Financial Informatics (Ricardo Mancera Math 512): Project #2

University of Southern California

MATH 512: Financial Informatics and Simulation

Wenyang Sun

Troy Tao

Xihao Wang

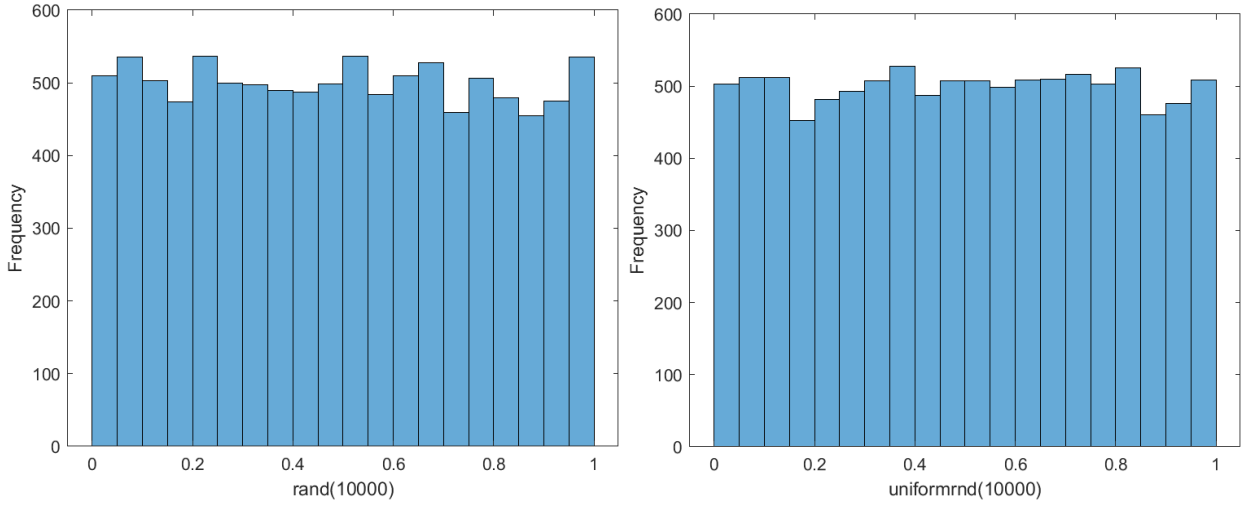
Feb 7, 2022

Question 1

Uniform Random Number Generator

a) c) d)The uniform random number generator is our most important function, as most other random number generators will call this function. To ensure the randomness, we set seed with MATLAB built-in function clock, which allows us to generate random initial x value.

Histogram:



Graph on the right hand is the histogram asked in a), and graph on the left hand side is the histogram asked in c) and d). By comparing the results of each number generator, number values in these two histograms distribute without specific pattern, while uniform random number generator seems to be more likely equidistribution.

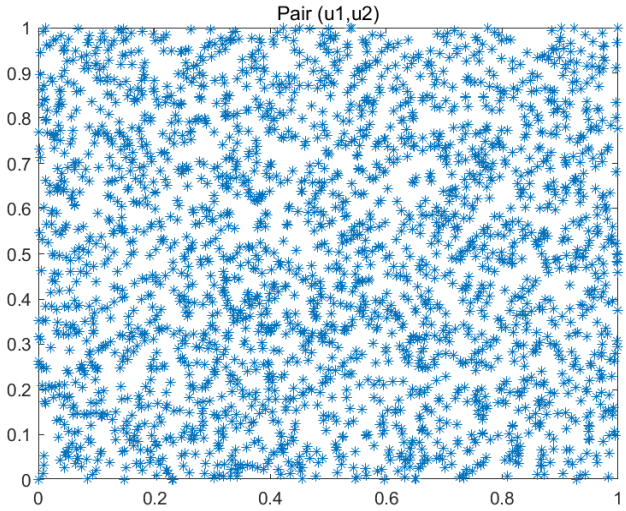
To further investigate the degree of how random are values from uniform RNG, we ran 3 statistical tests:

Statistical Test Type	Run Test	Frequency Test	Autocorrelation
Result:	p = 1.9993e-06	pv = 0	All significantly less than 1 except the first one equals to one
Interpretation:	Does not reject the null hypothesis that value from uniform RNG in random order at the default 5% significance level	Does not reject the null hypothesis that value from uniform RNG in random order at the default 5% significance level	Demonstrate low correlation between those number

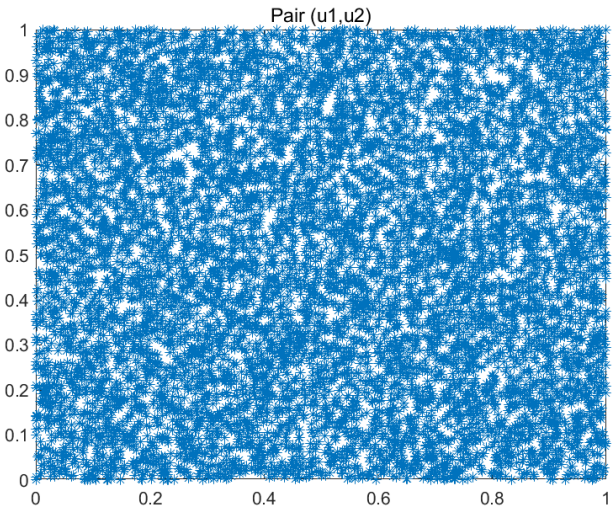
b)Generating by hand with case x0=3, a=6 and m=11 and case x0=3, a=6 and m=10:

x0 = 3, a = 6, m = 11	x0 = 3, a = 6, m = 10
x1 = (6*3) mod 11 =7	x1 = (6*3) mod 10 =8
x2 = (6*7) mod 11 = 9	x2 = (6*8) mod 10 = 8
x3 = (6*9) mod 11 = 10	
x4 = (6*10) mod 11 = 5	
x5 = (6*5) mod 11 = 8	
x6 = (6*8) mod 11 = 4	
x7 = (6*4) mod 11 = 2	
x8 = (6*2) mod 11 = 1	
x9 = (6*1) mod 11 = 6	
x10 = (6*6) mod 11 = 3	
x11 = (6*3) mod 11 = 7	
period = 10	period = 1

