S7001E Lab 1

Calle Rautio
calrau-1@student.ltu.se

Jim Agnevik
jimagn-8@student.ltu.se

Contents

1 Introduction

In this lab, exercises 1-5 were completed, focusing on the statistical distribution of dice, the transmission of binary blocks, and the Gaussian distribution. The following sections provide the answers and discussions related to the tasks assigned for this laboratory exercise.

2 Exercise 1

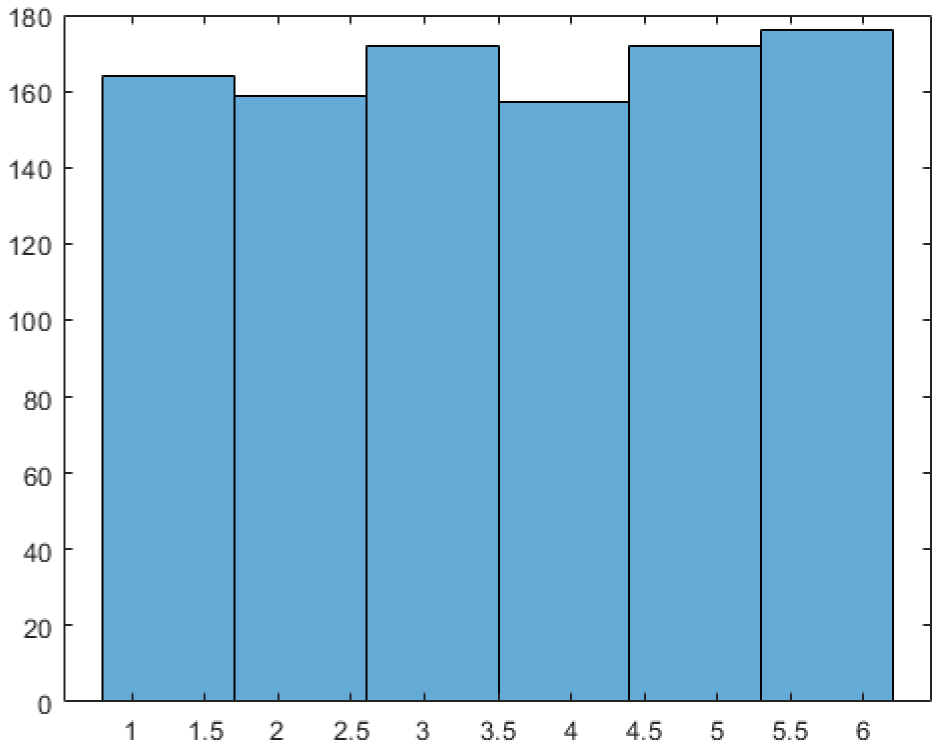
2.1 a)

The provided code was ran $N = 1000$ times and the outcomes were saved inside of an 1 by 6 array each column representing an outcome of the dice. The array was then divided by the amount of times the dice was rolled giving the estimated probability of each face.

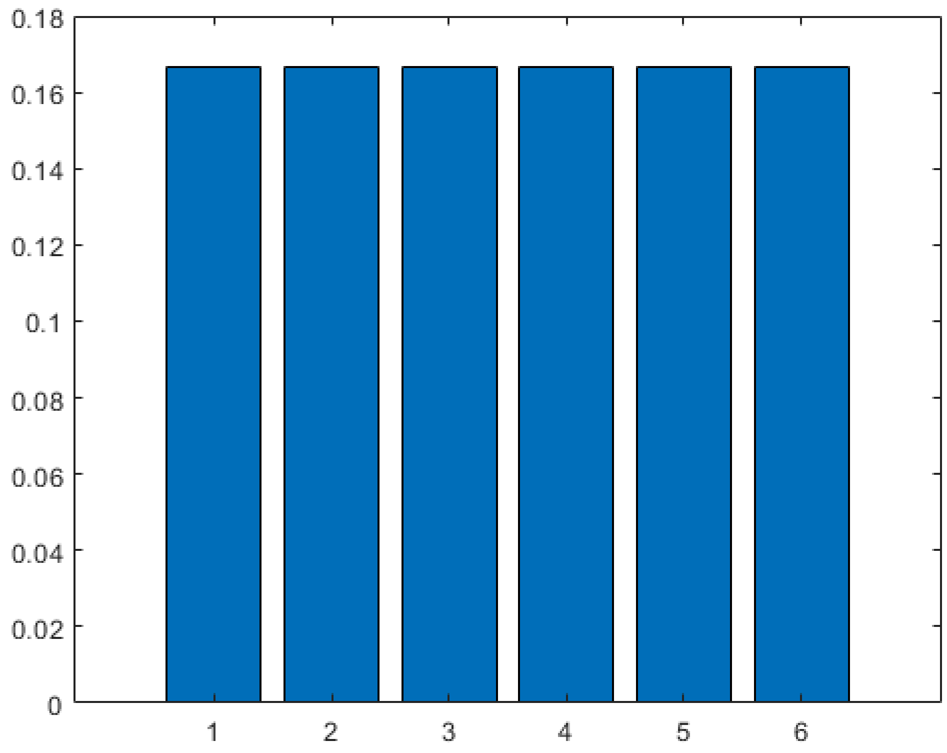
$P = \begin{bmatrix} 0.176 & 0.169 & 0.67 & 0.17 & 0.182 & 0.13 \end{bmatrix}$

