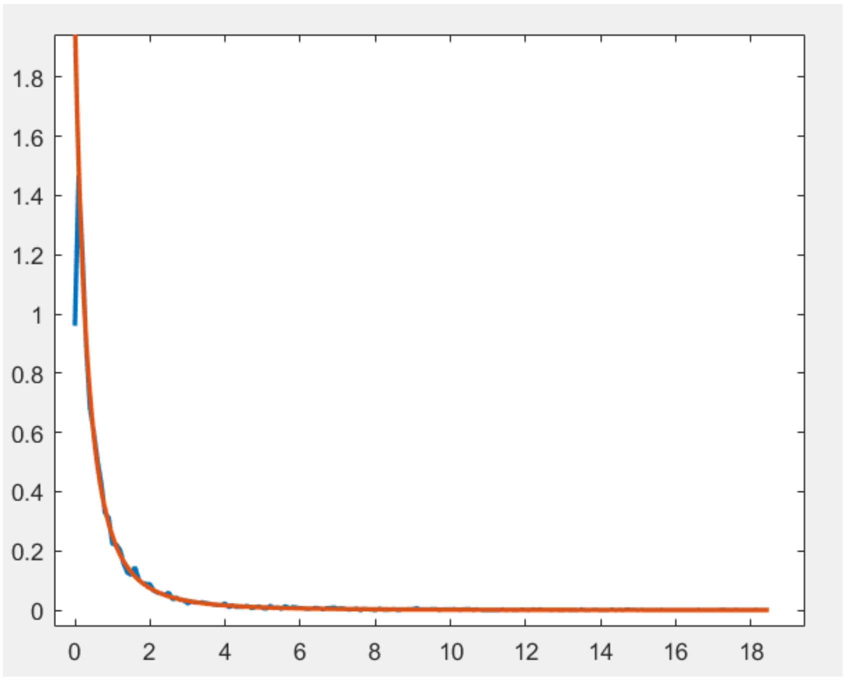


## Code for n = 30:

```
%------X(30)

n = 30; %assigning the value of n
s = zeros(n,1); %initializing sample array(MC) size to n
s(1) = 0; %setting the value of the sample to 0
new_samples = zeros(10000,1); %Initializing new markov chain to hold values of X(1)
for j = 1:length(new_samples) %Generating X(3) value 10000 times.
    for i = 2:n %run 3 times to obtain X(3)
        xprime = s(i-1) + randn*1; %Find x' value by adding normally distributed random value with SD = 1 to
previous sample
        alpha = f(xprime)/f(s(i-1)); %Find the alpha ratio, based on which the sample is either accepted or rejected
        if rand < alpha %if alpha>=1, the sample will be accepted. else will be rejected
            s(i) = xprime; %if true, then set the current value to x'
        else
            s(i) = s(i-1); %else set the current value to past value
        end
    end
    new_samples(j) = s(i); %Assign X(3) value obtained from above to MC
    s(1) = s(i); %Reinitialize the first element to the latest X(3) value
end

%Plotting estimate PDF for sample s and comparing it with real PDF
```

```
del = 0.1; %delta value, a small range for displacement
smin = min(new_samples); %minimum value from the baseline samples
smax = max(new_samples); %maximum value from the baseline samples
bincenters = smin:del:smax; %a vector containing all the values from smin to smax with an interval of del
pdf_est = zeros(1,length(bincenters)); %initializing the vector pdf_est to the same size as that of bincenters
for i=1:length(bincenters) %each value of bincenters must be compared
 pdf_est(i) = nnz(new_samples>bincenters(i)-del/2 & new_samples<=bincenters(i)+del/2)/length(new_samples)/del;
%calculating the number of nonzeros between bincenters(i)-del/2 bincenters(i)+del/2
end %Probability_est = count/nsamples. PDF est = probability_est/del
x_real = linspace(smin,smax,1000); %create a new vector with 1000 values between smin,smax linearly
f_x = @(x)1./(1+x).^3; %To obtain normalizing constant
c = 1./integral(f_x, 0 ,Inf); %Calculation of Normalizing Constant
pdf_real = c./((1+x_real).^3); %calculation of PDF real
plot(bincenters, pdf_est, x_real, pdf_real,'LineWidth',2); %plot graph of both real and estimated PDF for comparison
```

## Output for n = 30:

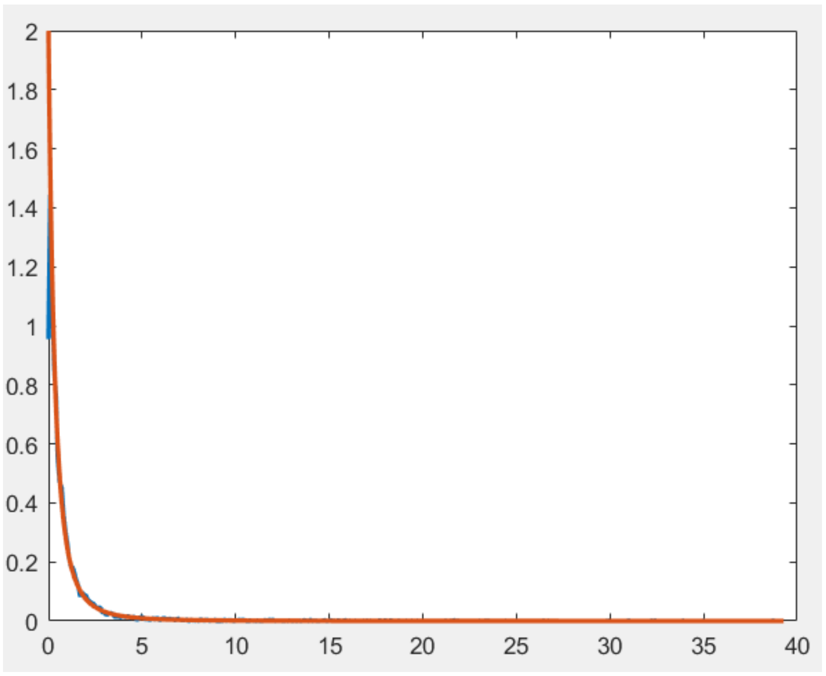


## Code for n = 100:

```
%------X(100)

n = 100; %assigning the value of n
s = zeros(n,1); %initializing sample array(MC) size to n
s(1) = 0; %setting the value of the sample to 0
new_samples = zeros(10000,1); %Initializing new markov chain to hold values of X(1)
for j = 1:length(new_samples) %Generating X(3) value 10000 times.
    for i = 2:n %run 3 times to obtain X(3)
        xprime = s(i-1) + randn*1; %Find x' value by adding normally distributed random value with SD = 1 to
previous sample
        alpha = f(xprime)/f(s(i-1)); %Find the alpha ratio, based on which the sample is either accepted or rejected
        if rand < alpha %if alpha>=1, the sample will be accepted. else will be rejected
            s(i) = xprime; %if true, then set the current value to x'
        else
```

```
            s(i) = s(i-1); %else set the current value to past value
        end
    end
    new_samples(j) = s(i); %Assign X(3) value obtained from above to MC
    s(1) = s(i); %Reinitialize the first element to the latest X(3) value
end

%disp(new_samples);

%Plotting estimate PDF for sample s and comparing it with real PDF
del = 0.1; %delta value, a small range for displacement
smin = min(new_samples); %minimum value from the baseline samples
smax = max(new_samples); %maximum value from the baseline samples
bincenters = smin:del:smax; %a vector containing all the values from smin to smax with an interval of del
pdf_est = zeros(1,length(bincenters)); %initializing the vector pdf_est to the same size as that of bincenters
for i=1:length(bincenters) %each value of bincenters must be compared
 pdf_est(i) = nnz(new_samples>bincenters(i)-del/2 & new_samples<=bincenters(i)+del/2)/length(new_samples)/del;
%calculating the number of nonzeros between bincenters(i)-del/2 bincenters(i)+del/2
end %Probability_est = count/nsamples. PDF est = probability_est/del
x_real = linspace(smin,smax,1000); %create a new vector with 1000 values between smin,smax linearly
f_x = @(x)1./(1+x).^3; %To obtain normalizing constant
c = 1./integral(f_x, 0 ,Inf); %Calculation of Normalizing Constant
pdf_real = c./((1+x_real).^3); %calculation of PDF real
plot(bincenters, pdf_est, x_real, pdf_real,'LineWidth',2); %plot graph of both real and estimated PDF for comparison
```

## Output for n = 100:



## Task 3: Effect of g on Markov chain mixing

The choice of g is very important as it determines how quickly the Markov chain mixes. In this task I am changing the g for different values of n.

For Burr distribution, $g(x'|x) = N(x, 10)$ mixes fastest of all.

All the distributions satisfy the condition $g(x'|x) = g(x|x')$. It is clearly seen when $g(x'|x) = N(x, 10)$ and $g(x'|x) = N(x, 0.1)$. For $g(x'|x) = U(x - 0.5, x + 0.5)$, though it satisfies the condition, it is taking very long to mix. As it is not able to reach the mid value between x-0.5 and x+0.5.

**In MATLAB:** I have repeated the code from task 2 and changed the g(x'|x) values for each value of n = 1,3,5,10,30,100

**Code for $g(x'|x) = N(x, 10)$ *and* $n = 1, 3, 5, 10, 30, 100$:**

```
%TASK 3

f = @(x) 1/(1+x)^3*(x>=0); %Burr function for all x greater than or equal to 0
rng(0); %setting seed for rand generation

%------g(x'|x) = N(x,10) and n = 1

n = 1; %assigning the value of n
s = zeros(n,1); %initializing sample array(MC) size to n
s(1) = 0; %setting the value of the sample to 0
new_samples = zeros(10000,1); %Initializing new markov chain to hold values of X(1)

for j = 1:length(new_samples) %Generating X(1) value 10000 times.
   xprime = s(1) + randn*10; %Find x' value by adding normally distributed random value with SD = 10 to previous
sample
   alpha = f(xprime)/f(s(1)); %Find the alpha ratio, based on which the sample is either accepted or rejected
   if rand < alpha %if alpha>=1, the sample will be accepted. else will be rejected
      s(1) = xprime; %if true, then set the current value to x' (aslo re-initializes first element to new value)
   else
      s(1) = s(1); %else set the current value to past value
   end
   new_samples(j) = s(1); %Append the new value to new_samples(MC)
end %Generate X(1) 10000 times

%Plotting estimate PDF for sample s and comparing it with real PDF
del = 0.1; %delta value, a small range for displacement
smin = min(new_samples); %minimum value from the new_samples
smax = max(new_samples); %maximum value from the new_samples
bincenters = smin:del:smax; %a vector containing all the values from smin to smax with an interval of del
pdf_est = zeros(1,length(bincenters)); %initializing the vector pdf_est to the same size as that of bincenters
for i=1:length(bincenters) %each value of bincenters must be compared
 pdf_est(i) = nnz(new_samples>bincenters(i)-del/2 & new_samples<=bincenters(i)+del/2)/length(new_samples)/del;
%calculating the number of nonzeros between bincenters(i)-del/2 bincenters(i)+del/2
end %Probability_est = count/nsamples. PDF est = probability_est/del
x_real = linspace(smin,smax,1000); %create a new vector with 1000 values between smin,smax linearly
```