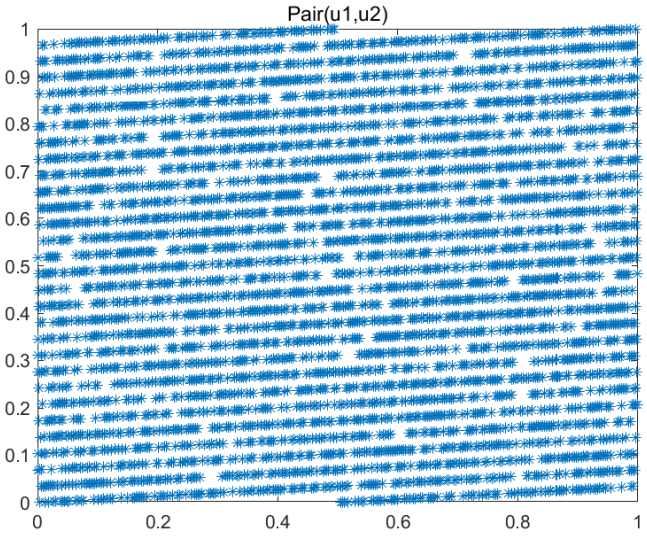
e)Pairs Plot: Tried different combinations of a, m and size to find the pattern. With large size and relatively small a and m we have some parallel lines.



A = 7^5, m = 2^31-1, size = 2500



A = 7^5, m = 2^31-1, size = 10000



A = 35831, m=9553, size = 100000

f)Advantage: The congruence method generating random numbers is largely consistent with uniform distribution.

Disadvantage: Since the correlation due to the mechanism of the method can not be eliminated, therefore proper choice of parameters is needed.

Script:

```
s = clock;
seed = ceil(s(6)*1000)
x(1)=seed;
%a = 7^5
a = 35831;
%m = 2^31-1
m = 9533;
for i = 1:100000, x(i+1) = mod(x(i)*a,m); end
uni = x/m;

x1=uni(1:99999);
y=uni(2:100000);
figure()
plot(x1,y,LineStyle='none',Marker='*')
title('Pair(u1,u2)')
%test 1: runtest return if that values in uni in random order
[h,p] = runstest(uni,median(uni))
%test 2: autocorrelation
auto = autocorr(uni)
%test 3
pv = freqtest(uni)
```

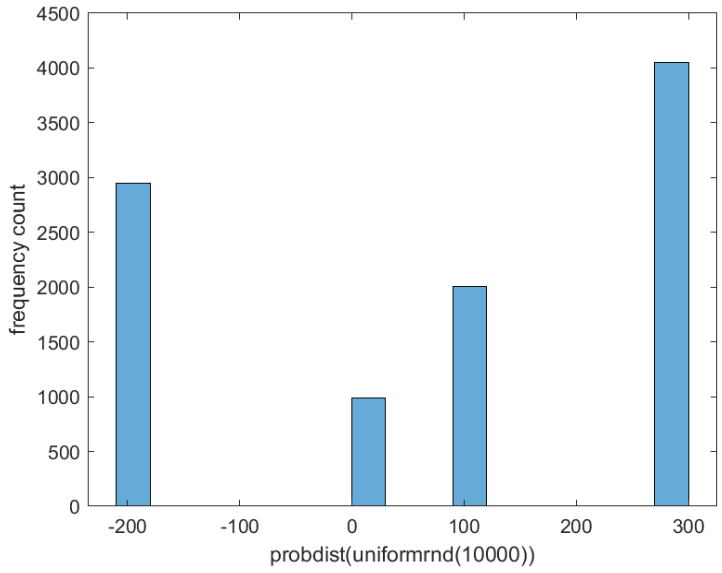
Question 2

Discrete random number generator

To generate discrete random numbers with specific probability, we utilized the uniformly distributed number in (0,1) we have in question 1. The procedure is to check which cumulative probability interval the number falls one by one, namely [0.1,0.3,0.6,1]. As the randomness of uniform random variables in problem 1 is sufficient, by using those values, the discrete probability distribution is close to desired.

Histogram:

Financial Informatics (Ricardo Mancera Math 512): Project #2



Value	Count	Percent
-200	2952	29.52%
15	991	9.91%
100	2005	20.05%
300	4052	40.52%

Script:

```
function y = myprob(uni)
P = [0.1, 0.3, 0.6, 1];
X = [15, 100, -200, 300];
    counter = 1;
    y = zeros(length(uni),1);
    for i = 1:length(uni)
        counter = 1;
        while(uni(i) > P(counter))
            counter = counter +1;

        end

    y(i) = X(counter);
    end
end
function p_value=freqtest(uni)
% Input:    e is the binary sequence being tested.
% Output:   p_value is the required probability value (if p_value < 0.01,then we conclude that the sequence is non-random.
%           Otherwise, we conclude that the sequence is random.)
z=2*uni;
c=histc(z,[0 2]);
s=c(2)-c(1);
sn=abs(s/sqrt(c(1)+c(2)));
p_value=erfc(sn/sqrt(2));
end
```

Question 3

Generate 6000 Binomial distributed (n = 100, p = 0.7) random numbers by doing: a) Generate Bernoulli random variables and add the results. Plot the histogram and use your data to calculate the probability that the Binomial random variable is less or equal to 70. Compare with the theoretical answer.

Similar to our process of generating discrete random numbers, we will first call our uniform number generator and then generate a binomial distribution in two different ways.