The simulated value has not yet reached the values for probability was made. But if the simulation would have had a bigger amount of dice rolls the probability would have converged to the actual probability.

2.2 b)

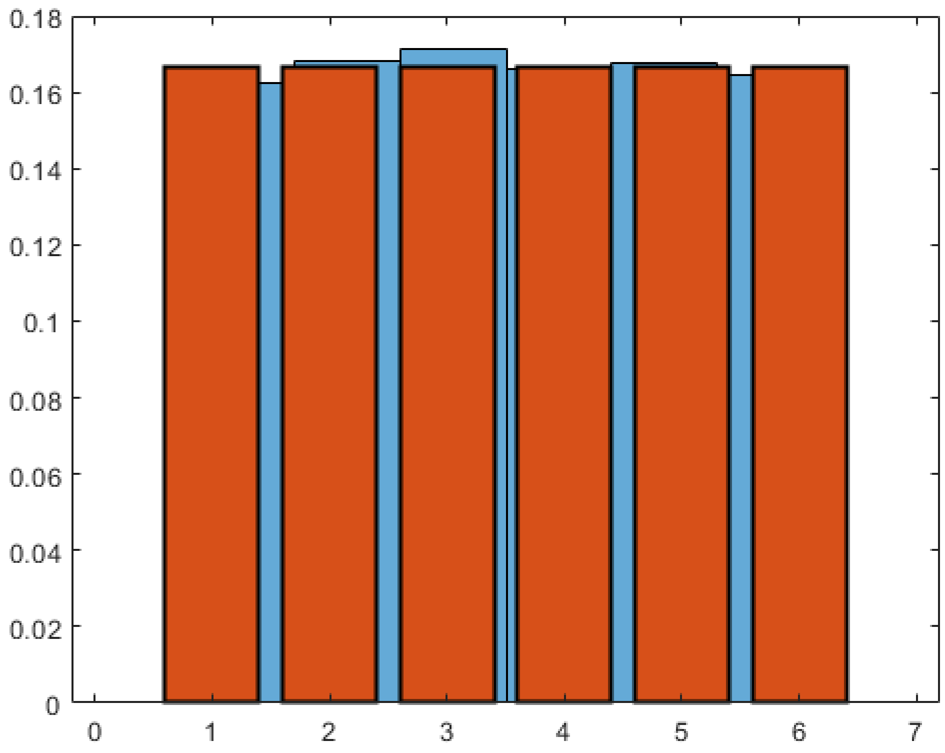


Histogram for the simulated dice



the theoretical probability density function.
The Histogram could be used as a approximation to the theoretical pdf, The more measurements the closer the Histogram will converge to the pdf.