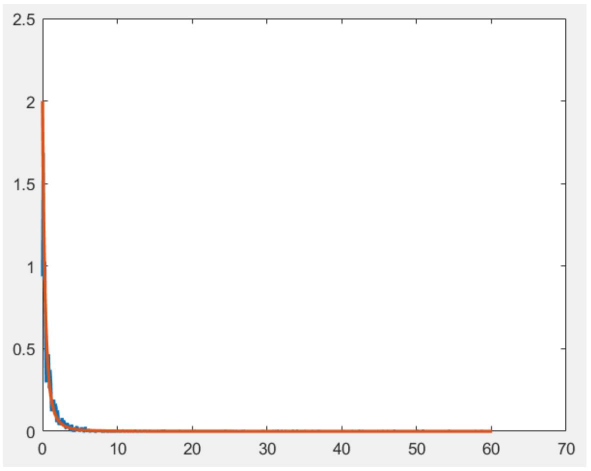
```
f_x = @(x)1./(1+x).^3; %To obtain normalizing constant
c = 1./integral(f_x, 0 ,Inf); %Calculation of Normalizing Constant;
pdf_real = c./((1+x_real).^3); %calculation of PDF real
plot(bincenters, pdf_est, x_real, pdf_real,'LineWidth',2); %plot graph of both real and estimated PDF for comparison


%------g(x'|x) = N(x,10) and n = 3

n = 3; %assigning the value of n
s = zeros(n,1); %initializing sample array(MC) size to n
s(1) = 0; %setting the value of the sample to 0
new_samples = zeros(10000,1); %Initializing new markov chain to hold values of X(1)
for j = 1:length(new_samples) %Generating X(3) value 10000 times.
   for i = 2:n %run 3 times to obtain X(3)
       xprime = s(i-1) + randn*10; %Find x' value by adding normally distributed random value with SD = 10 to
previous sample
       alpha = f(xprime)/f(s(i-1)); %Find the alpha ratio, based on which the sample is either accepted or rejected
       if rand < alpha %if alpha>=1, the sample will be accepted. else will be rejected
          s(i) = xprime; %if true, then set the current value to x'
       else
          s(i) = s(i-1); %else set the current value to past value
       end
   end
   new_samples(j) = s(i); %Assign X(3) value obtained from above to MC
   s(1) = s(i); %Reinitialize the first element to the latest X(3) value
end

%Plotting estimate PDF for sample s and comparing it with real PDF
del = 0.1; %delta value, a small range for displacement
smin = min(new_samples); %minimum value from the baseline samples
smax = max(new_samples); %maximum value from the baseline samples
bincenters = smin:del:smax; %a vector containing all the values from smin to smax with an interval of del
pdf_est = zeros(1,length(bincenters)); %initializing the vector pdf_est to the same size as that of bincenters
for i=1:length(bincenters) %each value of bincenters must be compared
 pdf_est(i) = nnz(new_samples>bincenters(i)-del/2 & new_samples<=bincenters(i)+del/2)/length(new_samples)/del;
%calculating the number of nonzeros between bincenters(i)-del/2 bincenters(i)+del/2
end %Probability_est = count/nsamples. PDF est = probability_est/del
x_real = linspace(smin,smax,1000); %create a new vector with 1000 values between smin,smax linearly
f_x = @(x)1./(1+x).^3; %To obtain normalizing constant
c = 1./integral(f_x, 0 ,Inf); %Calculation of Normalizing Constant
pdf_real = c./((1+x_real).^3); %calculation of PDF real
plot(bincenters, pdf_est, x_real, pdf_real,'LineWidth',2); %plot graph of both real and estimated PDF for comparison


%------g(x'|x) = N(x,10) and n = 5

n = 5; %assigning the value of n
s = zeros(n,1); %initializing sample array(MC) size to n
s(1) = 0; %setting the value of the sample to 0
new_samples = zeros(10000,1); %Initializing new markov chain to hold values of X(1)
for j = 1:length(new_samples) %Generating X(3) value 10000 times.
   for i = 2:n %run 3 times to obtain X(3)
       xprime = s(i-1) + randn*10; %Find x' value by adding normally distributed random value with SD = 10 to
previous sample
       alpha = f(xprime)/f(s(i-1)); %Find the alpha ratio, based on which the sample is either accepted or rejected
       if rand < alpha %if alpha>=1, the sample will be accepted. else will be rejected
```

```
            s(i) = xprime; %if true, then set the current value to x'
        else
            s(i) = s(i-1); %else set the current value to past value
        end
    end
    new_samples(j) = s(i); %Assign X(3) value obtained from above to MC
    s(1) = s(i); %Reinitialize the first element to the latest X(3) value
end

%Plotting estimate PDF for sample s and comparing it with real PDF
del = 0.1; %delta value, a small range for displacement
smin = min(new_samples); %minimum value from the baseline samples
smax = max(new_samples); %maximum value from the baseline samples
bincenters = smin:del:smax; %a vector containing all the values from smin to smax with an interval of del
pdf_est = zeros(1,length(bincenters)); %initializing the vector pdf_est to the same size as that of bincenters
for i=1:length(bincenters) %each value of bincenters must be compared
 pdf_est(i) = nnz(new_samples>bincenters(i)-del/2 & new_samples<=bincenters(i)+del/2)/length(new_samples)/del;
%calculating the number of nonzeros between bincenters(i)-del/2 bincenters(i)+del/2
end %Probability_est = count/nsamples. PDF est = probability_est/del
x_real = linspace(smin,smax,1000); %create a new vector with 1000 values between smin,smax linearly
f_x = @(x)1./(1+x).^3; %To obtain normalizing constant
c = 1./integral(f_x, 0 ,Inf); %Calculation of Normalizing Constant
pdf_real = c./((1+x_real).^3); %calculation of PDF real
plot(bincenters, pdf_est, x_real, pdf_real,'LineWidth',2); %plot graph of both real and estimated PDF for comparison

%------g(x'|x) = N(x,10) and n = 10

n = 10; %assigning the value of n
s = zeros(n,1); %initializing sample array(MC) size to n
s(1) = 0; %setting the value of the sample to 0
new_samples = zeros(10000,1); %Initializing new markov chain to hold values of X(1)
for j = 1:length(new_samples) %Generating X(3) value 10000 times.
    for i = 2:n %run 3 times to obtain X(3)
        xprime = s(i-1) + randn*10; %Find x' value by adding normally distributed random value with SD = 10 to
previous sample
        alpha = f(xprime)/f(s(i-1)); %Find the alpha ratio, based on which the sample is either accepted or rejected
        if rand < alpha %if alpha>=1, the sample will be accepted. else will be rejected
            s(i) = xprime; %if true, then set the current value to x'
        else
            s(i) = s(i-1); %else set the current value to past value
        end
    end
    new_samples(j) = s(i); %Assign X(3) value obtained from above to MC
    s(1) = s(i); %Reinitialize the first element to the latest X(3) value
end

%Plotting estimate PDF for sample s and comparing it with real PDF
del = 0.1; %delta value, a small range for displacement
smin = min(new_samples); %minimum value from the baseline samples
smax = max(new_samples); %maximum value from the baseline samples
bincenters = smin:del:smax; %a vector containing all the values from smin to smax with an interval of del
pdf_est = zeros(1,length(bincenters)); %initializing the vector pdf_est to the same size as that of bincenters
for i=1:length(bincenters) %each value of bincenters must be compared
 pdf_est(i) = nnz(new_samples>bincenters(i)-del/2 & new_samples<=bincenters(i)+del/2)/length(new_samples)/del;
%calculating the number of nonzeros between bincenters(i)-del/2 bincenters(i)+del/2
end %Probability_est = count/nsamples. PDF est = probability_est/del
```

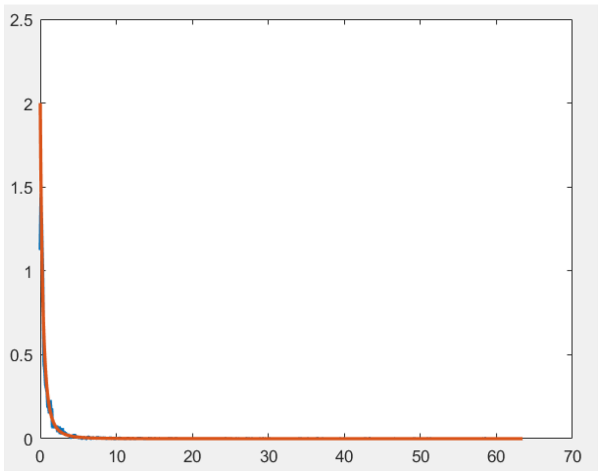
```
x_real = linspace(smin,smax,1000); %create a new vectore with 1000 values between smin,smax linearly
f_x = @(x)1./(1+x).^3; %To obtain normalizing constant
c = 1./integral(f_x, 0 ,Inf); %Calculation of Normalizing Constant
pdf_real = c./((1+x_real).^3); %calculation of PDF real
plot(bincenters, pdf_est, x_real, pdf_real,'LineWidth',2); %plot graph of both real and estimated PDF for comparison

%------g(x'|x) = N(x,10) and n = 30

n = 30; %assigning the value of n
s = zeros(n,1); %initializing sample array(MC) size to n
s(1) = 0; %setting the value of the sample to 0
new_samples = zeros(10000,1); %Initializing new markov chain to hold values of X(1)
for j = 1:length(new_samples) %Generating X(3) value 10000 times.
    for i = 2:n %run 3 times to obtain X(3)
        xprime = s(i-1) + randn*10; %Find x' value by adding normally distributed random value with SD = 10 to
previous sample
        alpha = f(xprime)/f(s(i-1)); %Find the alpha ratio, based on which the sample is either accepted or rejected
        if rand < alpha %if alpha>=1, the sample will be accepted. else will be rejected
            s(i) = xprime; %if true, then set the current value to x'
        else
            s(i) = s(i-1); %else set the current value to past value
        end
    end
    new_samples(j) = s(i); %Assign X(3) value obtained from above to MC
    s(1) = s(i); %Reinitialize the first element to the latest X(3) value
end

%Plotting estimate PDF for sample s and comparing it with real PDF
del = 0.1; %delta value, a small range for displacement
smin = min(new_samples); %minimum value from the baseline samples
smax = max(new_samples); %maximum value from the baseline samples
bincenters = smin:del:smax; %a vector containing all the values from smin to smax with an interval of del
pdf_est = zeros(1,length(bincenters)); %initializing the vector pdf_est to the same size as that of bincenters
for i=1:length(bincenters) %each value of bincenters must be compared
 pdf_est(i) = nnz(new_samples>bincenters(i)-del/2 & new_samples<=bincenters(i)+del/2)/length(new_samples)/del;
%calculating the number of nonzeros between bincenters(i)-del/2 bincenters(i)+del/2
end %Probability_est = count/nsamples. PDF est = probability_est/del
x_real = linspace(smin,smax,1000); %create a new vector with 1000 values between smin,smax linearly
f_x = @(x)1./(1+x).^3; %To obtain normalizing constant
c = 1./integral(f_x, 0 ,Inf); %Calculation of Normalizing Constant
pdf_real = c./((1+x_real).^3); %calculation of PDF real
plot(bincenters, pdf_est, x_real, pdf_real,'LineWidth',2); %plot graph of both real and estimated PDF for comparison


%------g(x'|x) = N(x,10) and n = 100

n = 100; %assigning the value of n
s = zeros(n,1); %initializing sample array(MC) size to n
s(1) = 0; %setting the value of the sample to 0
new_samples = zeros(10000,1); %Initializing new markov chain to hold values of X(1)
for j = 1:length(new_samples) %Generating X(3) value 10000 times.
    for i = 2:n %run 3 times to obtain X(3)
        xprime = s(i-1) + randn*10; %Find x' value by adding normally distributed random value with SD = 10 to
previous sample
        alpha = f(xprime)/f(s(i-1)); %Find the alpha ratio, based on which the sample is either accepted or rejected
        if rand < alpha %if alpha>=1, the sample will be accepted. else will be rejected
```

```
            s(i) = xprime; %if true, then set the current value to x'
        else
            s(i) = s(i-1); %else set the current value to past value
        end
    end
    new_samples(j) = s(i); %Assign X(3) value obtained from above to MC
    s(1) = s(i); %Reinitialize the first element to the latest X(3) value
end

%Plotting estimate PDF for sample s and comparing it with real PDF
del = 0.1; %delta value, a small range for displacement
smin = min(new_samples); %minimum value from the baseline samples
smax = max(new_samples); %maximum value from the baseline samples
bincenters = smin:del:smax; %a vector containing all the values from smin to smax with an interval of del
pdf_est = zeros(1,length(bincenters)); %initializing the vector pdf_est to the same size as that of bincenters
for i=1:length(bincenters) %each value of bincenters must be compared
 pdf_est(i) = nnz(new_samples>bincenters(i)-del/2 & new_samples<=bincenters(i)+del/2)/length(new_samples)/del;
%calculating the number of nonzeros between bincenters(i)-del/2 bincenters(i)+del/2
end %Probability_est = count/nsamples. PDF est = probability_est/del
x_real = linspace(smin,smax,1000); %create a new vector with 1000 values between smin,smax linearly
f_x = @(x)1./(1+x).^3; %To obtain normalizing constant
c = 1./integral(f_x, 0 ,Inf); %Calculation of Normalizing Constant
pdf_real = c./((1+x_real).^3); %calculation of PDF real
plot(bincenters, pdf_est, x_real, pdf_real,'LineWidth',2); %plot graph of both real and estimated PDF for comparison
```

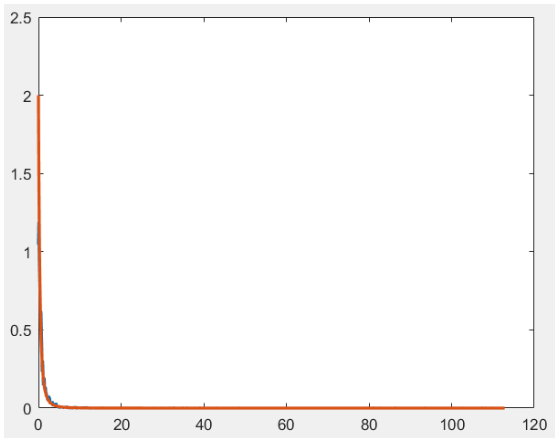
**Output for $g(x'|x) = N(x, 10), n = 1$:**