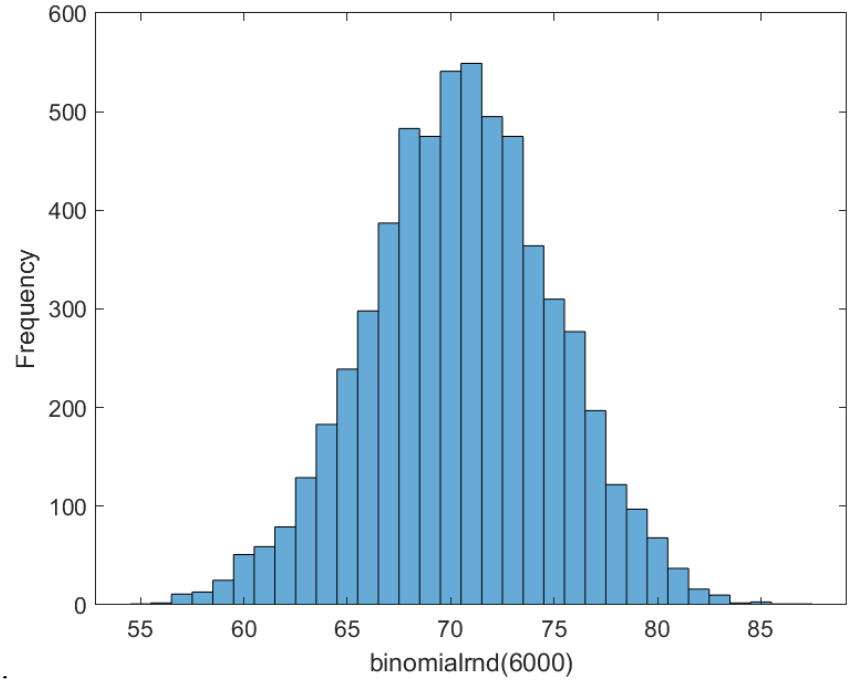
Binomial Random Number Generator

binomialrnd(n,p,N)

The straightforward method to generate 6000 random values with a binomial distribution is to directly summing counts for n times judging by the uniformly generated numbers, That is:

$$P(Xi = k) = 1\text{-}p \text{ (for } k = 0), p \text{ (for } k=1)$$

Below we show the histogram for 6000 binomially distributed random values generated in this way, with parameters p = 0.7, n = 100.



The histogram for this method shows that the distribution is accurate.

Inverse Transformation Method

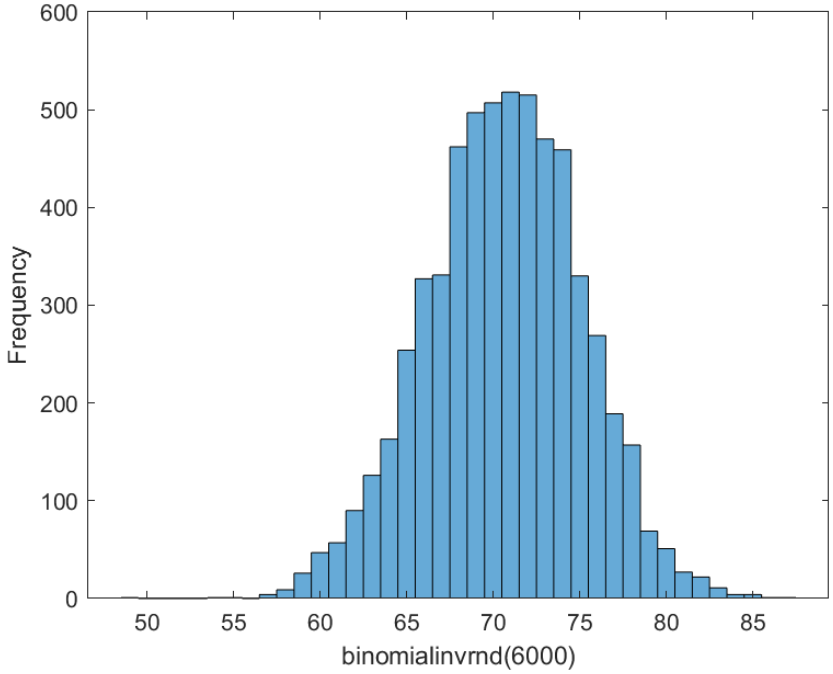
binomialinvrnd(n,p,N)

An alternative method to generate 6000 random values with a binomial distribution is to directly apply the inverse transformation method to the binomial distribution function. That is:

$$P(Xi = m) = C(n,m)*p^m*(1-p)^(n-m)$$

$$C(n,m) = fac(n)/(fac(n-m)*fac(m))$$

In the inverse transformation method we directly compute the probability of the binomial less than the set value. This way we only need to use our uniform random generator to get 6000 values that will each then be converted into a binomial value, as opposed to the Bernoulli method will requires 100*6000 = 600000 uniform random numbers to be generated and summed. The inverse transformation method will execute a lot faster compared to the previous method. The histogram for this method is shown below, the distribution pattern is still accurate.



Methods	Mean	Variance	P(X<=70)	Runtime(sec)
binomialrnd(100,0.6,6000)	70.5833	20.6395	0.4887	0.1523
binomialinvrnd(100,0.6,6000)	70.6325	20.4597	0.4805	0.0125
Theoretical Value	70	21	0.4779	/

As we can see from the comparison, the mean and the variance of both methods are closed to the theoretical value. The inverse transformation method directly applied to the cumulative distribution function of the binomial random variable, the run time is significantly lower than generating Bernoulli values first and then summing them.

Round off error existed in the histogram. The value at the tail is significantly small.