2.3 c)



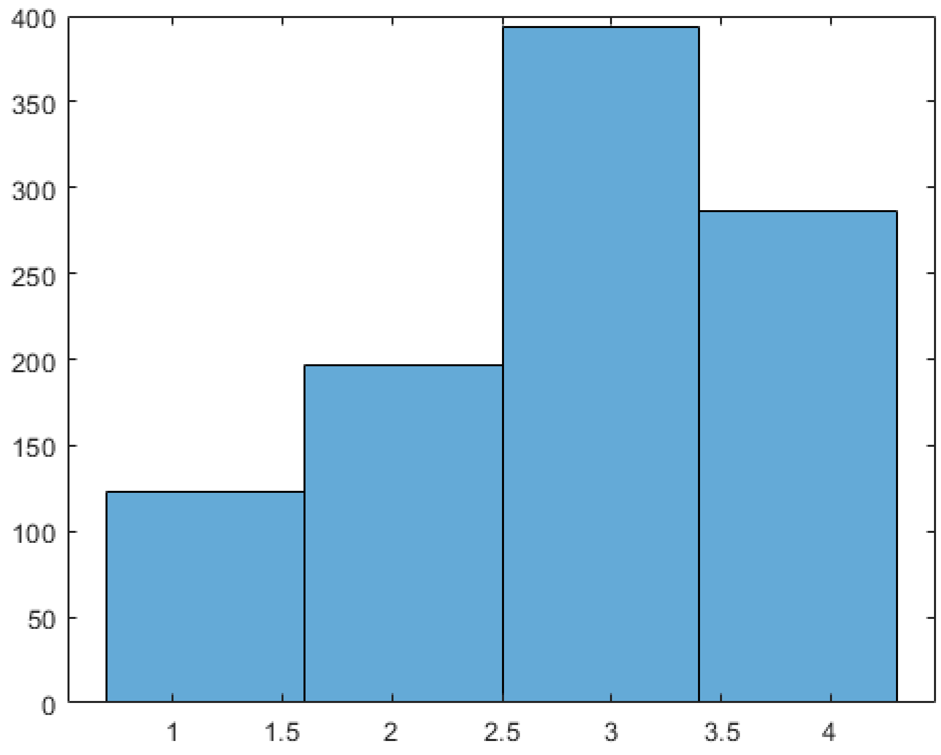
Histogram in blue scaled properly so it could be considered an estimation of the pdf in orange.

3 Exercise 2

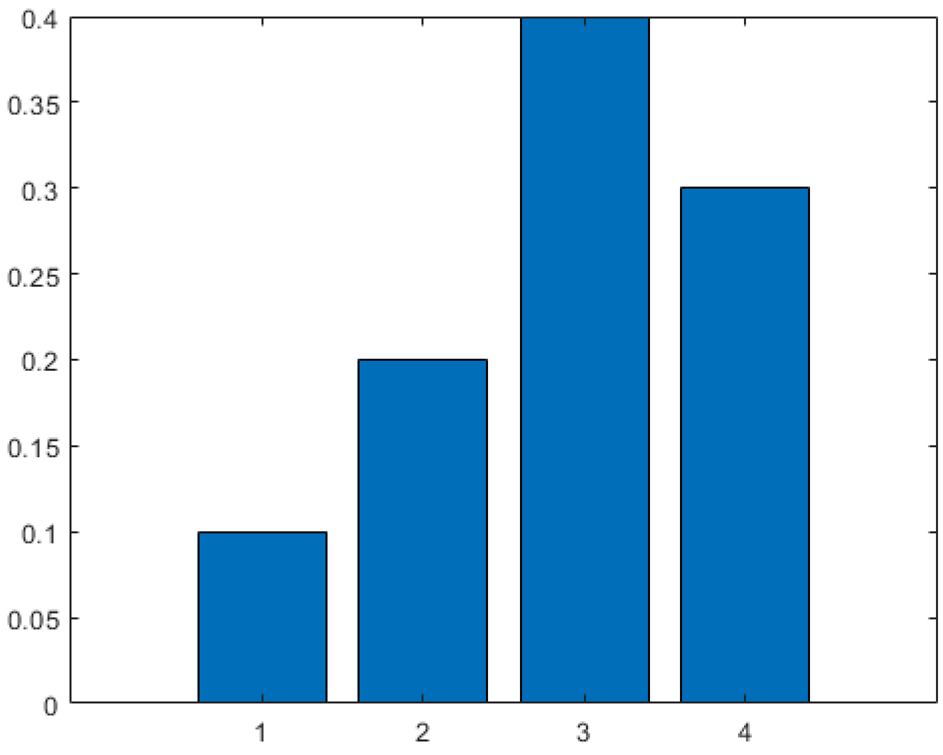
3.1 a)

As same as 1a in Exercise 1 the estimated probability of each face.
$P = \begin{bmatrix} 0.1040 & 0.1890 & 0.3980 & 0.3090 \end{bmatrix}$ The same answer as exercise 1a the simulated dice roll has not yet converged to the actual probabilities yet. But with more dice rolls it will converge more and more to the actual probabilities.