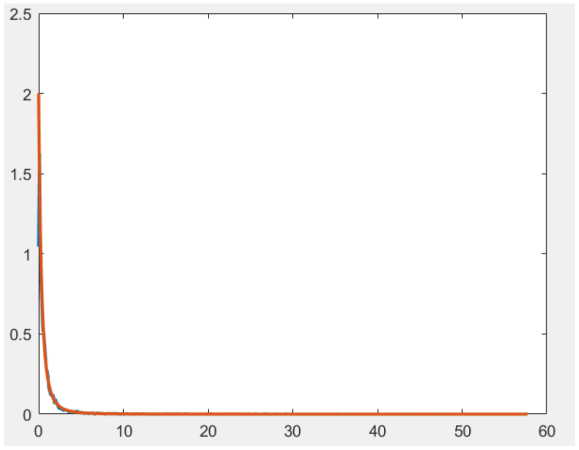
**Output for $g(x'|x) = N(x, 10), n = 3$:**



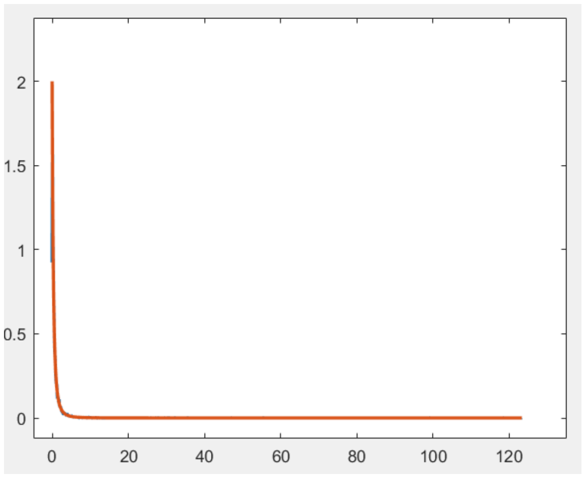
**Output for $g(x'|x) = N(x, 10), n = 5$:**



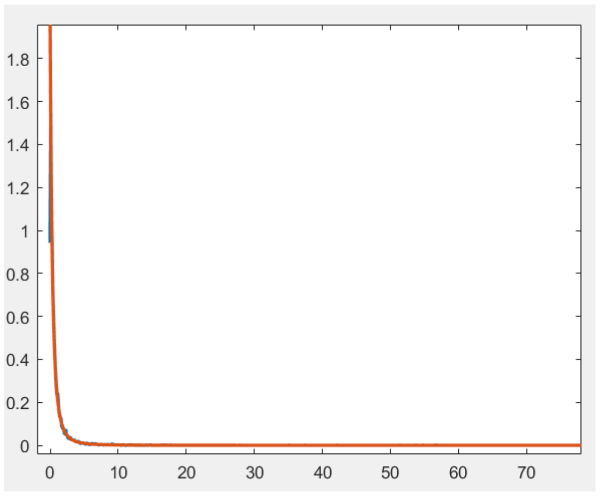
**Output for $g(x'|x) = N(x, 10), n = 10$:**



**Output for $g(x'|x) = N(x, 10), n = 30$:**



**Output for $g(x'|x) = N(x, 10), n = 100$:**



**Code for $g(x'|x) = N(x, 0.1)$ *and* $n = 1, 3, 5, 10, 30, 100$ :**

```
%TASK 3

f = @(x) 1/(1+x)^3*(x>=0); %Burr function for all x greater than or equal to 0
rng(0); %setting seed for rand generation

%------g(x'|x) = N(x,0.1) and n = 1

n = 1; %assigning the value of n
s = zeros(n,1); %initializing sample array(MC) size to n
s(1) = 0; %setting the value of the sample to 0
new_samples = zeros(10000,1); %Initializing new markov chain to hold values of X(1)

for j = 1:length(new_samples) %Generating X(1) value 10000 times.
    xprime = s(1) + randn*0.1; %Find x' value by adding normally distributed random value with SD = 0.1 to
previous sample
```

```
    alpha = f(xprime)/f(s(1)); %Find the alpha ratio, based on which the sample is either accepted or rejected
    if rand < alpha %if alpha>=1, the sample will be accepted. else will be rejected
        s(1) = xprime; %if true, then set the current value to x' (aslo re-initializes first element to new value)
    else
        s(1) = s(1); %else set the current value to past value
    end
    new_samples(j) = s(1); %Append the new value to new_samples(MC)
end %Generate X(1) 10000 times

%Plotting estimate PDF for sample s and comparing it with real PDF
del = 0.1; %delta value, a small range for displacement
smin = min(new_samples); %minimum value from the new_samples
smax = max(new_samples); %maximum value from the new_samples
bincenters = smin:del:smax; %a vector containing all the values from smin to smax with an interval of del
pdf_est = zeros(1,length(bincenters)); %initializing the vector pdf_est to the same size as that of bincenters
for i=1:length(bincenters) %each value of bincenters must be compared
 pdf_est(i) = nnz(new_samples>bincenters(i)-del/2 & new_samples<=bincenters(i)+del/2)/length(new_samples)/del;
%calculating the number of nonzeros between bincenters(i)-del/2 bincenters(i)+del/2
end %Probability_est = count/nsamples. PDF est = probability_est/del
x_real = linspace(smin,smax,1000); %create a new vectore with 1000 values between smin,smax linearly
f_x = @(x)1./(1+x).^3; %To obtain normalizing constant
c = 1./integral(f_x, 0 ,Inf); %Calculation of Normalizing Constant;
pdf_real = c./((1+x_real).^3); %claculation of PDF real
plot(bincenters, pdf_est, x_real, pdf_real,'LineWidth',2); %plot graph of both real and estimated PDF for
comparision


%------g(x'|x) = N(x,0.1) and n = 3

n = 3; %assigning the value of n
s = zeros(n,1); %initializing sample array(MC) size to n
s(1) = 0; %setting the value of the sample to 0
new_samples = zeros(10000,1); %Initializing new markov chain to hold values of X(1)
for j = 1:length(new_samples) %Generating X(3) value 10000 times.
    for i = 2:n %run 3 times to obtain X(3)
        xprime = s(i-1) + randn*0.1; %Find x' value by adding normally distributed random value with SD = 0.1 to
previous sample
        alpha = f(xprime)/f(s(i-1)); %Find the alpha ratio, based on which the sample is either accepted or rejected
        if rand < alpha %if alpha>=1, the sample will be accepted. else will be rejected
            s(i) = xprime; %if true, then set the current value to x'
        else
            s(i) = s(i-1); %else set the current value to past value
        end
    end
    new_samples(j) = s(i); %Assign X(3) value obtained from above to MC
    s(1) = s(i); %Reinitialize the first element to the latest X(3) value
end

%disp(new_samples);

%Plotting estimate PDF for sample s and comparing it with real PDF
del = 0.1; %delta value, a small range for displacement
smin = min(new_samples); %minimum value from the baseline samples
smax = max(new_samples); %maximum value from the baseline samples
bincenters = smin:del:smax; %a vector containing all the values from smin to smax with an interval of del
pdf_est = zeros(1,length(bincenters)); %initializing the vector pdf_est to the same size as that of bincenters
```

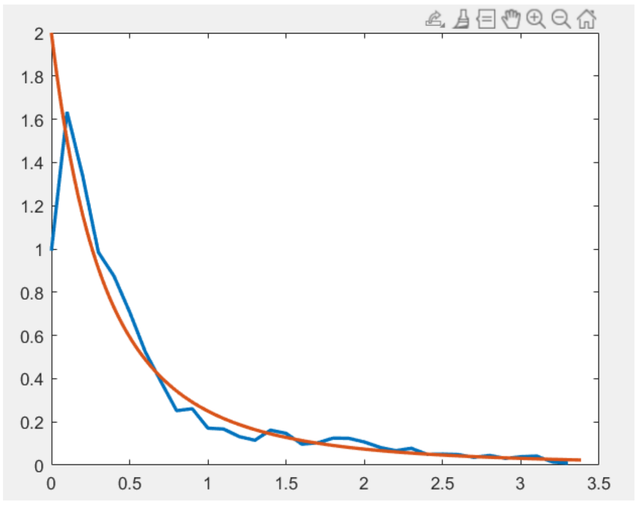
```
for i=1:length(bincenters) %each value of bincenters must be compared
 pdf_est(i) = nnz(new_samples>bincenters(i)-del/2 & new_samples<=bincenters(i)+del/2)/length(new_samples)/del;
%calculating the number of nonzeros between bincenters(i)-del/2 bincenters(i)+del/2
end %Probability_est = count/nsamples. PDF est = probability_est/del
x_real = linspace(smin,smax,1000); %create a new vectore with 1000 values between smin,smax linearly
f_x = @(x)1./(1+x).^3; %To obtain normalizing constant
c = 1./integral(f_x, 0 ,Inf); %Calculation of Normalizing Constant
pdf_real = c./((1+x_real).^3); %claculation of PDF real
plot(bincenters, pdf_est, x_real, pdf_real,'LineWidth',2); %plot graph of both real and estimated PDF for
comparision


%------g(x'|x) = N(x,0.1) and n = 5

n = 5; %assigning the value of n
s = zeros(n,1); %initializing sample array(MC) size to n
s(1) = 0; %setting the value of the sample to 0
new_samples = zeros(10000,1); %Initializing new markov chain to hold values of X(1)
for j = 1:length(new_samples) %Generating X(3) value 10000 times.
    for i = 2:n %run 3 times to obtain X(3)
        xprime = s(i-1) + randn*0.1; %Find x' value by adding normally distributed random value with SD = 0.1 to
previous sample
        alpha = f(xprime)/f(s(i-1)); %Find the alpha ratio, based on which the sample is either accepted or rejected
        if rand < alpha %if alpha>=1, the sample will be accepted. else will be rejected
            s(i) = xprime; %if true, then set the current value to x'
        else
            s(i) = s(i-1); %else set the current value to past value
        end
    end
    new_samples(j) = s(i); %Assign X(3) value obtained from above to MC
    s(1) = s(i); %Reinitialize the first element to the latest X(3) value
end

%disp(new_samples);

%Plotting estimate PDF for sample s and comparing it with real PDF
del = 0.1; %delta value, a small range for displacement
smin = min(new_samples); %minimum value from the baseline samples
smax = max(new_samples); %maximum value from the baseline samples
bincenters = smin:del:smax; %a vector containing all the values from smin to smax with an interval of del
pdf_est = zeros(1,length(bincenters)); %initializing the vector pdf_est to the same size as that of bincenters
for i=1:length(bincenters) %each value of bincenters must be compared
 pdf_est(i) = nnz(new_samples>bincenters(i)-del/2 & new_samples<=bincenters(i)+del/2)/length(new_samples)/del;
%calculating the number of nonzeros between bincenters(i)-del/2 bincenters(i)+del/2
end %Probability_est = count/nsamples. PDF est = probability_est/del
x_real = linspace(smin,smax,1000); %create a new vectore with 1000 values between smin,smax linearly
f_x = @(x)1./(1+x).^3; %To obtain normalizing constant
c = 1./integral(f_x, 0 ,Inf); %Calculation of Normalizing Constant
pdf_real = c./((1+x_real).^3); %claculation of PDF real
plot(bincenters, pdf_est, x_real, pdf_real,'LineWidth',2); %plot graph of both real and estimated PDF for
comparision

%------g(x'|x) = N(x,0.1) and n = 10

n = 10; %assigning the value of n
s = zeros(n,1); %initializing sample array(MC) size to n
```

```
s(1) = 0; %setting the value of the sample to 0
new_samples = zeros(10000,1); %Initializing new markov chain to hold values of X(1)
for j = 1:length(new_samples) %Generating X(3) value 10000 times.
    for i = 2:n %run 3 times to obtain X(3)
        xprime = s(i-1) + randn*0.1; %Find x' value by adding normally distributed random value with SD = 0.1 to
previous sample
        alpha = f(xprime)/f(s(i-1)); %Find the alpha ratio, based on which the sample is either accepted or rejected
        if rand < alpha %if alpha>=1, the sample will be accepted. else will be rejected
            s(i) = xprime; %if true, then set the current value to x'
        else
            s(i) = s(i-1); %else set the current value to past value
        end
    end
    new_samples(j) = s(i); %Assign X(3) value obtained from above to MC
    s(1) = s(i); %Reinitialize the first element to the latest X(3) value
end

%disp(new_samples);

%Plotting estimate PDF for sample s and comparing it with real PDF
del = 0.1; %delta value, a small range for displacement
smin = min(new_samples); %minimum value from the baseline samples
smax = max(new_samples); %maximum value from the baseline samples
bincenters = smin:del:smax; %a vector containing all the values from smin to smax with an interval of del
pdf_est = zeros(1,length(bincenters)); %initializing the vector pdf_est to the same size as that of bincenters
for i=1:length(bincenters) %each value of bincenters must be compared
 pdf_est(i) = nnz(new_samples>bincenters(i)-del/2 & new_samples<=bincenters(i)+del/2)/length(new_samples)/del;
%calculating the number of nonzeros between bincenters(i)-del/2 bincenters(i)+del/2
end %Probability_est = count/nsamples. PDF est = probability_est/del
x_real = linspace(smin,smax,1000); %create a new vectore with 1000 values between smin,smax linearly
f_x = @(x)1./(1+x).^3; %To obtain normalizing constant
c = 1./integral(f_x, 0 ,Inf); %Calculation of Normalizing Constant
pdf_real = c./((1+x_real).^3); %claculation of PDF real
plot(bincenters, pdf_est, x_real, pdf_real,'LineWidth',2); %plot graph of both real and estimated PDF for
comparision

%------g(x'|x) = N(x,0.1) and n = 30

n = 30; %assigning the value of n
s = zeros(n,1); %initializing sample array(MC) size to n
s(1) = 0; %setting the value of the sample to 0
new_samples = zeros(10000,1); %Initializing new markov chain to hold values of X(1)
for j = 1:length(new_samples) %Generating X(3) value 10000 times.
    for i = 2:n %run 3 times to obtain X(3)
        xprime = s(i-1) + randn*0.1; %Find x' value by adding normally distributed random value with SD = 0.1 to
previous sample
        alpha = f(xprime)/f(s(i-1)); %Find the alpha ratio, based on which the sample is either accepted or rejected
        if rand < alpha %if alpha>=1, the sample will be accepted. else will be rejected
            s(i) = xprime; %if true, then set the current value to x'
        else
            s(i) = s(i-1); %else set the current value to past value
        end
    end
    new_samples(j) = s(i); %Assign X(3) value obtained from above to MC
    s(1) = s(i); %Reinitialize the first element to the latest X(3) value
end
```