Script:

```
%binimial random numbers n=100 p=0.7 N=6000
[x4,tq31] = binomialrnd(100,0.7,6000);
figure; histogram(x4);
xlabel('binomialrnd(6000)'); ylabel('Frequency');

%implementing using the inverse transformation method
[x4i,tq32] = binomialrnd(100,0.7,6000);
figure; histogram(x4i);
xlabel('binomialinvrnd(6000)'); ylabel('Frequency');
%calculating the mean/variance and Probability of occurance of each method
tq3 = 0;
for i =0:70
tq3=tq3+factorial(100)/(factorial(i)*factorial(100-i))*0.7^i*0.3^(100-i);
end
binmean = mean(x4);
biiinvmean = mean(x4i);
binvar = var(x4);
bininvvar = var(x4i);
Pbin = length(find(x4<=70))/length(x4);
Pbininv = length(find(x4i<=70))/length(x4i);
```

```
function [y,t] = binomialrnd(n,p,N)
    tic;
    y = zeros(1,N);

    for i=1:N
        rn = uniformrnd(100,0,1,rand(1)*10000,7^5,0,2^31-1,0);
        y(i) = 0;
        for j=1:n
            if rn(j) <= p
                y(i) = y(i) + 1;
            end
        end
    end
    t = toc;
end

function [y,t] = binomialinvrnd(n,p,N)
    tic;
    rn = uniformrnd(100,0,1,4,7^5,0,2^31-1,0);
    y = zeros(6000,1);
    cdf = zeros(n+1,1);
    cdf(1) = (1-p)^n;

    for i = 1:n
        cdf(i+1) = cdf(i) + factorial(n)/(factorial(i)*factorial(n-i))*p^(i)*(1-p)^(n-i);
        y(rn(i)>cdf(i) & rn(i)<=cdf(i+1)) =  i;
    end
    t = toc(t);
end
```

Question 4

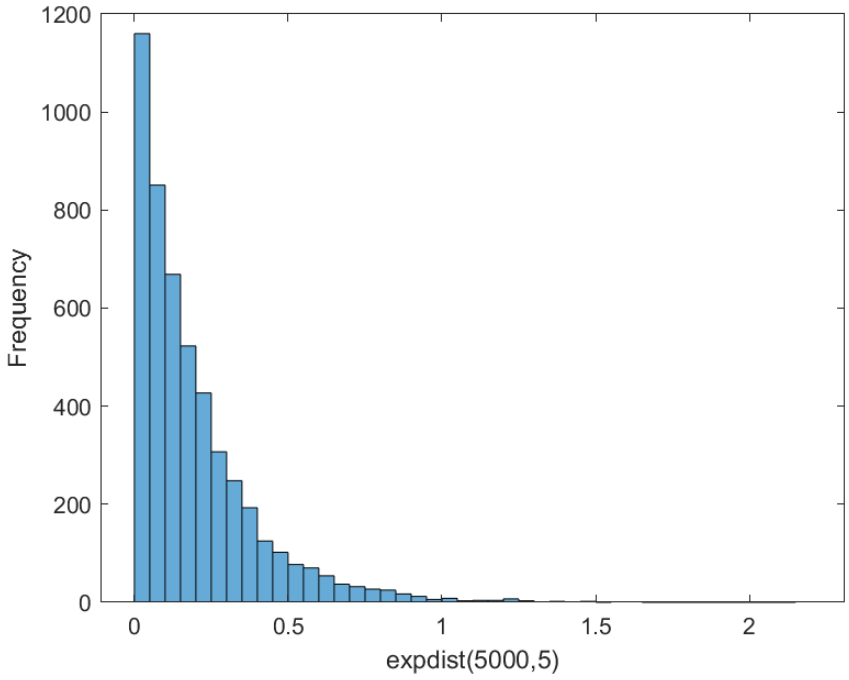
Exponential Random Number Generator

Generate 5000 Exponentially distributed with mean $\lambda = 5$ random numbers using the inverse transformation method mentioned in class. Plot the histogram of your results. Compare with the theoretical exponential density.

Algorithm:

1. Generating uniform random number U(5000,0,1) ;
2. expdist = -log(1-U)/λ;

In order to show the generator more intuitively, we draw the histogram for the generated random numbers:



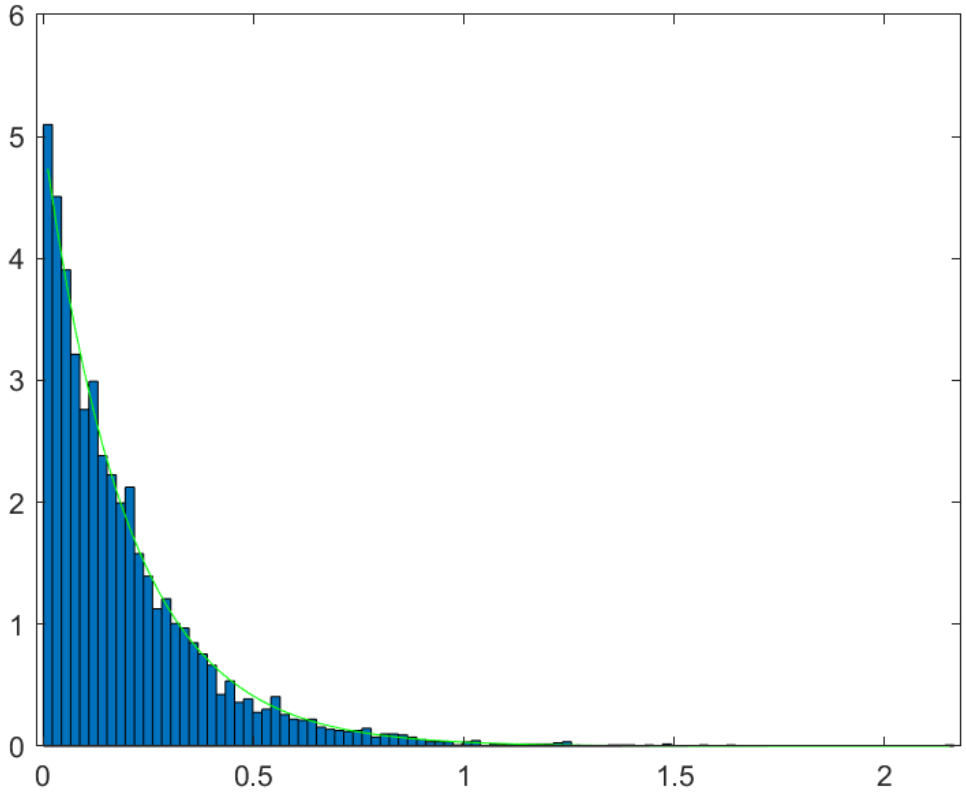
In order to verify whether the generated exponential random numbers are in accordance with the exponential probability density function, we compare the generated number with the theoretical value. Firstly, we transform the data into probability density functions by adjusting the Frequency value by scale.