3.2 b)

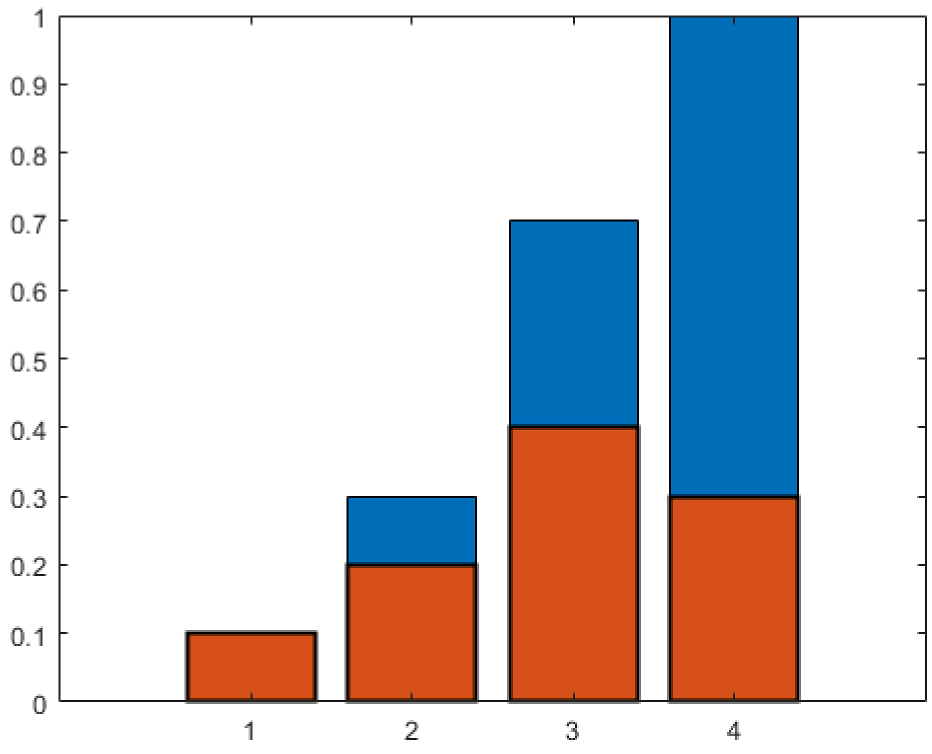


Histogram for the simulated dice



the theoretical probability density function.
The Histogram could be used as a approximation to the theoretical pdf, The more measurements the closer the Histogram will converge to the pdf.

3.3 c)



Approximation to this theoretical PDF using the cumsum command.

4 Exercise 3

4.1 a)

To calculate the theoretical probability of the sum of two dice being even, you need to consider the following:

$P(\text{Sum is even}) = P(\text{Both dice are even}) + P(\text{Both dice are odd})$

When adding two numbers, the sum is even if both numbers are even or both numbers are odd. For die #1 and die #2, the scenarios can be described as follows:

$P(\text{Sum is even}) = \frac{3}{6} * (0.2 + 0.3) + \frac{3}{6} * (0.1 + 0.4)\ (=)\ 0.5$

4.2 b)

As demonstrated in the MATLAB figure, when running a large number of simulations, the values eventually converge and match.

4.3 c)

To calculate the theoretical probability of the sum of two dice being even, given that die #2 is greater than 2, we can apply the same formula as before, but adjust it to account for the condition that die #2 is greater than 2. The calculation would look as follows:

$P(\text{Sum is even, given that die \#2 is greater then 2}) = \frac{3}{6} * 0.3 + \frac{3}{6} * 0.4 \ (=) \ 0.35$

4.4 d)

Yes, again as seen from the Matlab figure if you run a lot of simulations the values eventually match.