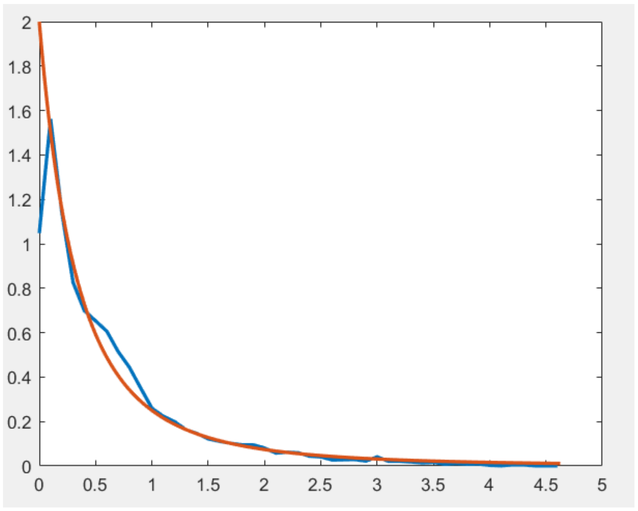
```
%disp(new_samples);

%Plotting estimate PDF for sample s and comparing it with real PDF
del = 0.1; %delta value, a small range for displacement
smin = min(new_samples); %minimum value from the baseline samples
smax = max(new_samples); %maximum value from the baseline samples
bincenters = smin:del:smax; %a vector containing all the values from smin to smax with an interval of del
pdf_est = zeros(1,length(bincenters)); %initializing the vector pdf_est to the same size as that of bincenters
for i=1:length(bincenters) %each value of bincenters must be compared
 pdf_est(i) = nnz(new_samples>bincenters(i)-del/2 & new_samples<=bincenters(i)+del/2)/length(new_samples)/del;
%calculating the number of nonzeros between bincenters(i)-del/2 bincenters(i)+del/2
end %Probability_est = count/nsamples. PDF est = probability_est/del
x_real = linspace(smin,smax,1000); %create a new vectore with 1000 values between smin,smax linearly
f_x = @(x)1./(1+x).^3; %To obtain normalizing constant
c = 1./integral(f_x, 0 ,Inf); %Calculation of Normalizing Constant
pdf_real = c./((1+x_real).^3); %claculation of PDF real
plot(bincenters, pdf_est, x_real, pdf_real,'LineWidth',2); %plot graph of both real and estimated PDF for
comparision


%------g(x'|x) = N(x,0.1) and n = 100

n = 100; %assigning the value of n
s = zeros(n,1); %initializing sample array(MC) size to n
s(1) = 0; %setting the value of the sample to 0
new_samples = zeros(10000,1); %Initializing new markov chain to hold values of X(1)
for j = 1:length(new_samples) %Generating X(3) value 10000 times.
    for i = 2:n %run 3 times to obtain X(3)
        xprime = s(i-1) + randn*0.1; %Find x' value by adding normally distributed random value with SD = 0.1 to
previous sample
        alpha = f(xprime)/f(s(i-1)); %Find the alpha ratio, based on which the sample is either accepted or rejected
        if rand < alpha %if alpha>=1, the sample will be accepted. else will be rejected
            s(i) = xprime; %if true, then set the current value to x'
        else
            s(i) = s(i-1); %else set the current value to past value
        end
    end
    new_samples(j) = s(i); %Assign X(3) value obtained from above to MC
    s(1) = s(i); %Reinitialize the first element to the latest X(3) value
end

%Plotting estimate PDF for sample s and comparing it with real PDF
del = 0.1; %delta value, a small range for displacement
smin = min(new_samples); %minimum value from the baseline samples
smax = max(new_samples); %maximum value from the baseline samples
bincenters = smin:del:smax; %a vector containing all the values from smin to smax with an interval of del
pdf_est = zeros(1,length(bincenters)); %initializing the vector pdf_est to the same size as that of bincenters
for i=1:length(bincenters) %each value of bincenters must be compared
 pdf_est(i) = nnz(new_samples>bincenters(i)-del/2 & new_samples<=bincenters(i)+del/2)/length(new_samples)/del;
%calculating the number of nonzeros between bincenters(i)-del/2 bincenters(i)+del/2
end %Probability_est = count/nsamples. PDF est = probability_est/del
x_real = linspace(smin,smax,1000); %create a new vector with 1000 values between smin,smax linearly
f_x = @(x)1./(1+x).^3; %To obtain normalizing constant
c = 1./integral(f_x, 0 ,Inf); %Calculation of Normalizing Constant
pdf_real = c./((1+x_real).^3); %calculation of PDF real
```

```
plot(bincenters, pdf_est, x_real, pdf_real,'LineWidth',2); %plot graph of both real and estimated PDF for comparison
```

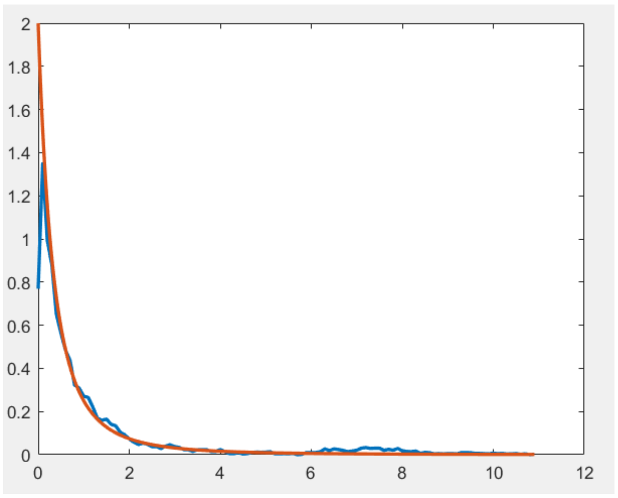
**Output for $g(x'|x) = N(x, 0.1)$ and $n = 1$:**



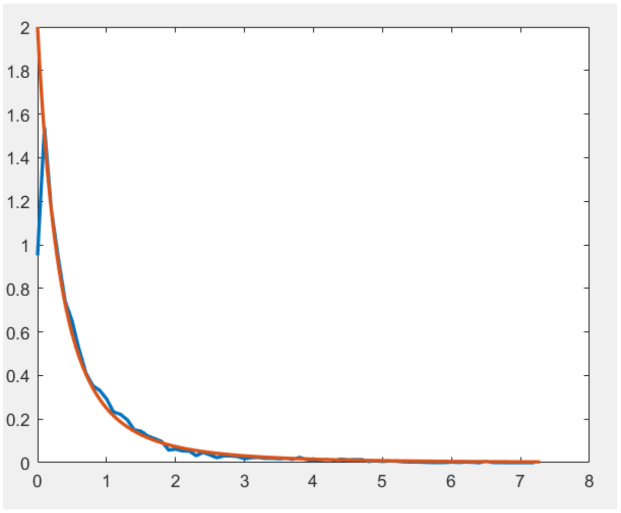
**Output for $g(x'|x) = N(x, 0.1)$ and $n = 3$:**



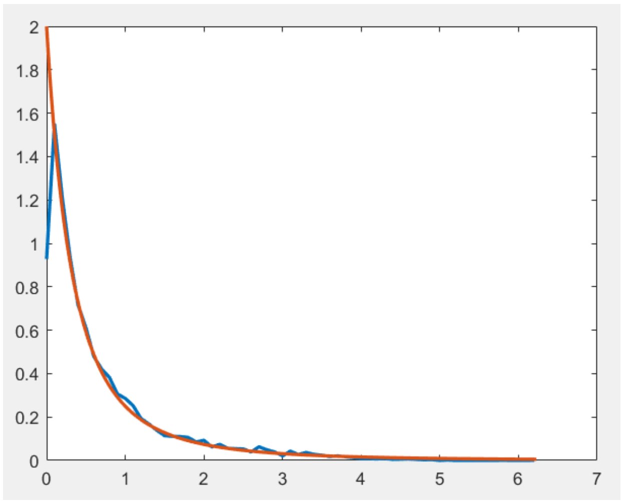
**Output for $g(x'|x) = N(x, 0.1)$ and $n = 5$:**



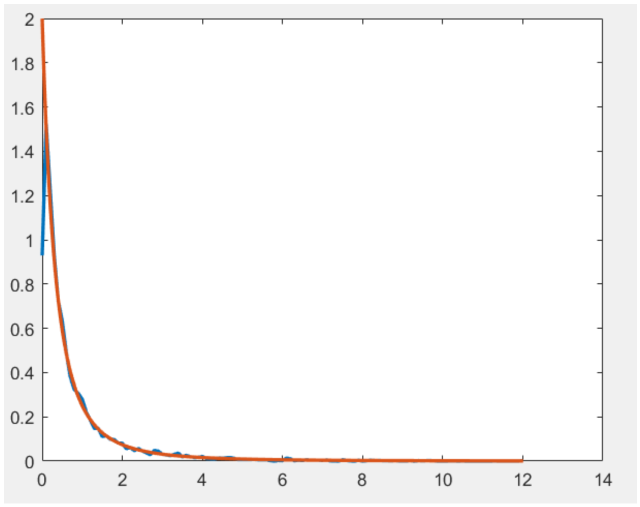
**Output for $g(x'|x) = N(x, 0.1)$ *and* $n = 10$:**