We placed the generated numbers into 100 equal spaced bins and then adjusted the occurrence by the total number of samples to keep the summation as 1.

```
[yx,yy] = histogram(x5,100)
yxadjust = yx/N/mean(diff(yy))
```

Then we plot the theoretical exponential function:

```
y = lambda * exp(-lambda*y)
```



We plot the generated distribution and the theoretical function on the same plot, from the graph we can conclude that the exponentially distributed generator fits the theoretical function quite well, which is reliable.

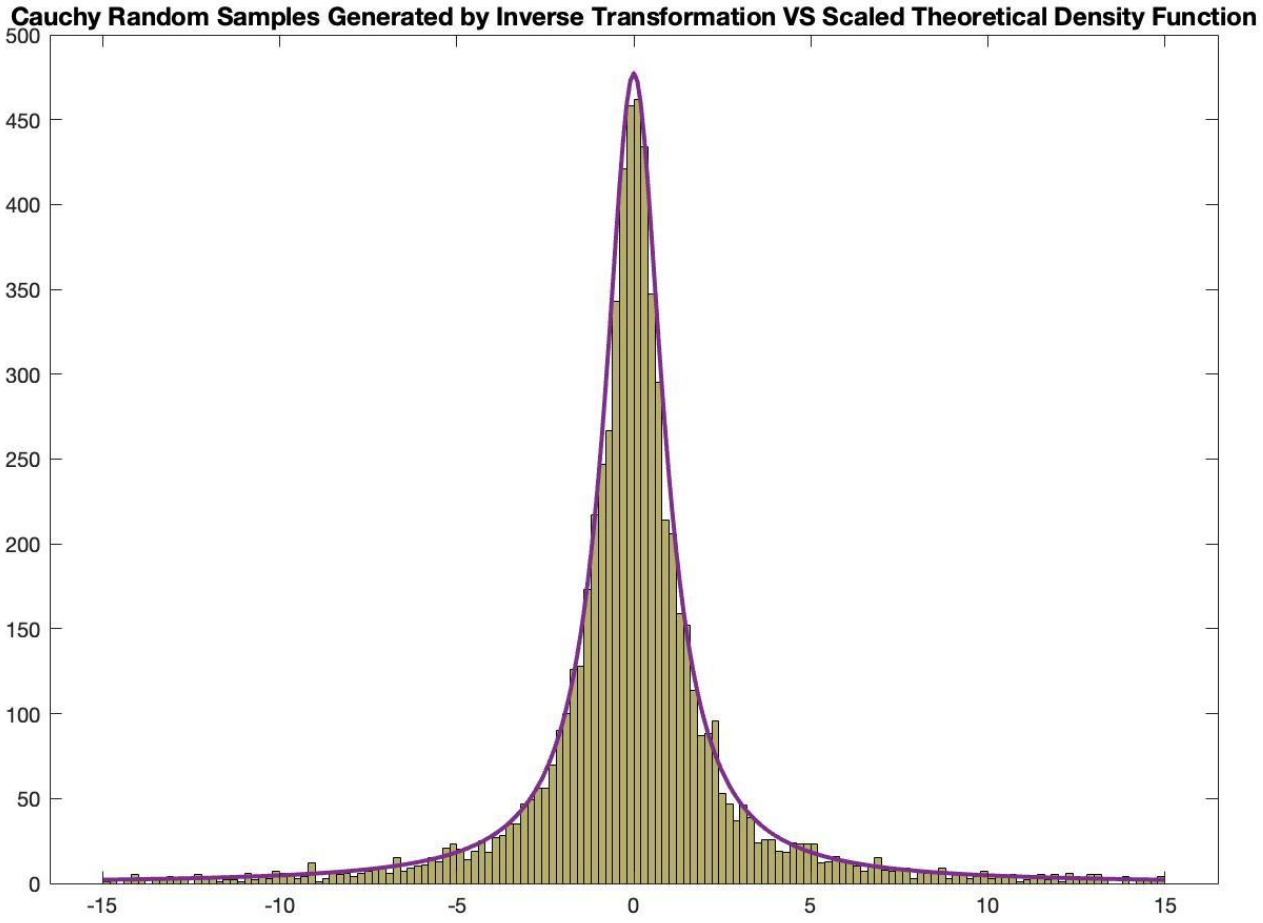
Script:

```
%q4
N = 5000;
lambda = 5;
s = clock;
seed = ceil(s(6)*1000);
x5 = uniformrnd(5000,0,1,seed,7^5,0,2^31-1,0);
x5 = -log(1-x5)/5;
figure; histogram(x5);
xlabel('expdist(5000,5)'); ylabel('Frequency');

[yx,yy] = histogram(x5,100);
%adjust the count to fit as a probability distribution
yxadjust = yx/5000/mean(diff(yy));
figure; bar(yy,yxadjust,1); hold on;
%plot the theoretic exponential value
plot(yy,lambda*exp(-lambda*yy),'g');
```

Question 5

To generate 7000 Cauchy distributed random numbers by inverse transformation, first we need to generate the same amount of uniform(0,1) random numbers. Here we used the congruence method. Then derive the inverse function of the distribution function of standard Cauchy distribution and plug uniform random variables into the inverse function. The outcomes will be close to Cauchy distributed samples. Next, we displayed the distribution of the transferred numbers into a histogram. Finally, we superimposed the scaled theoretical density function of Cauchy distribution to compare with (See the figure below).