5 Exercise 4

5.1 a)

To determine the theoretical probability of having 2 or fewer errors in the transmission of 100 bits, the binomial distribution method can be used. By calculating $\text{BinCDF}(100, 0.001, 2)$ with a calculator, the result of approximately $0.99984...$, or about 99.9% is received.

5.2 b)

They may not match exactly, but with an infinite number of simulations, they will eventually converge to the same statistic.

5.3 c)

To determine the theoretical probability of having no errors in a transmission of 100 bits, given that less then 2 errors occurred when transmitting the block is calculated using the following formula:

$P(A|B) = \frac{P(A \cap B)}{P(B)}$ which can be calculated this way:

$\frac{\text{BinPDF}(100,0.001,0)}{\text{BinCDF}(100,0.001,1)} = 0.909... \ (=) \ 90.9\%$

5.4 d)

Over time, it will eventually converge to the correct statistic if repeated many times.

6 Exercise 5

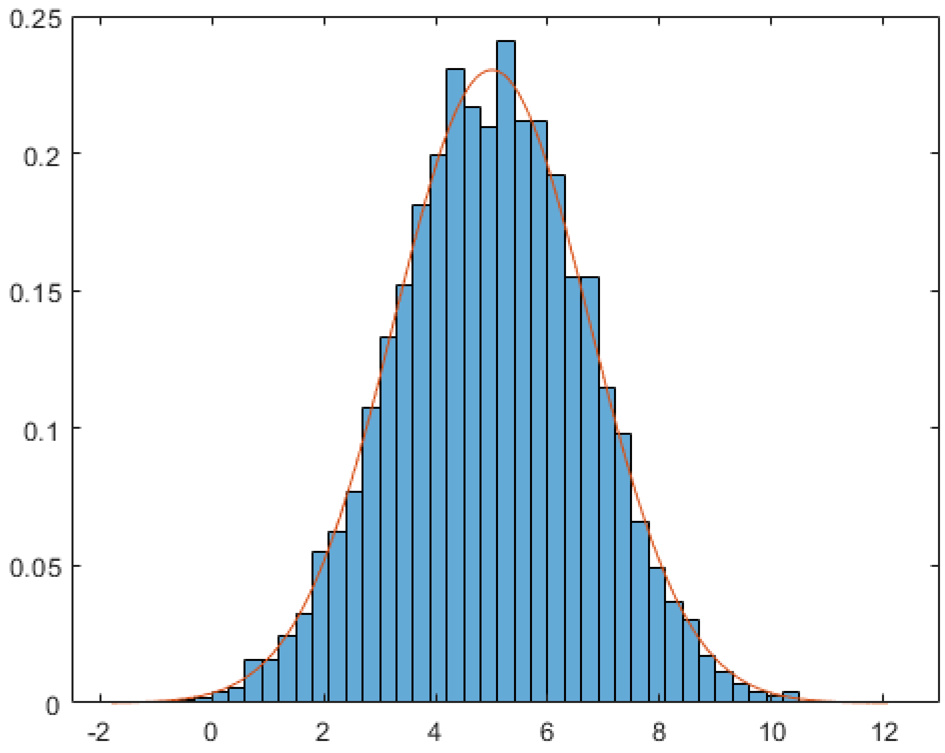
6.1 a)

A Matlab script was constructed to generate a vector X of 100 independent samples from a Gaussian distribution with mean $\mu = 5$ and variance $\sigma^2 = 3$. The mean and variance from the vector X came out as $\mu = 5.2589$ and $\sigma^2 = 3.2584$. The sample mean and sample variance do not match up with the give mean and variance. This is because of the random values that the randn function introduces. As more samples gets introduced the mean and variance value will move closer to the real given values.

6.2 b)

The Matlab script from part a was modified to do 10000 samples instead of 100. This yielded the result $\mu = 5.0076$ and $\sigma^2 = 2.9226$. Which because of Law of large numbers will make the estimated values converge closer to their values.

6.3 c)



Histogram of the X and the properly scaled histogram and theoretical pdf curve.

6.4 d)g

To determine what fractions of points that fall between $\{1 \leq X \leq 2\}$ the find function together with the length function to determine the amount of points that were between 1 and 2 and then divided with the amount of points in the total set. Which came out as 3.06%

To calculate the theoretical value for fractions of points that fall between $\{1 \leq X \leq 2\}$ equation $(2.4-13)$ in the book was used. Values for the erf function was taken from the look up table in the book.