**Output for $g(x'|x) = N(x, 0.1)$ *and* $n = 30$:**



**Output for $g(x'|x) = N(x, 0.1)$ *and* $n = 100$:**



**Code for $g(x'|x) = U(x+0.5, x-0.5)$ *and* $n = 1, 3, 5, 10, 30, 100$:**

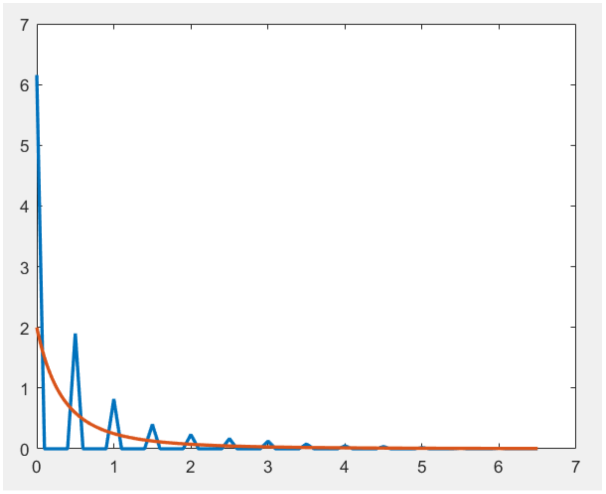
```
%TASK 3

f = @(x) 1/(1+x)^3*(x>=0); %Burr function for all x greater than or equal to 0
rng(0); %setting seed for rand generation

%------g(x'|x) = U(x-0.5, x+0.5) and n = 1

n = 1; %assigning the value of n
s = zeros(n,1); %initializing sample array(MC) size to n
s(1) = 0; %setting the value of the sample to 0
new_samples = zeros(10000,1); %Initializing new markov chain to hold values of X(1)

for j = 1:length(new_samples) %Generating X(1) value 10000 times.
    xprime = s(1) + (rand<0.5)-1/2; %Find x' value by adding or subtracting 0.5 from current sample
    alpha = f(xprime)/f(s(1)); %Find the alpha ratio, based on which the sample is either accepted or rejected
    if rand < alpha %if alpha>=1, the sample will be accepted. else will be rejected
        s(1) = xprime; %if true, then set the current value to x' (aslo re-initializes first element to new value)
    else
        s(1) = s(1); %else set the current value to past value
    end
    new_samples(j) = s(1); %Append the new value to new_samples(MC)
end %Generate X(1) 10000 times

%Plotting estimate PDF for sample s and comparing it with real PDF
del = 0.1; %delta value, a small range for displacement
smin = min(new_samples); %minimum value from the new_samples
smax = max(new_samples); %maximum value from the new_samples
bincenters = smin:del:smax; %a vector containing all the values from smin to smax with an interval of del
pdf_est = zeros(1,length(bincenters)); %initializing the vector pdf_est to the same size as that of bincenters
for i=1:length(bincenters) %each value of bincenters must be compared
 pdf_est(i) = nnz(new_samples>bincenters(i)-del/2 & new_samples<=bincenters(i)+del/2)/length(new_samples)/del;
%calculating the number of nonzeros between bincenters(i)-del/2 bincenters(i)+del/2
end %Probability_est = count/nsamples. PDF est = probability_est/del
x_real = linspace(smin,smax,1000); %create a new vector with 1000 values between smin,smax linearly
f_x = @(x)1./(1+x).^3; %To obtain normalizing constant
c = 1./integral(f_x, 0 ,Inf); %Calculation of Normalizing Constant;
pdf_real = c./((1+x_real).^3); %calculation of PDF real
```

```
plot(bincenters, pdf_est, x_real, pdf_real,'LineWidth',2); %plot graph of both real and estimated PDF for comparison


%------g(x'|x) = U(x-0.5, x+0.5) and n = 3

n = 3; %assigning the value of n
s = zeros(n,1); %initializing sample array(MC) size to n
s(1) = 0; %setting the value of the sample to 0
new_samples = zeros(10000,1); %Initializing new markov chain to hold values of X(1)
for j = 1:length(new_samples) %Generating X(3) value 10000 times.
    for i = 2:n %run 3 times to obtain X(3)
        xprime = s(i-1) + (rand<0.5)-1/2; %Find x' value by adding or subtracting 0.5 from current sample
        alpha = f(xprime)/f(s(i-1)); %Find the alpha ratio, based on which the sample is either accepted or rejected
        if rand < alpha %if alpha>=1, the sample will be accepted. else will be rejected
            s(i) = xprime; %if true, then set the current value to x'
        else
            s(i) = s(i-1); %else set the current value to past value
        end
    end
    new_samples(j) = s(i); %Assign X(3) value obtained from above to MC
    s(1) = s(i); %Reinitialize the first element to the latest X(3) value
end

%Plotting estimate PDF for sample s and comparing it with real PDF
del = 0.1; %delta value, a small range for displacement
smin = min(new_samples); %minimum value from the baseline samples
smax = max(new_samples); %maximum value from the baseline samples
bincenters = smin:del:smax; %a vector containing all the values from smin to smax with an interval of del
pdf_est = zeros(1,length(bincenters)); %initializing the vector pdf_est to the same size as that of bincenters
for i=1:length(bincenters) %each value of bincenters must be compared
 pdf_est(i) = nnz(new_samples>bincenters(i)-del/2 & new_samples<=bincenters(i)+del/2)/length(new_samples)/del;
%calculating the number of nonzeros between bincenters(i)-del/2 bincenters(i)+del/2
end %Probability_est = count/nsamples. PDF est = probability_est/del
x_real = linspace(smin,smax,1000); %create a new vector with 1000 values between smin,smax linearly
f_x = @(x)1./(1+x).^3; %To obtain normalizing constant
c = 1./integral(f_x, 0 ,Inf); %Calculation of Normalizing Constant
pdf_real = c./((1+x_real).^3); %calculation of PDF real
plot(bincenters, pdf_est, x_real, pdf_real,'LineWidth',2); %plot graph of both real and estimated PDF for comparison


%------g(x'|x) = U(x-0.5, x+0.5) and n = 5

n = 5; %assigning the value of n
s = zeros(n,1); %initializing sample array(MC) size to n
s(1) = 0; %setting the value of the sample to 0
new_samples = zeros(10000,1); %Initializing new markov chain to hold values of X(1)
for j = 1:length(new_samples) %Generating X(3) value 10000 times.
    for i = 2:n %run 3 times to obtain X(3)
        xprime = s(i-1) + (rand<0.5)-1/2; %Find x' value by adding or subtracting 0.5 from current sample
        alpha = f(xprime)/f(s(i-1)); %Find the alpha ratio, based on which the sample is either accepted or rejected
        if rand < alpha %if alpha>=1, the sample will be accepted. else will be rejected
            s(i) = xprime; %if true, then set the current value to x'
        else
            s(i) = s(i-1); %else set the current value to past value
        end
    end
    new_samples(j) = s(i); %Assign X(3) value obtained from above to MC
    s(1) = s(i); %Reinitialize the first element to the latest X(3) value
end

%Plotting estimate PDF for sample s and comparing it with real PDF
del = 0.1; %delta value, a small range for displacement
smin = min(new_samples); %minimum value from the baseline samples
smax = max(new_samples); %maximum value from the baseline samples
bincenters = smin:del:smax; %a vector containing all the values from smin to smax with an interval of del
pdf_est = zeros(1,length(bincenters)); %initializing the vector pdf_est to the same size as that of bincenters
for i=1:length(bincenters) %each value of bincenters must be compared
 pdf_est(i) = nnz(new_samples>bincenters(i)-del/2 & new_samples<=bincenters(i)+del/2)/length(new_samples)/del;
%calculating the number of nonzeros between bincenters(i)-del/2 bincenters(i)+del/2
end %Probability_est = count/nsamples. PDF est = probability_est/del
x_real = linspace(smin,smax,1000); %create a new vector with 1000 values between smin,smax linearly
f_x = @(x)1./(1+x).^3; %To obtain normalizing constant
c = 1./integral(f_x, 0 ,Inf); %Calculation of Normalizing Constant
pdf_real = c./((1+x_real).^3); %calculation of PDF real
plot(bincenters, pdf_est, x_real, pdf_real,'LineWidth',2); %plot graph of both real and estimated PDF for comparison

%------g(x'|x) = U(x-0.5, x+0.5) and n = 10

n = 10; %assigning the value of n
s = zeros(n,1); %initializing sample array(MC) size to n
s(1) = 0; %setting the value of the sample to 0
new_samples = zeros(10000,1); %Initializing new markov chain to hold values of X(1)
for j = 1:length(new_samples) %Generating X(3) value 10000 times.
    for i = 2:n %run 3 times to obtain X(3)
        xprime = s(i-1) + (rand<0.5)-1/2; %Find x' value by adding or subtracting 0.5 from current sample
        alpha = f(xprime)/f(s(i-1)); %Find the alpha ratio, based on which the sample is either accepted or rejected
        if rand < alpha %if alpha>=1, the sample will be accepted. else will be rejected
            s(i) = xprime; %if true, then set the current value to x'
        else
            s(i) = s(i-1); %else set the current value to past value
        end
    end
    new_samples(j) = s(i); %Assign X(3) value obtained from above to MC
    s(1) = s(i); %Reinitialize the first element to the latest X(3) value
end

%Plotting estimate PDF for sample s and comparing it with real PDF
del = 0.1; %delta value, a small range for displacement
smin = min(new_samples); %minimum value from the baseline samples
smax = max(new_samples); %maximum value from the baseline samples
bincenters = smin:del:smax; %a vector containing all the values from smin to smax with an interval of del
pdf_est = zeros(1,length(bincenters)); %initializing the vector pdf_est to the same size as that of bincenters
for i=1:length(bincenters) %each value of bincenters must be compared
 pdf_est(i) = nnz(new_samples>bincenters(i)-del/2 & new_samples<=bincenters(i)+del/2)/length(new_samples)/del;
%calculating the number of nonzeros between bincenters(i)-del/2 bincenters(i)+del/2
end %Probability_est = count/nsamples. PDF est = probability_est/del
x_real = linspace(smin,smax,1000); %create a new vector with 1000 values between smin,smax linearly
f_x = @(x)1./(1+x).^3; %To obtain normalizing constant
c = 1./integral(f_x, 0 ,Inf); %Calculation of Normalizing Constant
pdf_real = c./((1+x_real).^3); %calculation of PDF real
plot(bincenters, pdf_est, x_real, pdf_real,'LineWidth',2); %plot graph of both real and estimated PDF for comparison
```

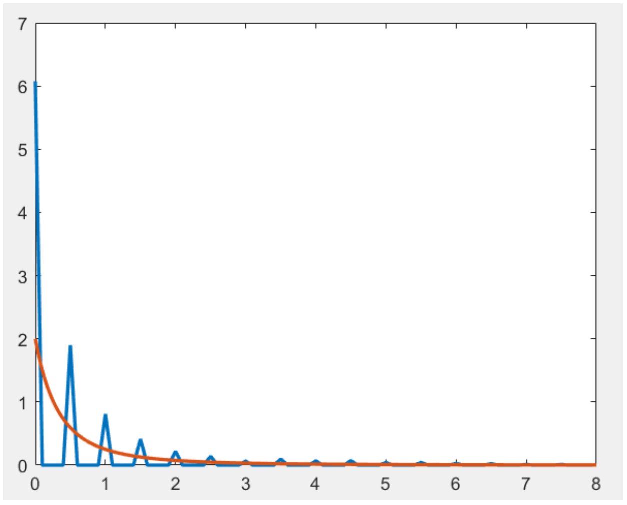
```
%------g(x'|x) = U(x-0.5, x+0.5) and n = 30

n = 30; %assigning the value of n
s = zeros(n,1); %initializing sample array(MC) size to n
s(1) = 0; %setting the value of the sample to 0
new_samples = zeros(10000,1); %Initializing new markov chain to hold values of X(1)
for j = 1:length(new_samples) %Generating X(3) value 10000 times.
    for i = 2:n %run 3 times to obtain X(3)
        xprime = s(i-1) + (rand<0.5)-1/2; %Find x' value by adding or subtracting 0.5 from current sample
        alpha = f(xprime)/f(s(i-1)); %Find the alpha ratio, based on which the sample is either accepted or rejected
        if rand < alpha %if alpha>=1, the sample will be accepted. else will be rejected
            s(i) = xprime; %if true, then set the current value to x'
        else
            s(i) = s(i-1); %else set the current value to past value
        end
    end
    new_samples(j) = s(i); %Assign X(3) value obtained from above to MC
    s(1) = s(i); %Reinitialize the first element to the latest X(3) value
end

%Plotting estimate PDF for sample s and comparing it with real PDF
del = 0.1; %delta value, a small range for displacement
smin = min(new_samples); %minimum value from the baseline samples
smax = max(new_samples); %maximum value from the baseline samples
bincenters = smin:del:smax; %a vector containing all the values from smin to smax with an interval of del
pdf_est = zeros(1,length(bincenters)); %initializing the vector pdf_est to the same size as that of bincenters
for i=1:length(bincenters) %each value of bincenters must be compared
 pdf_est(i) = nnz(new_samples>bincenters(i)-del/2 & new_samples<=bincenters(i)+del/2)/length(new_samples)/del;
%calculating the number of nonzeros between bincenters(i)-del/2 bincenters(i)+del/2
end %Probability_est = count/nsamples. PDF est = probability_est/del
x_real = linspace(smin,smax,1000); %create a new vector with 1000 values between smin,smax linearly
f_x = @(x)1./(1+x).^3; %To obtain normalizing constant
c = 1./integral(f_x, 0 ,Inf); %Calculation of Normalizing Constant
pdf_real = c./((1+x_real).^3); %calculation of PDF real
plot(bincenters, pdf_est, x_real, pdf_real,'LineWidth',2); %plot graph of both real and estimated PDF for comparison


%------g(x'|x) = U(x-0.5, x+0.5) and n = 100

n = 100; %assigning the value of n
s = zeros(n,1); %initializing sample array(MC) size to n
s(1) = 0; %setting the value of the sample to 0
new_samples = zeros(10000,1); %Initializing new markov chain to hold values of X(1)
for j = 1:length(new_samples) %Generating X(3) value 10000 times.
    for i = 2:n %run 3 times to obtain X(3)
        xprime = s(i-1) + (rand<0.5)-1/2; %Find x' value by adding or subtracting 0.5 from current sample
        alpha = f(xprime)/f(s(i-1)); %Find the alpha ratio, based on which the sample is either accepted or rejected
        if rand < alpha %if alpha>=1, the sample will be accepted. else will be rejected
            s(i) = xprime; %if true, then set the current value to x'
        else
            s(i) = s(i-1); %else set the current value to past value
        end
    end
    new_samples(j) = s(i); %Assign X(3) value obtained from above to MC
    s(1) = s(i); %Reinitialize the first element to the latest X(3) value
end
```

```
%Plotting estimate PDF for sample s and comparing it with real PDF
del = 0.1; %delta value, a small range for displacement
smin = min(new_samples); %minimum value from the baseline samples
smax = max(new_samples); %maximum value from the baseline samples
bincenters = smin:del:smax; %a vector containing all the values from smin to smax with an interval of del
pdf_est = zeros(1,length(bincenters)); %initializing the vector pdf_est to the same size as that of bincenters
for i=1:length(bincenters) %each value of bincenters must be compared
 pdf_est(i) = nnz(new_samples>bincenters(i)-del/2 & new_samples<=bincenters(i)+del/2)/length(new_samples)/del;
%calculating the number of nonzeros between bincenters(i)-del/2 bincenters(i)+del/2
end %Probability_est = count/nsamples. PDF est = probability_est/del
x_real = linspace(smin,smax,1000); %create a new vector with 1000 values between smin,smax linearly
f_x = @(x)1./(1+x).^3; %To obtain normalizing constant
c = 1./integral(f_x, 0 ,Inf); %Calculation of Normalizing Constant
pdf_real = c./((1+x_real).^3); %calculation of PDF real
plot(bincenters, pdf_est, x_real, pdf_real,'LineWidth',2); %plot graph of both real and estimated PDF for comparison
```