As we see from the histogram, the distribution of generated Cauchy random numbers fits the theoretical distribution well. Note that since the sample size is large, we limited the range of generated numbers to be counted to eliminate outliers. The purpose of doing that is to avoid weird scaling of the histogram due to some large deviation from the mean and to keep the main pattern of the distribution. Another adjustment we made is to scale the density so that the superimposition gives a better visualization. See the code below for more details.

```
% generate uniform random variables using congruence method
U_5 = zeros(7000,1);
U_5(1) = a;

for i = 1:length(U_5)-1
    U_5(i+1) = mod(a*U_5(i),m);
end
U_5 = U_5/m;
```

```
% convert uniform random variables into Cauchy distributed by inverse
% transformation method.
Cauchy = tan(pi*(U_5-0.5));

figure(8)
histogram(Cauchy,[-15:0.2:15])
hold on

x = -15:0.1:15;
scal = 1500;
% for the purpose of comparing, we scale the density function so we can
% superimpose the function graph onto the histogram and compare their
% shape.
y = 1./(pi*(1+x.^2))*scal;
plot(x,y)
```

Question 6

We want to generate 10000 independent standard normal distributed random samples using Box-Muller method, Marsaglia-Bray method, the acceptance-rejection method, and the MATLAB built-in function 'randn()'. We also record the time each method takes and compare the histograms generated along with the theoretical Gaussian density. Note that we choose to use the built-in uniform random number generator function 'rand()' instead of the congruence method to generate uniform random numbers because the correlation between pairs would affect the independence of the samples.

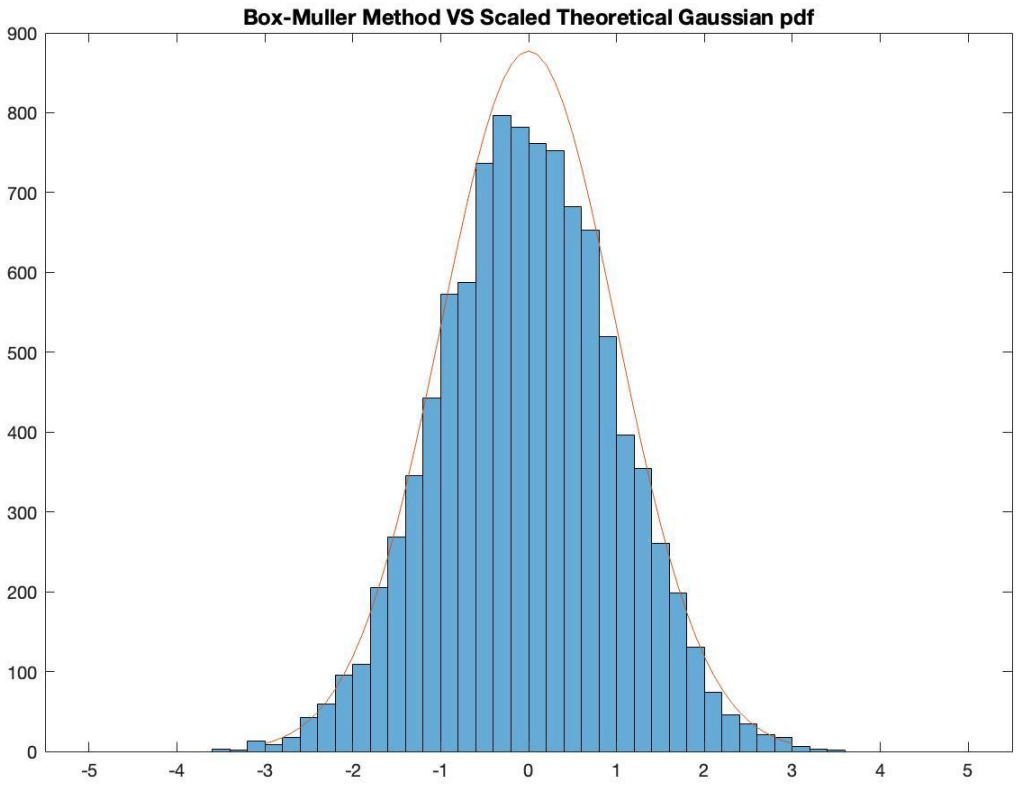
Box-Muller

Algorithm:

3. Take 2 independent uniform(0,1) random numbers, U and V;
4. Z1 = sqrt(-2log(U))*cos(2pi*V); Z2 = sqrt(-2log(U))*sin(2pi*V);
5. Save Z1 and Z2 into the array for Gaussian random numbers;
6. Repeat 5000 times

Run Time: 0.004012 seconds.