$P[a < X \leq b] =$erf$\left(\frac{b-\mu}{\sigma}\right) -erf\left(\frac{a-\mu}{\sigma}\right)$.

$P[1 < X \leq 2] =$erf$\left(\frac{2-5}{\sqrt{3}}\right) -erf\left(\frac{1-5}{\sqrt{3}}\right) = -0.48927 + 0.45993 = 2.934\%$

```
N = 10000;

p = [1/6 1/6 1/6 1/6 1/6 1/6 ]; % probability of each face
P = [0 cumsum(p)] ;
roll = zeros(N,1); % vector to hold results of each roll
% (the memory is "pre-allocated" in this
% way to speed up Matlab processing
for i=1:N % loop over number of rolls
    x=rand(1,1);
    for j=2:length(P), % determine result of each roll
        if( (P(j-1)<x) & (x<P(j)) )
            roll(i) = j-1;
            break
        end
    end
end

Prob_roll = zeros(1,length(p));

for i=1:length(p)
    Prob_roll(1,i) = length(find(roll==i))/N;
end

%1.A
Prob_roll
total_Prob_roll = sum(Prob_roll)

%1.B
figure(1)
histogram(roll, 6, 'Normalization', 'probability');

theoretical_pdf = ones(1, 6) * 1/6;

figure(2);
bar(1:6, theoretical_pdf);
%1.C

figure(3);
histogram(roll, 6, 'Normalization', 'pdf'); % Normalizing as a PDF
hold on;
bar(1:6, theoretical_pdf, 'LineWidth', 1.5); % Plotting the theoretical PDF on top
```

```
N = 1000;

%2.A
faces = [1 2 3 4];
p = [0.1 0.2 0.4 0.3]; % probability of each face
P = [0 cumsum(p)] ;
roll = zeros(N,1); % vector to hold results of each roll
% (the memory is "pre-allocated" in this
% way to speed up Matlab processing
for i=1:N % loop over number of rolls
    x=rand(1,1);
    for j=2:length(P), % determine result of each roll
        if( (P(j-1)<x) & (x<P(j)) )
            roll(i) = j-1;
            break
        end
    end
end

Prob_roll = zeros(1,length(p));

for i=1:length(p)
    Prob_roll(1,i) = length(find(roll==i))/N;
end

Prob_roll
total_Prob_roll = sum(Prob_roll)

%2.B
figure(1)
histogram(roll,4)

figure(2)
x=linspace(1,4,1000)
y = pdf('Normal',x,mean(roll),std(roll))
plot(x,y)

figure(3)
histogram(roll,4,'Normalization','pdf')
hold on
plot(x,y)
hold off

%2.C
figure(4);
bar(faces, p);
grid on;

rolls = randsrc(1, 1000, [faces; p]);

[counts, edges] = histcounts(rolls, 'BinMethod', 'integers', 'Normalization',↙
'probability');

cumsum_hist = cumsum(counts);

figure(5);
bar(faces, counts);
grid on;
```

```
%3.B
N = 1000;

p_1 = [1/6 1/6 1/6 1/6 1/6 1/6]; % probability of each face
P_1 = [0 cumsum(p_1)] ;
roll_1 = zeros(N,1); % vector to hold results of each roll
% (the memory is "pre-allocated" in this
% way to speed up Matlab processing


for i=1:N % loop over number of rolls
    x=rand(1,1);
    for j=2:length(P_1), % determine result of each roll
        if( (P_1(j-1)<x) & (x<P_1(j)) )
            roll_1(i) = j-1;
            break
        end
    end
end

p_2 = [0.1 0.2 0.4 0.3]; % probability of each face for rigged dice
P_2 = [0 cumsum(p_2)] ;
roll_2 = zeros(N,1);

for i=1:N % loop over number of rolls
    x=rand(1,1);
    for j=2:length(P_2), % determine result of each roll
        if( (P_2(j-1)<x) & (x<P_2(j)) )
            roll_2(i) = j-1;
            break
        end
    end
end

Togheter_roll = roll_1+roll_2;
Togheter_chance_odd_or_even = zeros(1,2);%First value is even, second is odd.

for i=1:N
    if ~mod(Togheter_roll(i),2)
        Togheter_chance_odd_or_even(1) = Togheter_chance_odd_or_even(1)+1;
    else
        Togheter_chance_odd_or_even(2) = Togheter_chance_odd_or_even(2)+1;