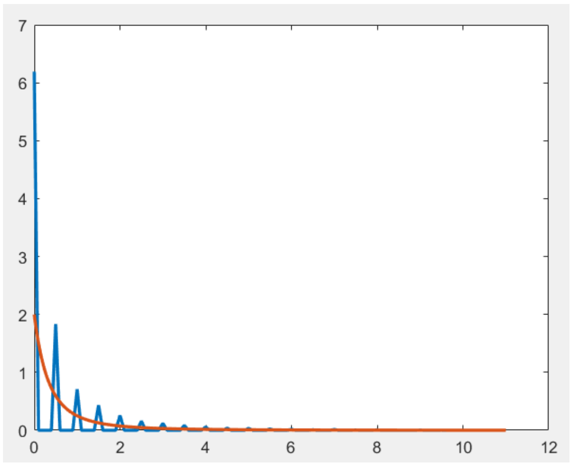
**Output for $g(x'|x) = U(x+0.5, x-0.5)$ *and* $n = 1$**



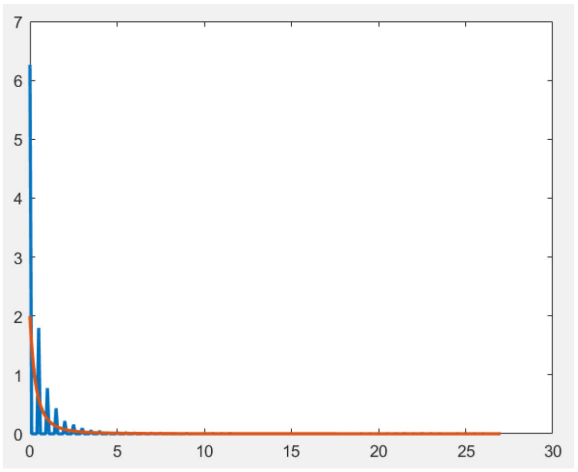
**Output for $g(x'|x) = U(x+0.5, x-0.5)$ *and* $n = 3$**



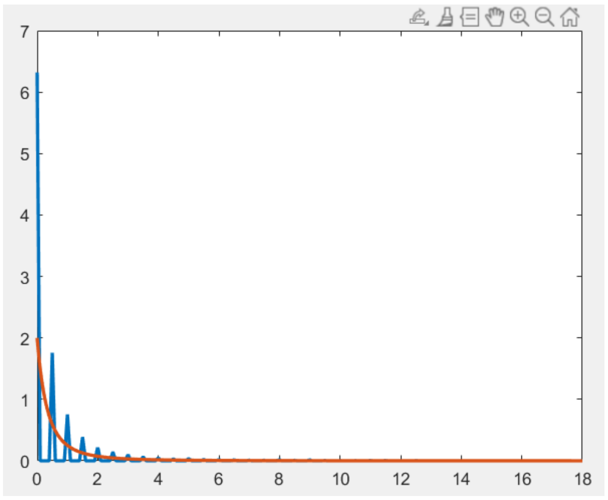
**Output for** $g(x'|x) = U(x + 0.5, x - 0.5)$ ***and*** $n = 5$



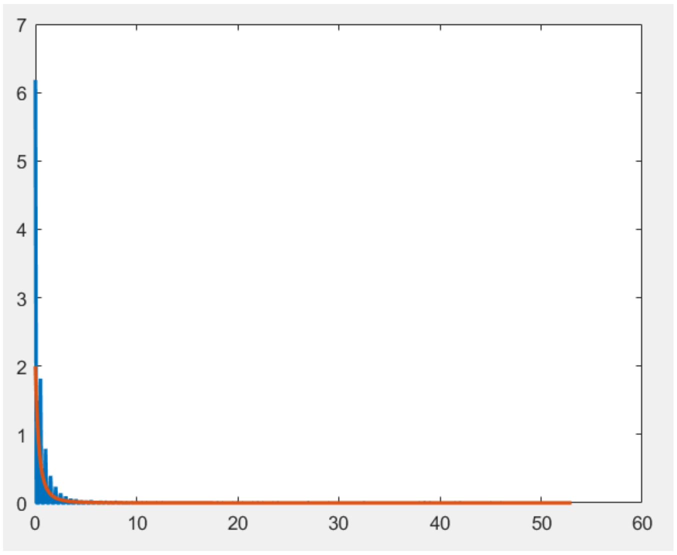
**Output for** $g(x'|x) = U(x + 0.5, x - 0.5)$ ***and*** $n = 10$



**Output for** $g(x'|x) = U(x + 0.5, x - 0.5)$ ***and*** $n = 30$



**Output for $g(x'|x) = U(x+0.5, x-0.5)$ and $n = 100$**



I am not sure if I am implementing U(x-0.5, x+0.5) correctly, as my MC is not mixing.

## Task 4: Reading and displaying an Image

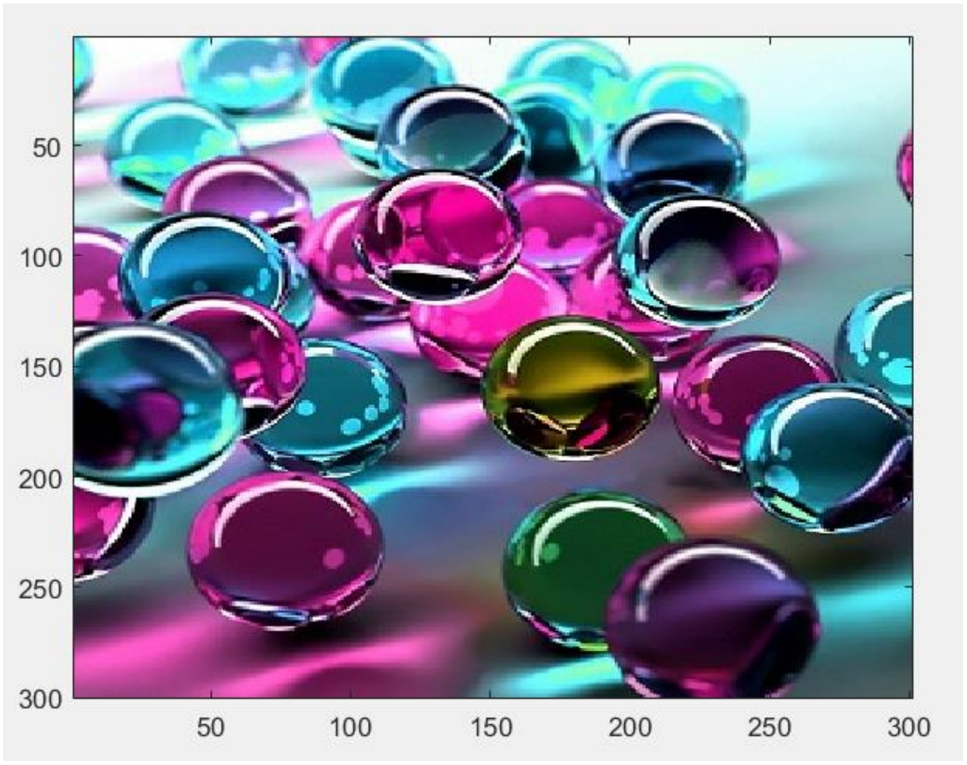
I have chosen an image with size 300x301. Details about the command is written in the comments of the code.

**Code:**

```
%TASK 4

%Reading an image

A = imread('Colourful_copy.jpg'); %imread converts an image to mxnx3 of uint8 dtype matrix and stores in A
A = double(A); %converts all the elements from uint8 to double as it is easier to work with
A = uint8(A); %converts all the elements from double to uint8
imwrite(A,'write_try.png','png'); %writes the array A into a file
image(A); %is a MATLAB command that visualizes matrix as matrix of numbers. gives a rough output
imshow(A); %is an image processing tool command. gives a finer output
```

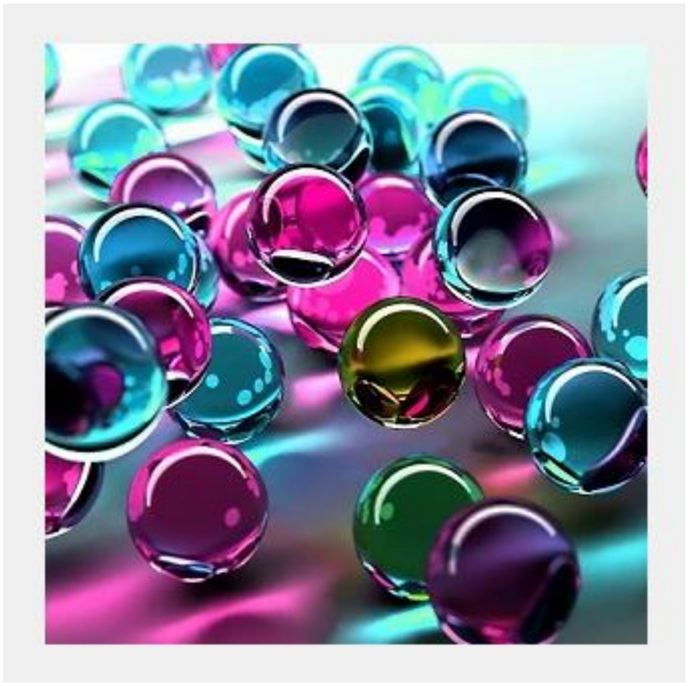
**Output of the command image(A):**

As the command image(A) treats the matrix A as matrix itself, the output is displayed as a plot. Hence, there is x-axis and y-axis with values ranging from 0-300 in x-axis, 0-301 in y-axis.



**Output of the command imshow(A):**

As the command imshow is a command from image processing toolbox, the matrix A will be displayed as an image rather than a plot. Therefore, it looks more clear.



## Task 5: Mosaic image creation without a reference Image

Pseudocode followed for the implementation of Task 5:

```
P ←U(0,255)
Z ←random integers from 1 to 4

for n = 1 to N do

    Change Z with a probability of 0.99

        select an element from Z matrix randomly

        $Z' \sim g(x'|x)$

    Change P with a probability of 0.01

        select an element from P matrix randomly

        $P' \sim g(p'|p)$

    If Z is changed

        $\alpha = \frac{P_X(x')}{P_X(x)} = \frac{f_P(p)}{f_P(p)} \frac{P_Z(z')}{P_Z(z)} = \frac{P_Z(z')}{P_Z(z)}$

        If rand<alpha

            Accept the sample

        Else

            Reject the sample

    If P is changed

        $\alpha = \frac{P_X(x')}{P_X(x)} = \frac{f_P(p)}{f_P(p)} \frac{P_Z(z)}{P_Z(z)} = 1$

        If rand<alpha

            Accept the sample

        Else

            Reject the sample
```

**In MATLAB:** These are the steps I followed to implement the code for Task 5:

1) Initialized P matrix with uniform random numbers from 0 to 255. P is a continuous random variable as it can take any value between 0 and 255. Initialize Z matrix with integers between 1 and 4. I am running my code 30000000 times as it seems to give me the best results.

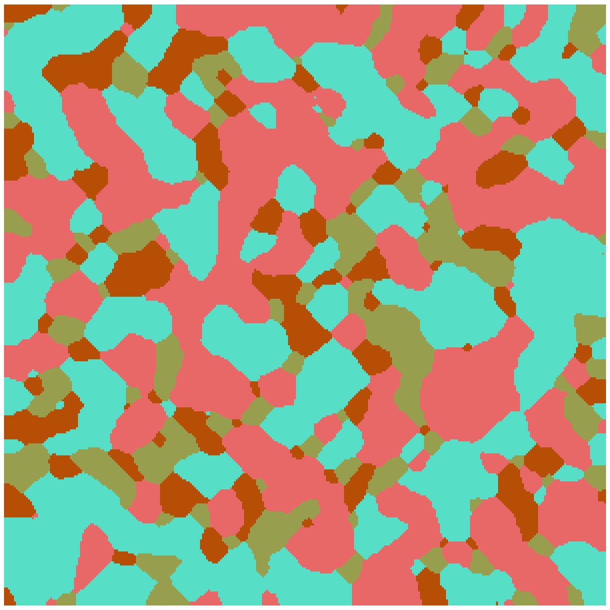
2) A random variable that needs to be changed is picked randomly from either P or Z. The decision to choose weather Z needs to be changed or P needs to be changed is arrived at by generating a random number and checking if it is less than 0.99. This means that 99% of the times Z needs to be changed and remaining 0.1% of the times, P will be changed. This is because there are 300*300 values in Z and P has 4*3 values. As there are 90000 values in Z and 12 values in P, maximum number of the times, Z must be changed to obtain a mosaic efficiently.
3) Here, we will not maintain a vector of random variables. But instead, we will use only the current value to obtain a new sample. For Z, new sample is generated using the round value of $g(x'|x) = N(x,2)$. I have tried using uniform distribution with steps up or down by 1 randomly. But it takes longer time to run and did not give a very good mosaic. For P, new sample is generated using the value $g(x'|x) = N(x,2)$.
4) The ratio alpha for Z is calculated by using the function P. It is the count of number of times the neighboring cells are same as our value multiplied by a constant r. If the count for Z' is greater, that means we need to accept Z'. Else the sample will be rejected. For P, as it is a uniform distribution, the value $\frac{f(P')}{f(P)} = 1$ for all P. Hence P will always be accepted.
5) Once the sample generation is done, I combined the P and X matrix to be able to obtain an image using a simple for loop.
6) I chose 10 for my r value as it worked the best.