Histogram:



Script:

```
%% 6a
% generate uniform random variables
U_6 = rand(100000,1);
% we are not using the modulo-generated psuedorandom number because of the
% linear correlation between consecutive uniform random variables

% Gaussian density
G_x = -3:0.1:3;
G_y = (2*pi)^0.5*exp(-0.5*G_x.^2);

% Box-Muller Method
tic
Z_BM = zeros(10000,1);
for i = 1:length(Z_BM)/2
    Z_BM(2*i-1) = (-2*log(U_6(2*i-1)))^(0.5)*cos(U_6(2*i)*2*pi);
    Z_BM(2*i) = (-2*log(U_6(2*i-1)))^(0.5)*sin(U_6(2*i)*2*pi);
end
toc

figure(9)
histogram(Z_BM,[-5:0.2:5])
hold on
plot(G_x,350*G_y) % scale = 350
title('Box-Muller Method VS Scaled Theoretical Gaussian pdf',FontSize = 12)
```

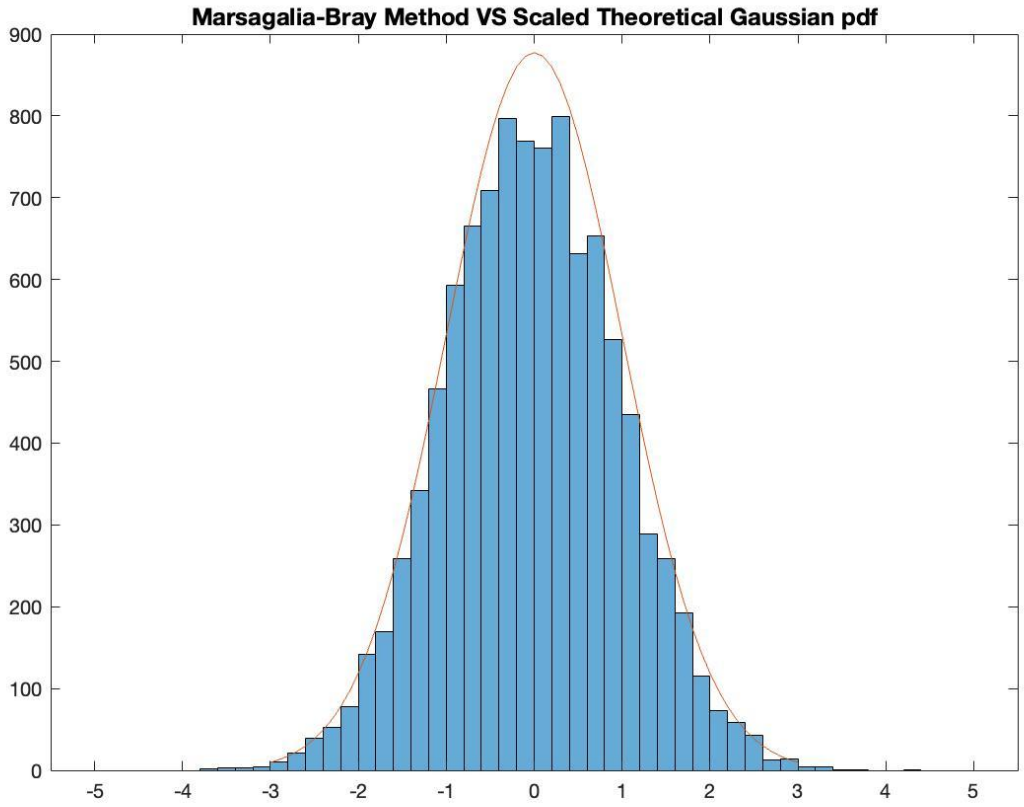
Marsaglia-Bray

Algorithm:

1. Take 2 independent uniform(0,1) random numbers U, V;
2. U = 2*U-1; V = 2*V-1 so that it is in uniform(-1,1); Let S = U^2+V^2;
3. If S >= 1, go back to 1; Otherwise, Z1 = U*sqrt(-2ln(S)/S); Z2 = V*sqrt(-2ln(S)/S);
4. Save Z1 and Z2 into the array for standard normal random numbers.
5. Repeat until we have 10000 such random numbers.

Run Time: 0.005828 seconds.

Histogram:



Script:

```
%% 6b
% Marsaglia-Bray Method
tic
Z_MB = zeros(10000,1);

i = 1;
ptr = 1; % pointer for uniform variables
while i <= length(Z_MB)
    x = U_6(ptr)*2-1;
    y = U_6(ptr+1)*2-1;
    s = x^2+y^2;
    if s<=1
        Z_MB(i) = x*(-2*log(s)/s)^(0.5);
        Z_MB(i+1) = y*(-2*log(s)/s)^(0.5);
        i = i+2;
    end
    ptr = ptr+2;
end
toc
figure(10)
histogram(Z_MB,[-5:0.2:5])
hold on
plot(G_x,350*G_y)
title('Marsagalia-Bray Method VS Scaled Theoretical Gaussian pdf',FontSize = 12)
```