    end
end

figure(1)

%3.D
bar(Togheter_chance_odd_or_even) %First value is even, second is odd.

Togheter_chance_odd_or_even_2 = zeros(1,2); %First value is greater than 2, second is 2.
```

```
for i=1:N
    if roll_2(i) > 2 && ~mod(Togheter_roll(i),2)
        Togheter_chance_odd_or_even_2(1) = Togheter_chance_odd_or_even_2(1)+1;
    else
        Togheter_chance_odd_or_even_2(2) = Togheter_chance_odd_or_even_2(2)+1;
    end
end

figure(2)
bar(Togheter_chance_odd_or_even_2) %First value is greater than 2 and even, second is↙
everything else.
```

```
N_blocks = 1000;

N = 100;
p = [0.001 0.999]; % probability of transmission success transmission error
P = [0 cumsum(p)] ;
Transmission = zeros(N,1); % vector to hold results of each Transmission
% (the memory is "pre-allocated" in this
% way to speed up Matlab processing

Transmission_succes = zeros(1,2);

for k = 1:N_blocks
    for i=1:N % loop over number of Transmission
        x=rand(1,1);
        for j=2:length(P), % determine result of each Transmission
            if( (P(j-1)<x) & (x<P(j)) )
                Transmission(i) = j-1;
                break
            end
        end
    end
end
%4.B
N = 10000;

% Probability of transmission success and transmission error
p = [0.999 0.001];
P = [0 cumsum(p)];

% Vector to hold results of each transmission
Transmission = zeros(N, 1);

% Loop over number of transmissions
for i = 1:N
    x = rand(1, 1);
    % Determine result of each transmission
    for j = 2:length(P)
        if P(j-1) < x && x < P(j)
            Transmission(i) = j - 1;
            break;
        end
    end

    Prob_signal = zeros(1,length(p));

    for l=1:length(p)
        Prob_signal(1,l) = length(find(Transmission==l));
    end

    if Prob_signal(1,1) >= 2
        Transmission_succes(1) = Transmission_succes(1) + 1;
    else
        Transmission_succes(2) = Transmission_succes(2) + 1;
    end

end

Transmission_succes/N_blocks
Prob_signal = zeros(1, length(p));
for i = 1:length(p)
    Prob_signal(1, i) = sum(Transmission == (i - 1)) / N;
end

disp(Prob_signal);

%4.D
N = 100000;
p = 0.001;

n_less_2_error = 0;
n_given_less_2_error = 0;

for i = 1:N
    bits = rand(1, 100) < (1 - p);
    num_errors = sum(~bits);

    if num_errors < 2
        n_less_2_error = n_less_2_error + 1;

        if num_errors == 0
            n_given_less_2_error = n_given_less_2_error + 1;
        end
    end
end

result = n_given_less_2_error / n_less_2_error;
disp([1-result, result])
```

```matlab
N_1 = 100; %Samples
N_2 = 10000; %Samples 2
mu = 5; %Mean
V = 3; %Variance
sigma = sqrt(V); %std


X_1 = mu + sigma * randn(N_1, 1); %Samples with Guassian distribution
mean_X_1 = mean(X_1) % mean
V_1 = var(X_1) % Variance

X_2 = mu + sigma * randn(N_2, 1); %Samples with Guassian distribution
mean_X_2 = mean(X_2)  % mean
V_2 = var(X_2) % Variance

x = linspace(min(X_2), max(X_2), 1000);
y = pdf('Normal',x, mu, sigma);

figure(1);
histogram(X_2, 'Normalization', 'pdf');
hold on
plot(x,y)
hold off

Between_1_2 = find(X_2 >= 1 & X_2 <= 2);
Amount_Between_1_2 = length(Between_1_2);
Fraction_in_point = Amount_Between_1_2 / N_2

% DO NOT USE!!!!!!!!!!!!!!!
% The erf function in matlab is different to the actual erf function
% erf((2-mu)/sigma)-erf((1-mu)/sigma)
% DO NOT USE!!!!!!!!!!!!!!!
```