**Code:**

```
%TASK 5
%Creating mosaic image

rng(0); %setting seed for rand generation
X = zeros(300,301,3); %Initializing the image matrix to 0s
P =255*rand(4,3); %Initializing the colour pallete matrix with values from 0 to 255
Z =randi([1 4],300,301); %Initializing colour index matrix with values from 1 to 4
nSamples = 30000000; % The number of times the algorithm must run to produce a good image

for t = 2:nSamples %Initialize for loop to run the algorithm 30000000 times
   currentZ=0; %holds current value of Z that is obtained by randomly choosing a position in the Z matrix
   currentP=0; %%holds current value of P that is obtained by randomly choosing a position in the P matrix
   currentZ_flag=0; %A flag variable to indicate that Z' is generated
   currentP_flag=0; %A flag variable to indicate that P' is generated
   zPrime =0; %Holds Z' value
   pPrime =0; %Holds P' value
   if rand < 0.99 %Decides if the random variable Z or P must be changed
      i = randi(300); %generate a random number between 0-300 to select a row randomly from Z
      j = randi(301); %generate a random number between 0-301 to select a row randomly from Z
      currentZ= Z(i,j); %Initialize the current value with the coordinates randomly generated above
      zPrime = currentZ + round(randn*2); %g(x'|x)=N(x,2). Rounded as Z is discrete
      currentZ_flag = 1; %set flag to indicate Z' is generated
   else
      k = randi(4); %Generate a random number between 1-4 to choose colour
      l = randi(3); %Generate a random number between 1-3 to choose R/G/B
      currentP = P(k,l); %Initialize the current value with the coordinates randomly generated
```

```
        pPrime = currentP + randn*2; %g(x'|x)=N(x,2). Rounded ad Z is discrete
        currentP_flag = 1; %set flag to indicate P' is generated
    end

    if  currentZ_flag == 1 && zPrime >0 && zPrime <5 %check if z' is generated and if it is valid
        alpha = pmfZ(zPrime,i,j,Z)/pmfZ(currentZ,i,j,Z); %The ratio alpha decides is the new sample is selected or not
        if rand < alpha  %Higher the alpha, greater the chance that the sample is picked
            Z(i,j) = zPrime; %Assigning the new sample to the current value of Z
        else
            Z(i,j) = currentZ; %Else reject the sample by making no change to the current value
        end
    end
    if currentP_flag == 1 && pPrime >0 && pPrime <256 %check if P' is generated and if it is valid
        alpha = 1; %This ratio will always be 1 as it is a uniform distribution between 0 and 255. f'(x)/f(x)=1
        if rand < alpha %Higher the alpha, greater the chance that the sample is picked
            P(k,l) = pPrime; %Assigning the new sample to the current value of P
        else
            P(k,l) = currentP; %Else reject the sample by making no change to the current value
        end
    end
end

%combine P and Z to produce X = (P,Z) where X(i,j,k) =
%(P(l,1),P(l,2),P(l,3) where l = Z(i,j)
for i = 1:300
    for j = 1:301
        for k = 1:3
            X(i,j,k) = P(Z(i,j),k);
        end
    end
end

X = uint8(X);

image(X)
%imshow(X)
imwrite(X,'mosaic.png','png');

%For the alpha value,the distribution pZ(z) is used.
%The delta function checks how many of the neighbours have same value
%If the value is same then the count is increased by one. Else the count
%will remain the same.
function p = pmfZ(z,i,j,Z)
    total =0;
    for a = (i-1):(i+1) %to access the immediate neighbours of current Z
        for b =(j-1):(j+1)
            if ~(a == i && b== j) && a <301 && b < 302 && a > 0 && b > 0 %To check if the address isnt out of
bounds
                temp = Z(a,b);
                if (z - temp) == 0 %counting the number for similar neighbours
                    total = total + 1;
                end
            end
        end
    end
    r = 10; %is a constant value that suits best for the code
    p = exp(r*total); %calculating P'(Z) and P(Z).
end
```

**Output:**



## Task 6: Mosaic image creation with a reference Image

The goal is to make a mosaic image resemble the original image. It is implemented the same way as task 5 except for a few changes. The reference image is incorporated into the algorithm while calculating the deciding ratio alpha.

**Pseudocode** followed for the implementation of Task 6:

```
P ←U(0,255)
Z ←random integers from 1 to 256(number of colors)

for n = 1 to N do

    Change Z with a probability of 0.99

        select an element from Z matrix randomly

        $Z'{\sim}g(x'|x)$

    Change P with a probability of 0.01

        select an element from P matrix randomly

        $P'{\sim}g(p'|p)$

    If Z is changed
        $\alpha = \frac{P_X(x')}{P_X(x)} = \frac{f_P(p)}{f_P(p)} \frac{P_Z(z')}{P_Z(z)} \frac{P_{PZ}(p',z')}{P_{PZ}(p,z)} = \frac{P_Z(z')}{P_Z(z)} \frac{P_{PZ}(p',z')}{P_{PZ}(p,z)}$
        If rand<alpha
            Accept the sample
        Else
            Reject the sample
    If P is changed
        $\alpha = \frac{P_X(x')}{P_X(x)} = \frac{f_P(p)}{f_P(p)} \frac{P_Z(z')}{P_Z(z)} \frac{P_{PZ}(p',z')}{P_{PZ}(p,z)} = \frac{P_{PZ}(p',z')}{P_{PZ}(p,z)}$
        If rand<alpha
            Accept the sample
        Else
            Reject the sample
```

**In MATLAB:** The steps followed to implement Task 6 are like Task 5

1) Initialize the P and Z matrix just as in task 5, except, I am choosing 256 as my number of colors instead of 4. I am running my algorithm 20 million times as it produces a pretty good result. And running it longer makes no difference.
2) The random variables P or Z is chosen randomly with probability of choosing Z higher than probability of choosing P. Just as in task 5. The function to generate new sample for P and Z are same as it is in Task 5.
3) As seen in pseudo code, the alpha value depends on both $P_Z(z)$ *and* $P_{P,Z}(P,Z)$. $\frac{P_Z(z')}{P_Z(z)}$ is generated using the function P like in task 5. For $\frac{P_{PZ}(p',z')}{P_{PZ}(p,z)}$, the value is calculated using the function q. The alpha value for P is obtained from function u.
4) Functions q and u produce alpha value by summing all the differences between the neighboring values and the sample. This sum is multiplied by a constant s.
5) I have chosen my r value to be 10. And my s value as -1.

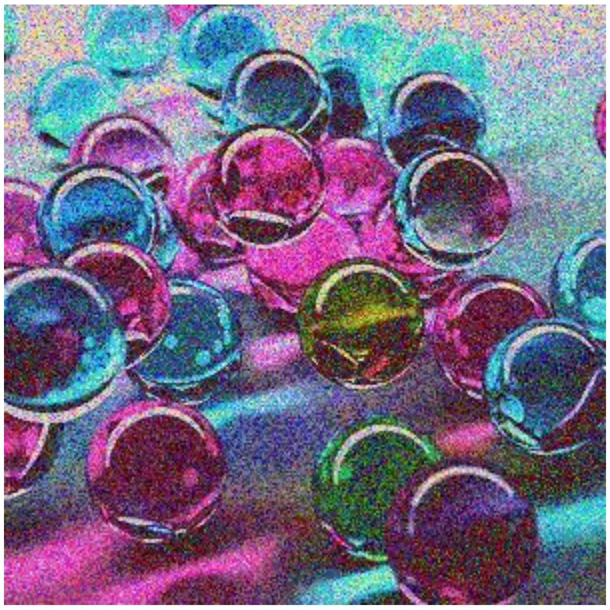
**Code:**

```
%Task 6
%Making mosaic look like the reference image

rng(0); %setting seed for rand generation
A=imread('Colourful_copy.jpg'); %imread converts an image to mxnx3 of uint8 dtype matrix and stores in A
A=double(A); %converts all the elements from uint8 to double as it is easier to work with

X = zeros(300,301,3); %Initializing the image matrix to 0s
```

```
P =255*rand(256,3); %Initializing the colour pallete matrix with values from 0 to 255
Z =randi([1 256],300,301); %Initializing colour index matrix with values from 1 to 4
nSamples = 20000000; % The number of times the algorithm must run to produce a good image

for t = 2:nSamples %Initialize for loop to run the algorithm 30000000 times
    currentZ=0; %holds current value of Z that is obtained by randomly choosing a position in the Z matrix
    currentP=0; %holds current value of Z that is obtained by randomly choosing a position in the P matrix
    zPrime =0; %Initialize Z' to a random starting point. I have chosen 0
    pPrime =0; %Initialize P' to a random starting point. I have chosen 0
    ZF=0; %A flag variable to indicate that Z' is generated
    PF=0; %A flag variable to indicate that P' is generated
    i=0;
    j=0;
    k=0;
    l=0;
    if rand < 0.99 %Decides if the random variable Z or P must be changed
        i = randi(300); %generate a random number between 0-300 to select a row randomly from Z
        j = randi(301); %generate a random number between 0-301 to select a row randomly from Z
        currentZ= Z(i,j); %Initialize the current value with the coordinates randomly generated above
        zPrime = currentZ + round(randn*2); %g(x'|x)=N(x,2). Rounded as Z is discrete
        ZF = 1; %set flag to indicate Z' is generated
    else
        k = randi(256); %Generate a random number between 1-256, to choose colour
        l = randi(3); %Generate a random number between 1-3 to choose R/G/B
        currentP= P(k,l); %Initialize the current value with the coordinates randomply generated
        pPrime = currentP + randn*2; %g(x'|x)=N(x,2). Rounded ad Z is discrete
        PF=1; %set flag to indicate P' is generated
    end

    if  ZF == 1 && zPrime > 0  && zPrime < 257 %check if z' is generated and if it is valid
        alpha1 = (pmfZ(zPrime,i,j,Z)*jPdfZ(i,j,zPrime,P,A))/(pmfZ(currentZ,i,j,Z)*jPdfZ(i,j,currentZ,P,A)); %The
ratio alpha decides if the new
                                                                          %sample is selected or not
        if rand < alpha1 %Higher the alpha, greater the chance that the sample is picked
            Z(i,j) = zPrime; %Assigning the new sample to the current value of Z
        else
            Z(i,j) = currentZ; %Else reject the sample by making no change to the current value
        end
    end

    if PF == 1 && pPrime > 0 && pPrime < 256 %check if P' is generated and if it is valid
        aplha2 = jPdfP(k,l,pPrime,Z,A)/jPdfP(k,l,currentP,Z,A); %As P is compared with reference image, f(P')/f(P) !=
1 for all P
        if rand < alpha2 %Higher the alpha, greater the chance that the sample is picked
            P(k,l) = pPrime; %Assigning the new sample to the current value of P
        else
            P(k,l) = currentP; %Else reject the sample by making no change to the current value
        end
    end

end

%combine P and Z to produce X = (P,Z) where X(i,j,k) =
%(P(l,1),P(l,2),P(l,3) where l = Z(i,j). To see how our mosaic
%looks after applying MH
for i = 1:300
```

```
    for j = 1:301
        for k = 1:3
            X(i,j,k) = P(Z(i,j),k);
        end
    end
end

X=uint8(X);
X= round(X);
imshow(X);

imwrite(X,'mosaicOne.png','png');

%This function is used to calculate Pz(Z')/Pz(Z)
%It is used for calculating the ratio alpha1
function p = pmfZ(z,i,j,Z)
    total =0;
    for a = (i-1):(i+1)
        for b =(j-1):(j+1)
            if ~(a == i && b== j) && a <301 && b < 302 && a > 0 && b > 0
                temp = Z(a,b);
                if (z - temp) == 0
                    total = total + 1;
                end
            end
        end
    end
    r = 10;
    p = exp(r*total);
end

%This function is used to calculate fp(P')/fp(P) between the mosaic and
% the original image
%It is used for calculating the ratio alpha2
function q = jPdfZ(iG,jG,zPrime,P,A)
    total =0;
    if zPrime >0
        for k=1:3
            oriA =A(iG,jG,k);
            total = total + abs(P(zPrime,k)-oriA);
        end
    end
    conS = -1;
    q = exp(conS*total);
end

%This function is used to calculate Pz(Z')/Pz(Z) between the mosaic and
%original image
%It is used for calculating the ratio alpha1
function u = jPdfP(kG,lG,pPrime,Z,A)
    total = 0;
    for i=1:300
        for j=1:301
            if pPrime > 0
                if Z(i,j)== kG
                    for k=1:3
                if k == lG
                    oriA =A(i,j,k);
                    total = total + abs(pPrime - oriA);
                end
            end
        end
    end
  end
 end
 conS = -1;
 u = exp(conS*total);
end
```