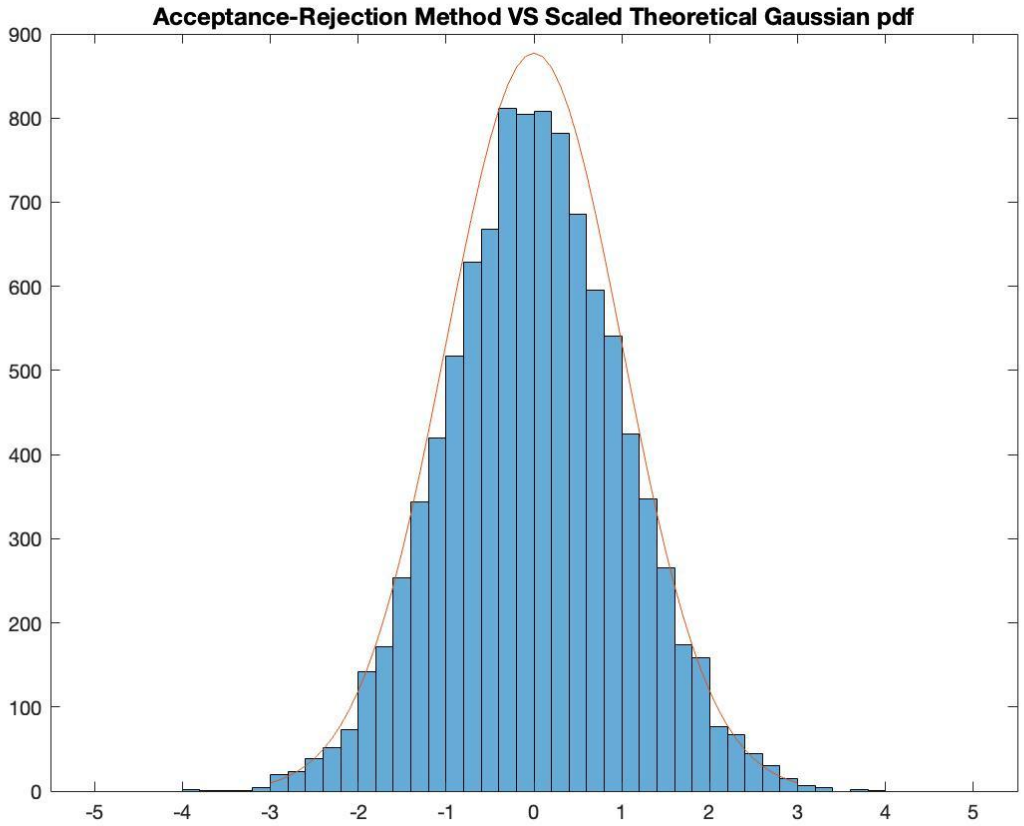
Acceptance-Rejection

Algorithm:

1. Take 3 uniform(0,1) random numbers U,V,W;
2. Let G = -ln(U), to represent a random number taken from exponential distribution.
3. If ln(V) < ln(f(G)/c*g(G)) = -½(G-1)^2, we accept G as an absolute value of standard normal random variable. Otherwise, go back to 1.
4. Z = G*(-1)^(W <= 0.5). The sign of the standard normal is decided by W equally likely.
5. Save Z to the array for standard normal variables and repeat until we have 10000 standard normal random numbers.

Run Time: 0.013402 seconds.

Histogram:



Script:

```
%% 6c
% acceptance-rejection method
tic
Z_ar = zeros(10000,1);

i = 1;
ptr = 1;
while i<= length(Z_ar)
    X1 = -log(U_6(ptr));
    X2 = -log(U_6(ptr+1));
    X3 = U_6(ptr+2);
    if X2 >= (X1-1)^2/2
        Z_ar(i) = abs(X1)*(-1)^(X3<=0.5);
        i = i+1;
    end
    ptr = ptr+3;
end
toc
figure(11)
histogram(Z_ar,[-5:0.2:5])
hold on
plot(G_x,350*G_y)
title('Acceptance-Rejection Method VS Scaled Theoretical Gaussian pdf',FontSize = 12)
```

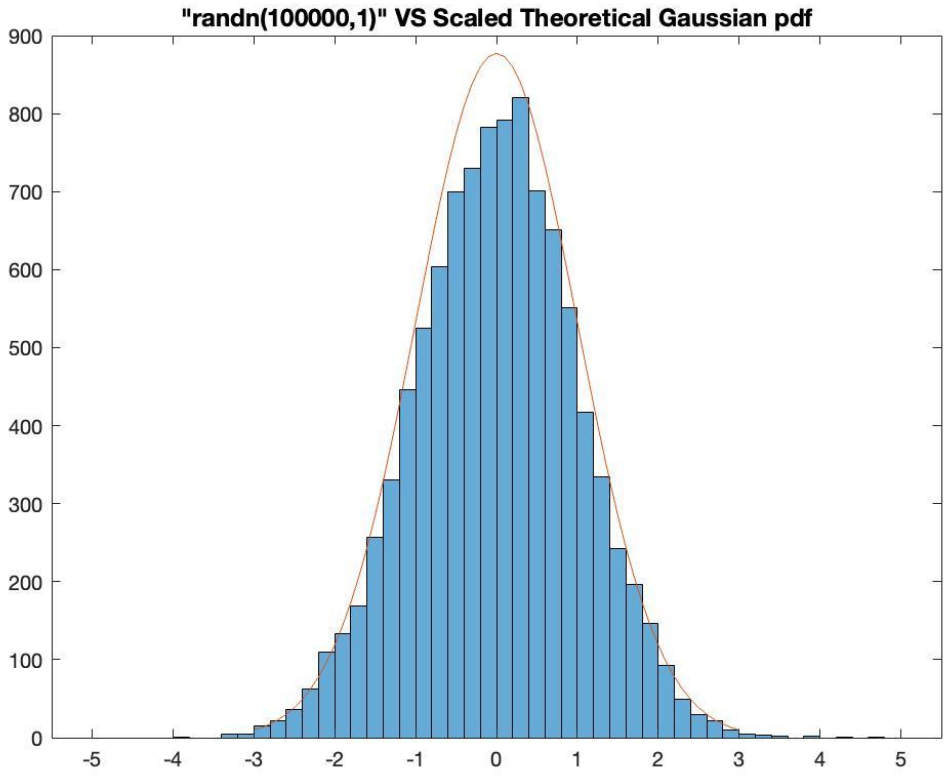
'randn()'

Algorithm:

1. Z = randn(100000,1)

Run Time: 0.000333 seconds.

Histogram:



Script:

```
%% 6d
tic
Z = randn(10000,1);
toc
figure(12)
histogram(Z,[-5:0.2:5])
hold on
plot(G_x,350*G_y)
title('"randn(100000,1)" VS Scaled Theoretical Gaussian pdf',FontSize = 12)
```

Discussion

Comparing four methods, we found that all four of them fit the scaled theoretical pdf well. Here is a table of run time:

Method	Box-Muller	Marsagalia-Bray	acceptance-rejection	'randn()'
Run time(s)	0.004012	0.005828	0.013402	0.000333

According to the table, the built-in function 'randn()' is the fastest way to generate 100000 standard normal random numbers. Among other three implementations, Box-Muller method and Marsagalia-Bray method are faster than the acceptance-rejection method. One thing to note is that for methods like Box-Muller and Marsaglia-Bray, we need to make sure that the dependency between pairs of uniform random variables is low. Otherwise it would affect our distribution.