**Output:**

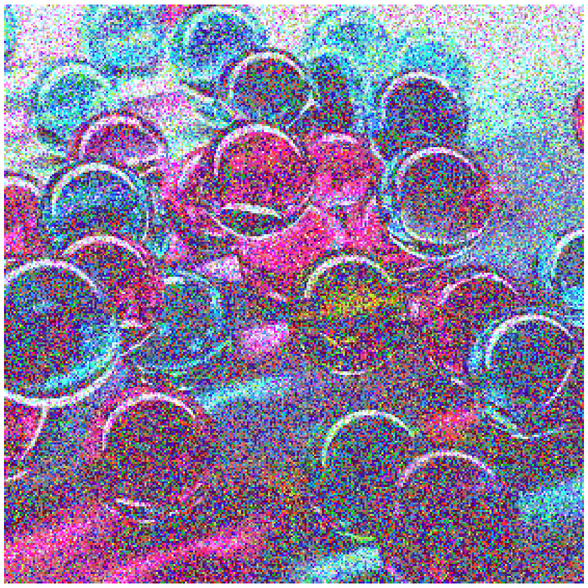


**Mosaic Image vs Real Image**

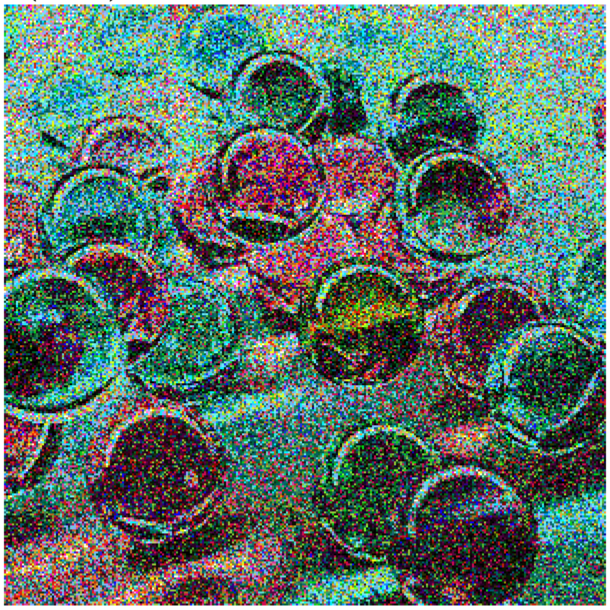
## Task 7:

1) **Change in the distribution of the color palette.**

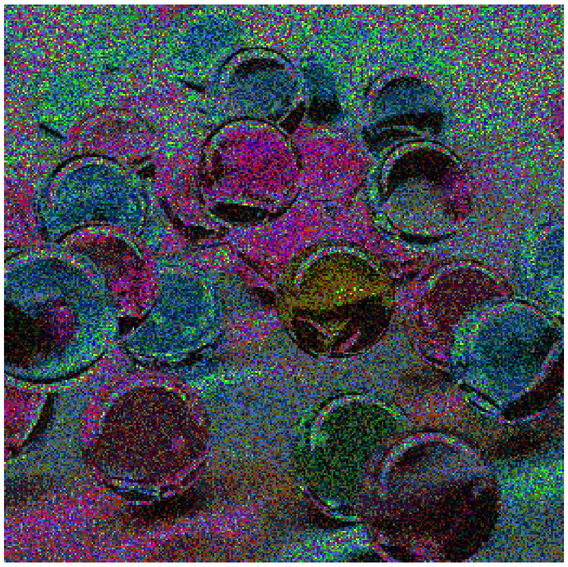
   Setting: P = 500*rand(256,3);



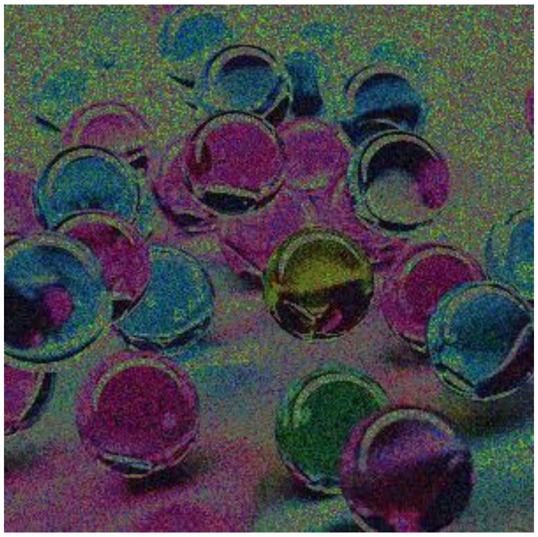
Setting: P = 255*randn(256,3);



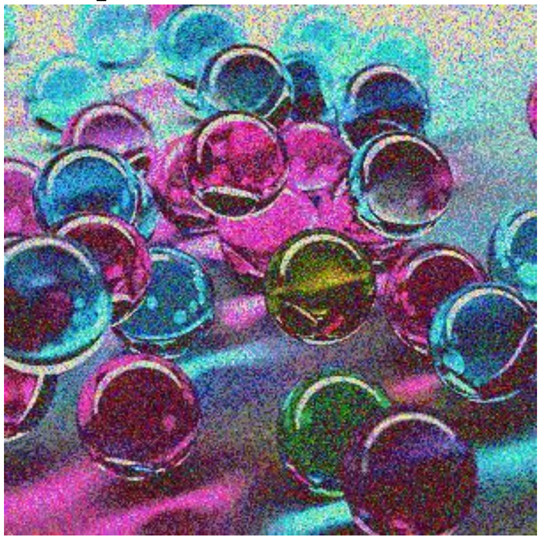
Setting: P = 100*randn(256,3);



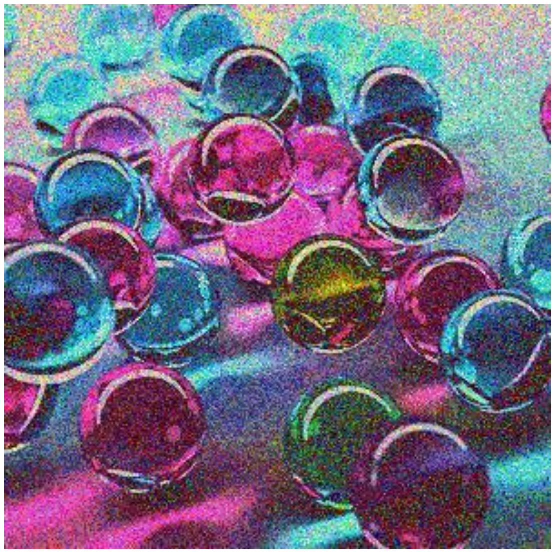
Setting: P = 100*rand(256,3);



## 2) Change in the relationship between color indexes in neighboring pixels.

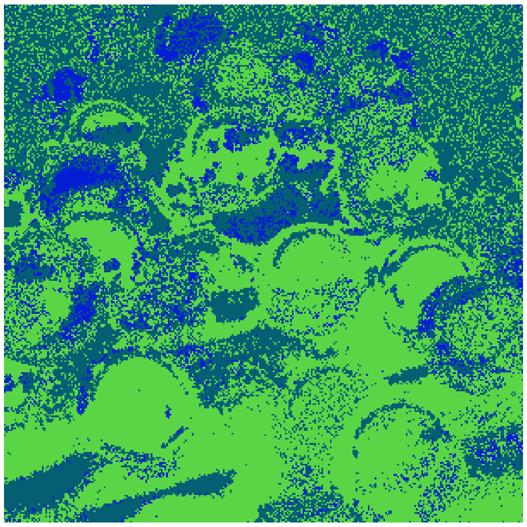


Comparing with 24 neighbors instead of 8 neighboring pixels:

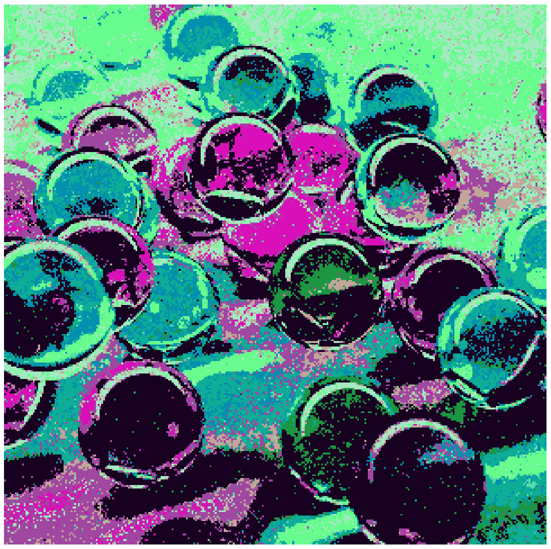


3) **Change what it means for the mosaic color and the real photo color to be “close”.**

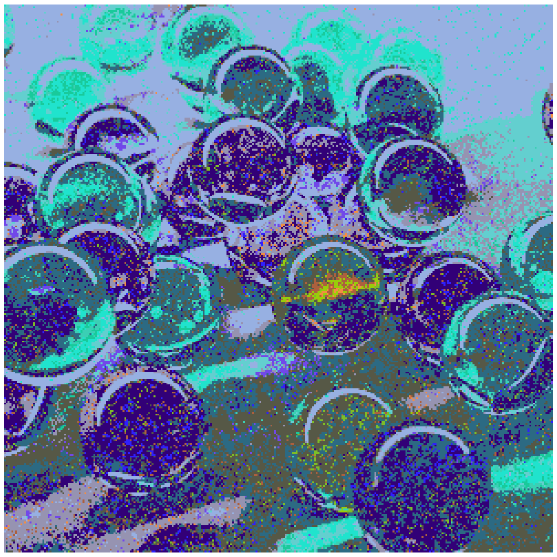
Setting number of colors to 3:



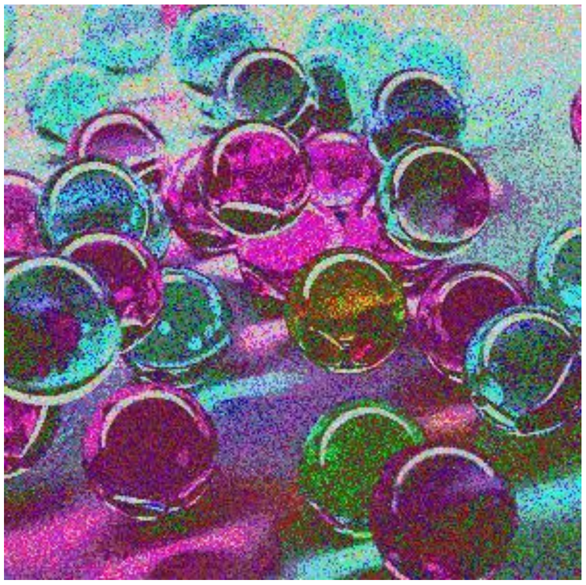
Setting number of colors to 10:



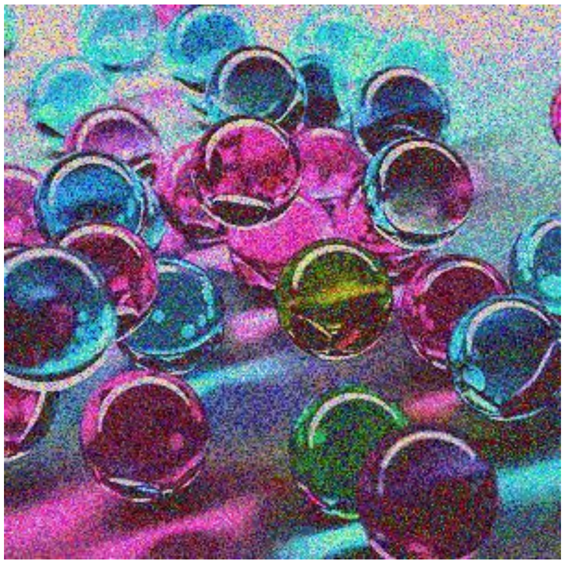
Setting number of colors to 15:



Setting number of colors to 100:



Setting number of colors to 256:



I understand that the color value for R B and G ranges from 0 – 255. I was just curious to know what will happen if the color values are set to range from 0 to 500: