CT216: Introduction to Communication Systems

Lab 3: Random Variables and the Probability Distribution Functions

In this Lab 3 exercise, you will perform statistical experiments and obtain a hands-on understanding of the various type of probability distributions - both discrete and continuous. You will experiment with the additive white Gaussian noise (AWGN) model of the impairments affecting the communication channel. Most of the programs are already given to you as a part of this Matlab Live Editor and you will be experiementing with them.

The experiments you will be performing in this Lab are also called ``simulations". This is an English word whose meaning, roughly, is ``recreating a real-world situation by modeling it in software (or sometimes even in hardware)". You will perform what is called the Monte Carlo simulation.

The Monte-Carlo method is a numerical simulation method for statistical parameter evaluation. This was invented by Stanislaw Ulam. The following are his words (see also this Wikepdia page.)

The first thoughts and attempts I made to practice [the Monte Carlo Method] were suggested by a question which occurred to me in 1946 as I was convalescing from an illness and playing solitaires. The question was what are the chances that a Canfield solitaire laid out with 52 cards will come out successfully? After spending a lot of time trying to estimate them by pure combinatorial calculations, I wondered whether a more practical method than ``abstract thinking" might not be to lay it out say one hundred times and simply observe and count the number of successful plays. This was already possible to envisage with the beginning of the new era of fast computers, and I immediately thought of problems of neutron diffusion and other questions of mathematical physics, and more generally how to change processes described by certain differential equations into an equivalent form interpretable as a succession of random operations. Later [in 1946], I described the idea to John von Neumann, and we began to plan actual calculations.

Simulation of Discrete Random Variables (RVs)

A Binary Information Source, i.e., the Bernouilli(q) RV

The output of any information-generating source can be modeled as the Bernoulli(q) RV provided the source output takes one of two values, i.e., the size of the source alphabet $K = 2$ (e.g., toss of a coin, the status of bodily health for a particular disease, the birth of a girl or a boy, whether it snows/rains on a particular day at a particular location, etc.). Here, q is the probability that the Bernoulli RV takes value of 1 (sometimes this q is denoted as p also; make sure the notation does not lead to a confusion), and this RV is characterized by

its probability mass function or PMF, which is given as $p_X(x) = [1-q, q]$, where the first element of this array corresponds to the probability that $X = 0$ and the second element denotes the probability that $X = 1$.

Step 1 Initialization

Begin by setting up the simulation parameters:

```
Nsim = 100 % Set the number of Monte Carlo simulations
```

```
Nsim = 100
```

```
q = 0.5% control the value of q by moving the slider
```

```
q = 0.5000
```

Step 2 Generation of the RV

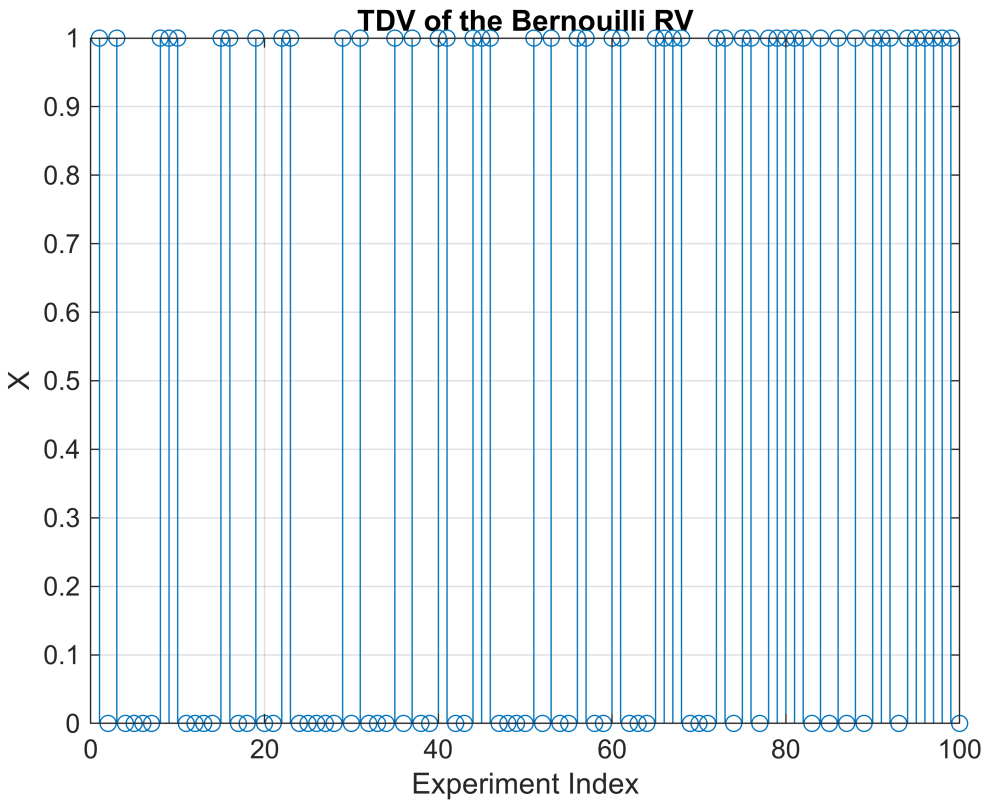
Use Matlab's rand function to generate N instances of the Bernouilli RV X. This is as if you did a series of N_{sim} experiments, where in each experiment you toss a coin. At the end you will have N_{sim} results (the outcome of each toss). The Matlab does this series of experiements much faster and we can easily control the property of the coin (the parameter q, which is called its biased-ness, $q = 0.5$ denotes an unbiased coin since it does not favor the outcome $X = 0$ versus $X = 1$). Also as a side note and on lighter vein, a real-world coin can possibly land on its edge where it somehow manages to perfectly balance itself (this is if we trust what is shown in an iconic Hindi movie) such that the outcome of the toss is neither 0 nor 1. We do not consider such possibilities in the following one line of simulation. Note that this is a succinct simulation code in Matlab. Make sure that you understand what Matlab does within this one line. If you wish, you can write your own program to do what this one line of code below does.

```
X = rand(1,Nsim)<q; % this is a quick way to generate Nsim realizations of
Bernouilli(q) RV X. Understand well what this one line of code does.
```

Step 3 Time Domain Visualization (TDV)

This shows us how the entire series of experiements turned out as a (time-domain) sequence.

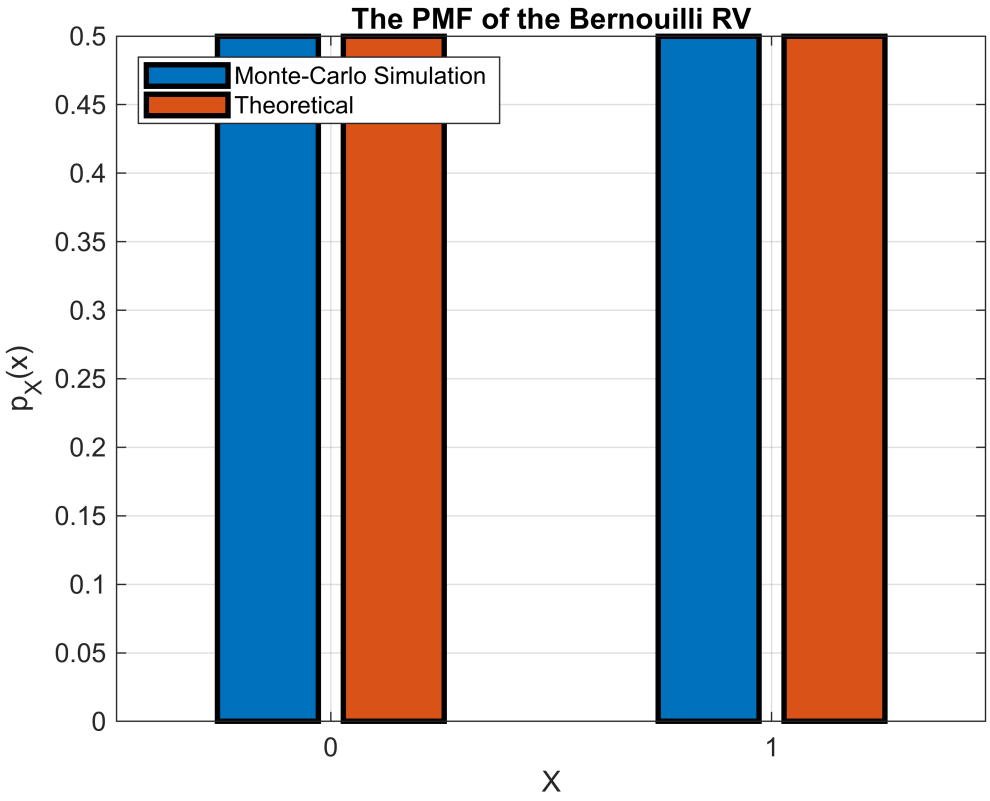
```
stem(1:Nsim,X(1:Nsim)); grid
xlabel('Experiment Index'); ylabel('X');
title('TDV of the Bernouilli RV')
```



Step 4 Statistical Evaluation

In this part, we will confirm that the generated RV X is indeed the Bernouilli RV with the specified parameter q. We will create two bins $X = 0$ and $X = 1$ and count the number of times X falls in each bin. This is called the histogram evaluation by Monte Carlo method. The Matlab has an inbuilt function called histcounts that does this (if you feel motivated, you can write your own histogram calculator function in Matlab instead). You will observe that the simulated PMF is in agreement with the theoretical PMF (does this agreement improve or worsen if N_{sim} is increased?).

```
xBins = [0 1];
xEdges =[-0.5 0.5 1.5] ;
pmf_simulated = histcounts(X,xEdges)/Nsim;
pmf_theoretical = [1-q q];

bar(xBins,[pmf_simulated; pmf_theoretical]','linewidth',2); grid;
xlabel('X'); ylabel('p_X(x)'); title('The PMF of the Bernouilli RV');
legend('Monte-Carlo Simulation','Theoretical','location','northwest');
```



Step 4 Continued, More Statistics

Evaluate the mean and the variance of the simulated RV and compare them against the theoretical expressions.

```
% the first element is the simulation result, the second is the theoretical value

xMeanSimulated_vs_theoretical = [mean(X) q]
```

```
xMeanSimulated_vs_theoretical = 1×2
    0.5000    0.5000
```

```
xVarianceSimulated_vs_theory = [var(X) q*(1-q)]
```

```
xVarianceSimulated_vs_theory = 1×2
    0.2525    0.2500
```

Step 5: Evaulate the entropy

The following is a simple Matlab code to evaluate the Entropy of the Bernouilli(q) random variable. This uses an entropy calculation function, entropy_function, that you will need to write yourself. Make sure that your function can accept any pmf_vector as an input (not just the PMF of the Bernouilli RVs).

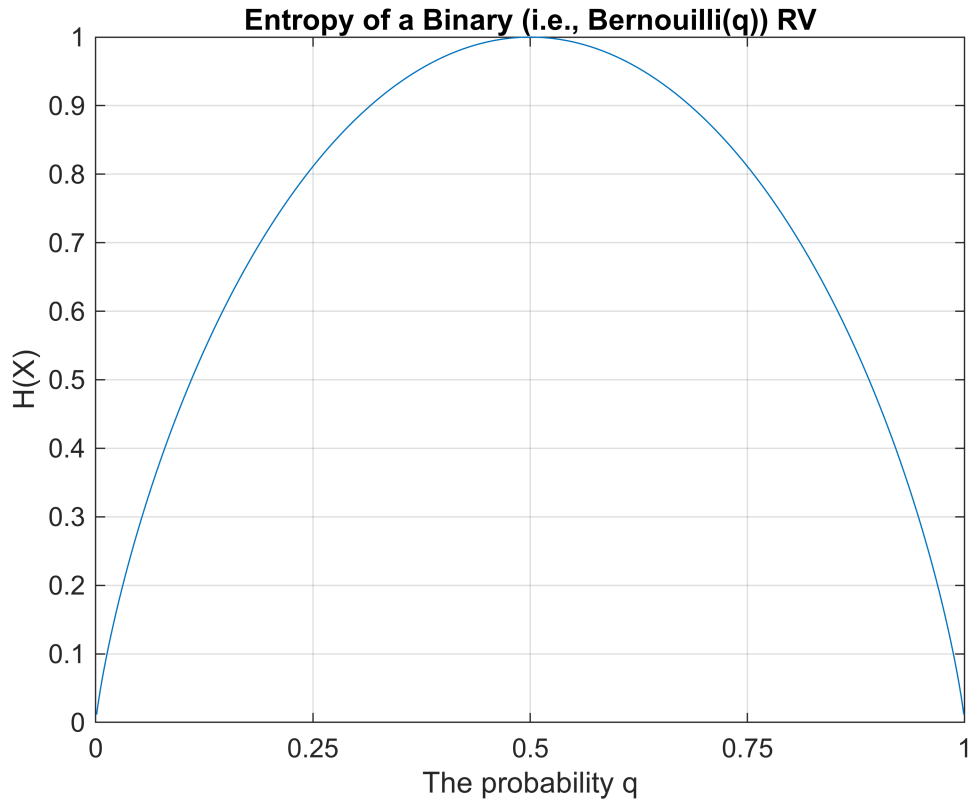
```
pmf_vector = [q 1-q];
Hx = entropy_function(pmf_vector); % entropy_function is a separate function that
you will need to write yourself
```

Step 6: Generate a plot of the entropy as q varies from 0 to 1

In this part of the code, we evaluate and plot the entropy by varying from 0 to 1. Here, we will first define a vector qVec that stores the entire range of values that q takes, and then we evaluate the entropy of each value of q in this range by running a FOR loop. We use the same entropy function inside the FOR loop. The output is stored in a vector Hvec so that it can be plotted as a function of qVec.

A programming note: it is a good habit to preallocate the memory for the arrays such as Hvec whose elements are evaluated one-by-one inside a FOR loop or a WHILE loop of Matlab. This is done below - a memory is created for Hvec by assigning it a vector of all zeros. These zeros are replaced by the calculated entropy value inside the FOR loop.

```
qVec = 0:0.001:1; % Vary the probability q that the Bernoulli RV takes value of 1
Hvec = zeros(1,length(qVec)); % Allocate the memory for the calculated entropy
values
kk = 0; % initialize an index of the FOR loop
for q = qVec
    kk = kk+1;
    pmf_vector = [q 1-q];
    Hvec(kk)=entropy_function(pmf_vector);
end
plot(qVec,Hvec); set(gca,'xtick',0:0.25:1); grid;
xlabel('The probability q'); ylabel('H(X)'); title('Entropy of a Binary (i.e.,
Bernouilli(q)) RV')
```



An $(N+1)$-ary Information Source: the Binomial (q, N) RV

The Binomial RV X is a non-binary but discrete-valued RV which finds many applications in the communication theory in specific and in many problems of the machine learning in general (there are, in fact, many real life problems that can be answered using the Binomial model).

This RV is defined as follows: $X = \sum_{n=1}^{N} Z_n$, where each Z_n , $1 \leq n \leq N$, is the Bernouilli(q) RV. One important requirement is that these N RVs which are summed to obtain X have to be independent random variables for X to be binomial. This independence assumption is maintained in the Monte Carlo simulations below. The Binomial RV is controlled by two parameters q and N, which, as you may have noticed, are specified in its name. The RV itself takes one of $N + 1$ values from the set $[0, 1, \ldots, N]$ (the first value occurs if all N RVs Z_n are zeros, the last value occurs if all of them are ones).

The PMF of this RV is given as $p_X(x = k) = \binom{N}{k} q^k (1 - q)^{N-k}$, where $0 \leq k \leq N$. Can you understand why this particular PMF formula is correct (why is the binomial coefficient multiplied with the powers of q and $1 - q$ in this specific way)? The expectation and variance of this RV are Nq and $Nq(1 - q)$. You will notice that there is a simple relation between these two expressions and the expressions of the mean and the variance of the Bernouilli RV.

Step 2 Generation of the RV

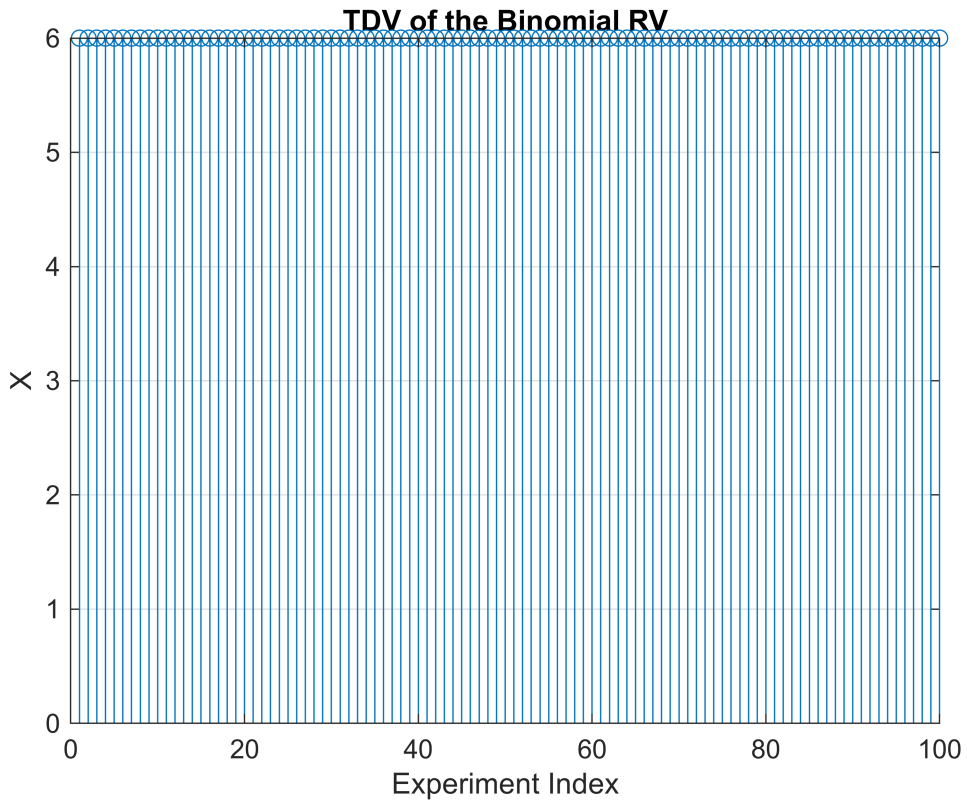
(make sure that you have executed Step 1 for the Bernouilli RV before you run this part of the program). There are multiple ways to generate the Binomial RV. One possibility is to run N_{sim} trials, and generate and add N Bernouilli RVs in each trial. This would require either a FOR or a WHILE loop. In Matlab, there is, however, a simpler way to do this, which avoids the looping (Matlab runs faster if you avoid the FOR or WHILE loops). Here, we generate a matrix of size $N \times N_{sim}$ whose elements are the Bernouilli RVs, and then we sum each column of this matrix. This will give us N_{sim} realizations of the Binomial RV. You should try alternative (e.g., FOR loop based) method of geenrating the Binomial RV. Matlab also provides an inbuilt function called Binornd. Once you know how to generate this RV yourself, you can directly use Matlab's inbuilt function.

```
N = 6;
Z = rand(N,Nsim)<q;
X = sum(Z); % Matlab sum function takes a sum of each column of the input matrix
```

Step 3 Time Domain Visualization (TDV)

This shows us how the entire series of experiements turns out as a (time-domain) sequence.

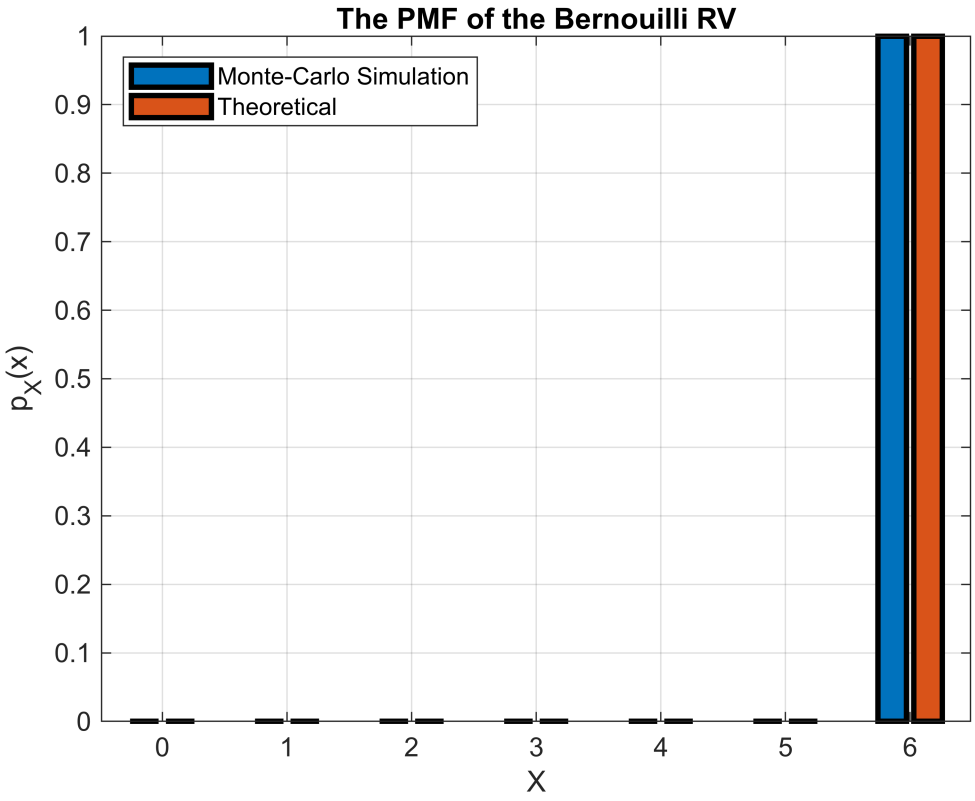
```
stem(1:Nsim,X(1:Nsim)); grid
xlabel('Experiment Index'); ylabel('X');
title('TDV of the Binomial RV')
```



Step 4 Statistical Evaluation

In this part, we will confirm that the generated RV X is indeed the Binomial RV. We will create $N+1$ bins from $X=0,1,\ldots,$ to $X=N$ and count the number of times X occurs in each bin. You will observe that the simulated PMF is in agreement with the theoretical PMF, which is obtained using Matlab's inbuilt binopdf function. You can consider writing your own function that calculates the Binomial PMF.

```
xBins = 0:N;
xEdges =[-0.5 xBins+0.5]; % understand why the edges of the histogram bins have to
be defined in this manner

pmf_simulated = histcounts(X,xEdges)/Nsim;

pmf_theoretical = binopdf(xBins,N,q);

bar(xBins,[pmf_simulated; pmf_theoretical]','linewidth',2); grid;
xlabel('X'); ylabel('p_X(x)'); title('The PMF of the Bernouilli RV');
legend('Monte-Carlo Simulation','Theoretical','location','northwest');
```



Step 4 Continued, More Statistics

Evaluate the mean and the variance of the simulated RV and compare them against the theoretical expressions.

```
% the first element is the simulation result, the second is the theoretical value

xMeanSimulated_vs_theoretical = [mean(X) N*q]
```

```
xMeanSimulated_vs_theoretical = 1×2
     6     6
```

```
xVarianceSimulated_vs_theory = [var(X) N*q*(1-q)]
```

```
xVarianceSimulated_vs_theory = 1×2
     0     0
```

A Non-Binary Information Source and Various Experiments

The following is a simple Matlab code to evaluate the Entropy of an information source that produces one of $K = 4$ symbols at a time.

- **A main assumption:** the PMF of this source is given as $p_X(x) = \left[1-q, \frac{q}{3}, \frac{q}{3}, \frac{q}{3}\right]$

- Note that the above assignment is required so as to make the entropy calculation dependent on a single parameter q that we can control. Also, the specific values $1-q$ and $q/3$ are needed to ensure that the sum of the elements of this PMF vector equals 1.

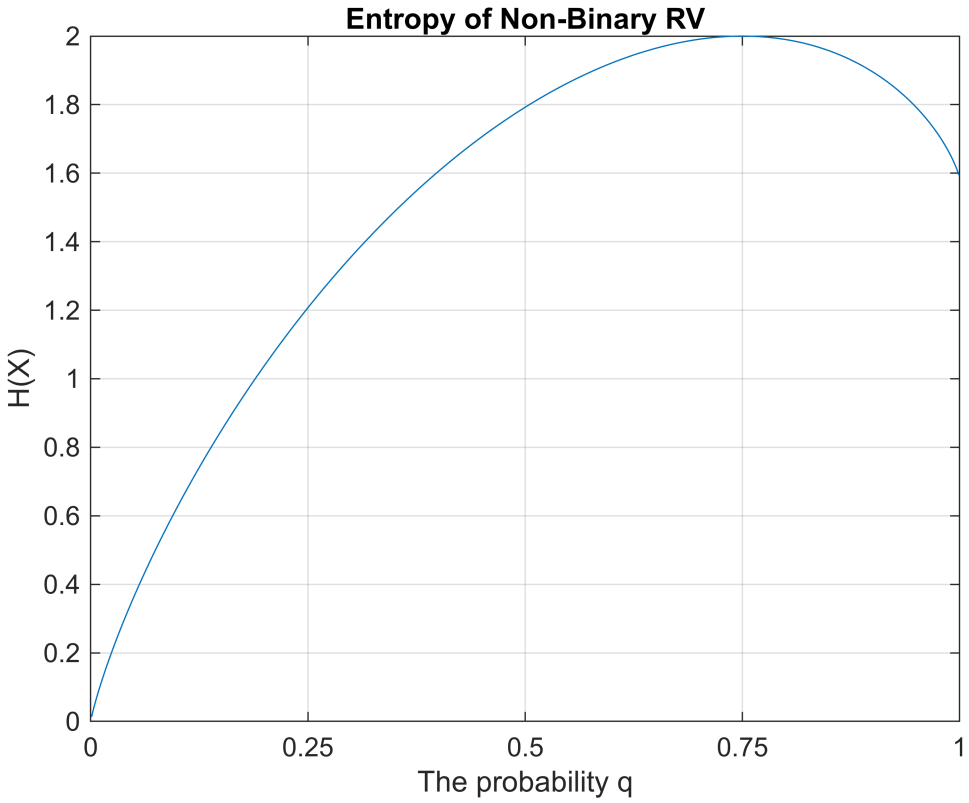
```
q =0.94;
pmf_vector = [1-q q/3 q/3 q/3];
H = entropy_function(pmf_vector)
```

```
H = 1.8173
```

Add your own work here

- Generate a plot of the entropy for the above non-binary information source, with $K=4$ as q varies from 0 to 1. When does the entropy get maximized?

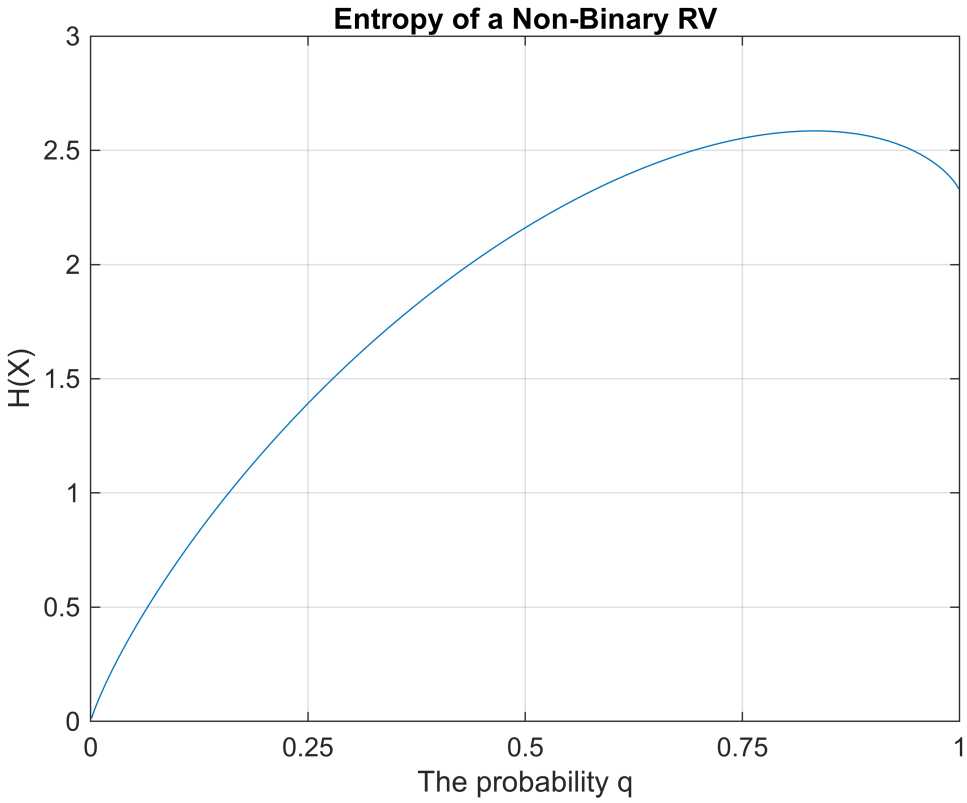
```
% Your code here
qVec = 0:0.001:1; % Vary the probability q that the Bernoulli RV takes value of 1
Hvec = zeros(1,length(qVec)); % Allocate the memory for the calculated entropy
values
kk = 0; % initialize an index of the FOR loop
for q = qVec
    kk = kk+1;
    pmf_vector = [q/3 1-q q/3 q/3];
    Hvec(kk)=entropy_function(pmf_vector);
end
plot(qVec,Hvec); set(gca,'xtick',0:0.25:1); grid;
xlabel('The probability q'); ylabel('H(X)'); title('Entropy of Non-Binary RV')
```




```
% For q=0.75 it gets maximized.
```

- Increase the source alphabet in the above code from $K = 4$ to a larger number and observe for what value of q does the entropy get maximized.

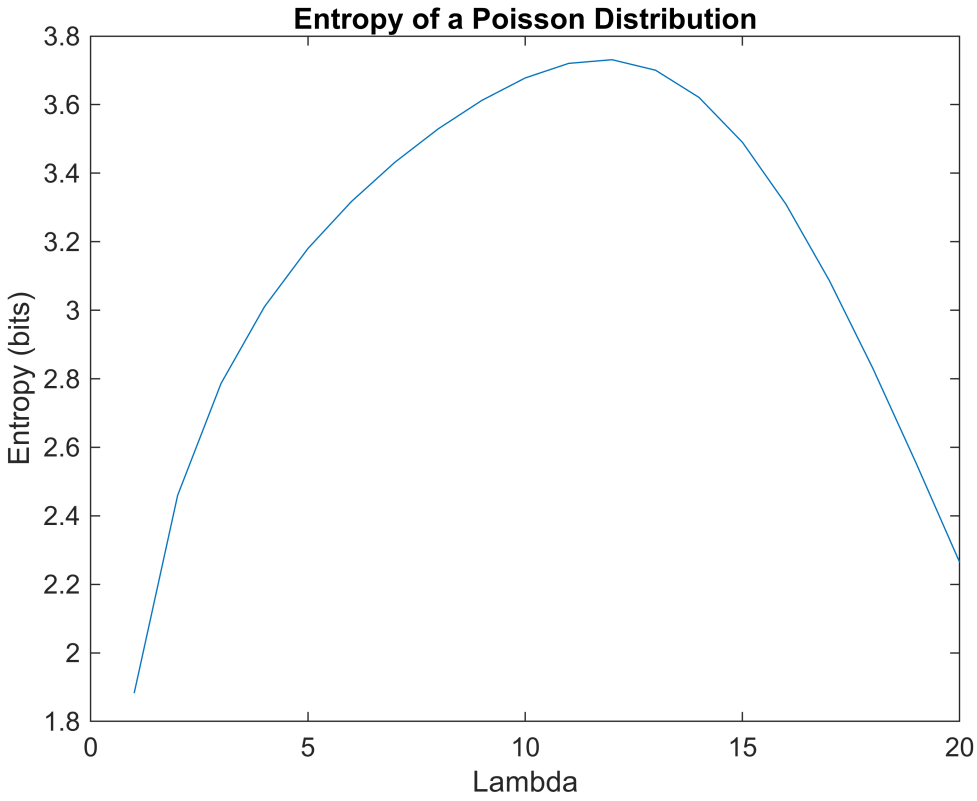
```
% Your code here
qVec = 0:0.001:1; % Vary the probability q that the Bernoulli RV takes value of 1
Hvec = zeros(1,length(qVec)); % Allocate the memory for the calculated entropy
values
kk = 0; % initialize an index of the FOR loop
for q = qVec
    kk = kk+1;
    pmf_vector = [q/5 q/5 q/5 q/5 q/5 1-q];
    Hvec(kk)=entropy_function(pmf_vector);
end
plot(qVec,Hvec); set(gca,'xtick',0:0.25:1); grid;
xlabel('The probability q'); ylabel('H(X)'); title('Entropy of a Non-Binary RV')
```




```
% For x=0.835(approx) and for K=6 it gets maximum.
```

- The binary source is uniquely characterized by the parameter q, i.e., the only way that the PMF can be defined is $p_X(x) = [q, 1-q]$. There is no single such parameter for a non-binary source. We have used above one, highly specialized, way to make the PMF a function of a single parameter q. There are obviously other ways. See if you can come up with a new way of modeling the PMF of a non-binary source with $K \geq 2$.
-

```
% Parameters
lambda = 5;  % Average rate of success
k_max = 20;  % Maximum number of successes
% Your code here
k = 0:k_max;  % Create a vector of values from 0 to k_max
pmf = poisspdf(k, lambda);  % Compute the PMF
% Compute the entropy for each value of lambda from 1 to k_max
H_values = arrayfun(@(lambda) poisson_entropy(lambda, k_max), 1:k_max);

% Plot the entropy
figure;
plot(1:k_max, H_values);
xlabel('Lambda');
ylabel('Entropy (bits)');
title('Entropy of a Poisson Distribution');
```




```
%{
function pmf = poisson_pmf(lambda, k_max)
    % Initialize the PMF vector
    pmf = zeros(1, k_max+1);

    % Compute the PMF for each value from 0 to k_max
    for k = 0:k_max
        pmf(k+1) = (lambda^k * exp(-lambda)) / factorial(k);
    end
end

function H = poisson_entropy(lambda, k_max)
    % Compute the PMF
    pmf = poisspdf(0:k_max, lambda);

    % Compute the entropy
    H = -sum(pmf .* log2(pmf + (pmf == 0)));
end
%}
```

- The entropy function obtains the average information generated by source. Write a function that lets you also observe the information I_k produced by each individual source output for a specific value of q.

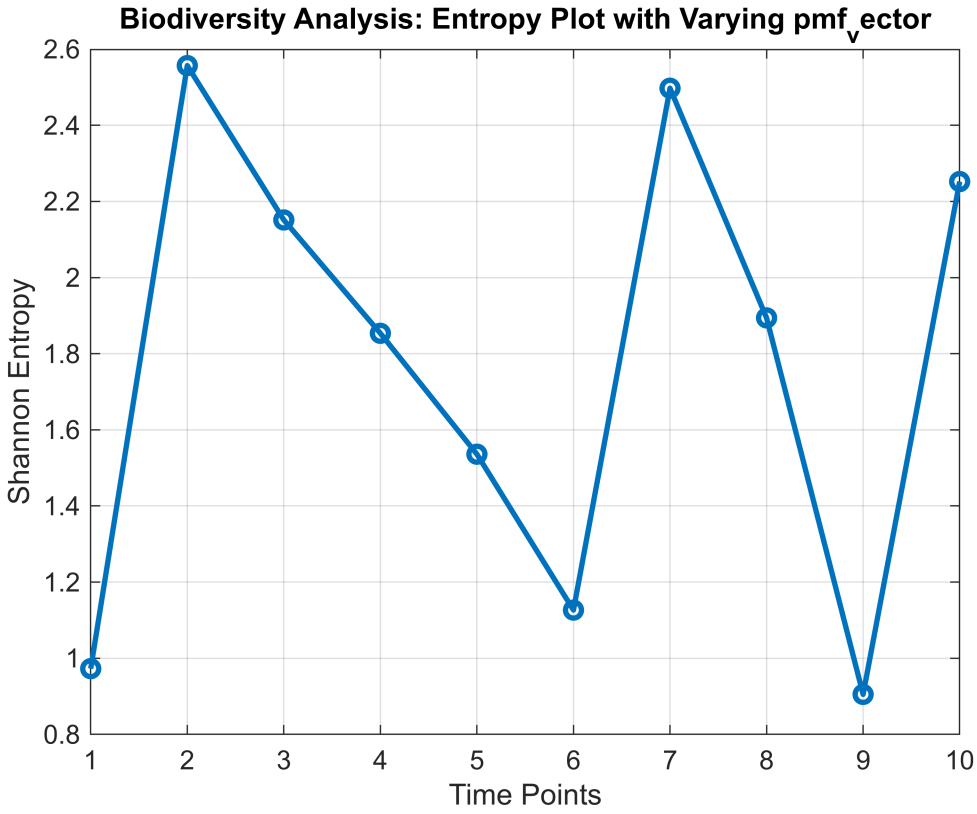
```
% Your code here
%{
```

```
function[ans] = value_of_ik(pmf_vector)
    ans=zeros(length(pmf_vector));
    for i=1:length(pmf_vector);
        ans(i)=-log2(pmf_vector(i));
    end
end
%}
```

- Think of some study, similar to the above, on your own and write your code to implement your study.

```
% Your code here
%{
The study of entropy for non-binary random variables has numerous applications
across various fields.
Physics and Thermodynamics:
Quantum Information Theory: In quantum mechanics, entropy measures are extended to
non-binary quantum systems. Entropy is used to study the information content and
correlations in quantum states.

Environmental Studies:

Biodiversity Analysis: Entropy measures can be applied to study the diversity and
distribution of species within ecosystems, considering non-binary data representing
various species.

%}
% Biodiversity Analysis: Entropy Plot with Varying pmf_vector

% Example data (replace with your own time-series data)
time_points = 1:10; % Time points
num_species = 5;    % Number of species

% Preallocate entropy array
entropies = zeros(1, length(time_points));

% Calculate entropy at each time point with varying pmf_vector
for t = 1:length(time_points)
    % Vary the probability q for each time point (replace with your own method)
    q = rand(); % Replace with your own method to get the desired probability q

    % Use a varying pmf_vector for each time point
    pmf_vector = [q / 5, q / 5, q / 5, q / 5, q / 5, 1 - q];

    % Calculate Shannon entropy at the current time point
    entropies(t) = -sum(pmf_vector .* log2(pmf_vector + eps));
end

% Plotting
figure;
plot(time_points, entropies, 'o-', 'LineWidth', 2);
```

```
xlabel('Time Points');
ylabel('Shannon Entropy');
title('Biodiversity Analysis: Entropy Plot with Varying pmf_vector');
grid on;
```



Simulation of Continuous Random Variables (RVs)

In this part, we will be studying the continuous RVs. These take analog values (i.e., the source alphabet is the entire real number line and its size is infinity) in contrast with the discrete RVs which take one value from a finite sized set. The PDF is a continuous-valued counterpart of the PMF.

Uniform Distribution

The Uniform Probability Distribution Function is used to model a Continuous (or Analog-valued) Random Variable (RV) X that takes values over a range $[a, b]$ with a probability density function or PDF $p_X(x) = 1/(b-a)$. An application of this distribution is in analyzing the effect of quantization noise that arises when the analog samples of a Discrete-Time (D-T) signals are quantized.

The expected or the mean value of this RV is given as $m_X = E[X] = \int_{x=a}^{b} x p_X(x) dX$. A simple integration will show to you that the mean of the uniform PDF is given as $(a+b)/2$. The variance of this RV, given as $Var[X] = E[X^2] - (m_X)^2$ is given as $(b-a)^2/12$. You should be able to derive this formula yourself, or you can see the derivation on Penn State Statistics Department website.

Step 1 Initialization

Begin by setting up the simulation parameters:

```
a = 0; b = 1; % change these values and observe the effect
```

Step 2 Modeling of the Source Output

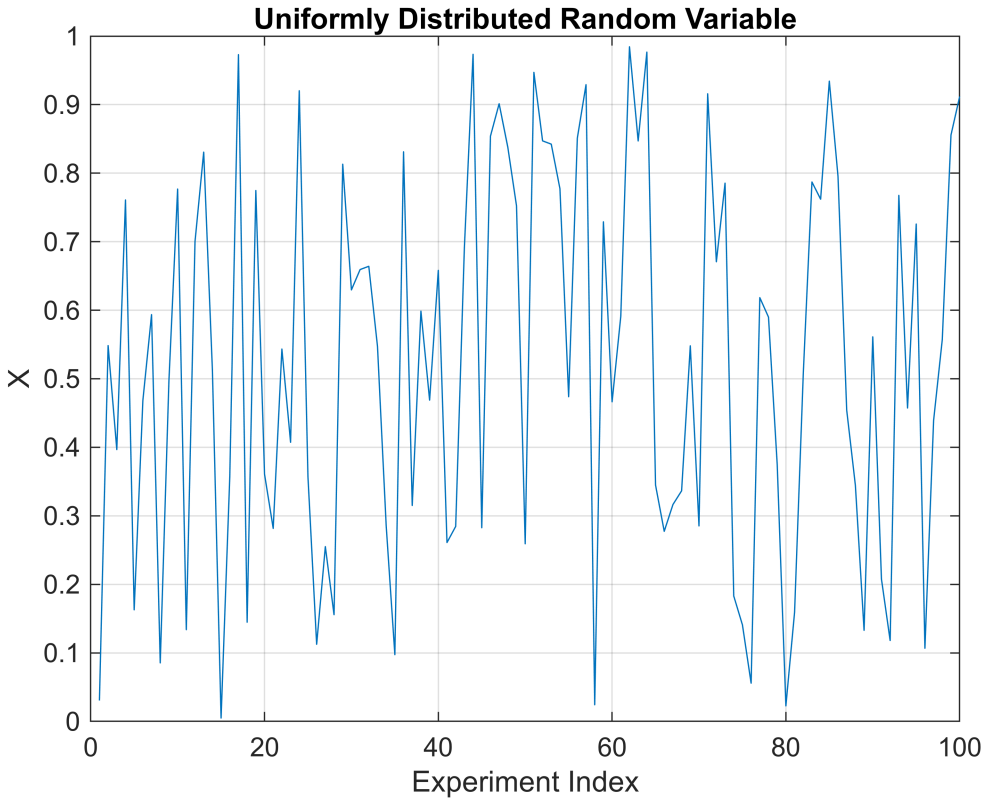
Use Matlab's rand function to generate N_{sim} instances of the uniform RV X, which is uniformly distributed over the specified range $[a, b]$.

```
X = a + (b-a)*rand(1,Nsim); % understand what this line of code does
```

Step 3 Time Domain Visualization (TDV)

See how the uniform random variable looks like if a sequence of them occurs one after another in the time.

```
plot(1:Nsim,X(1:Nsim)); grid
xlabel('Experiment Index'); ylabel('X');
title('Uniformly Distributed Random Variable')
```

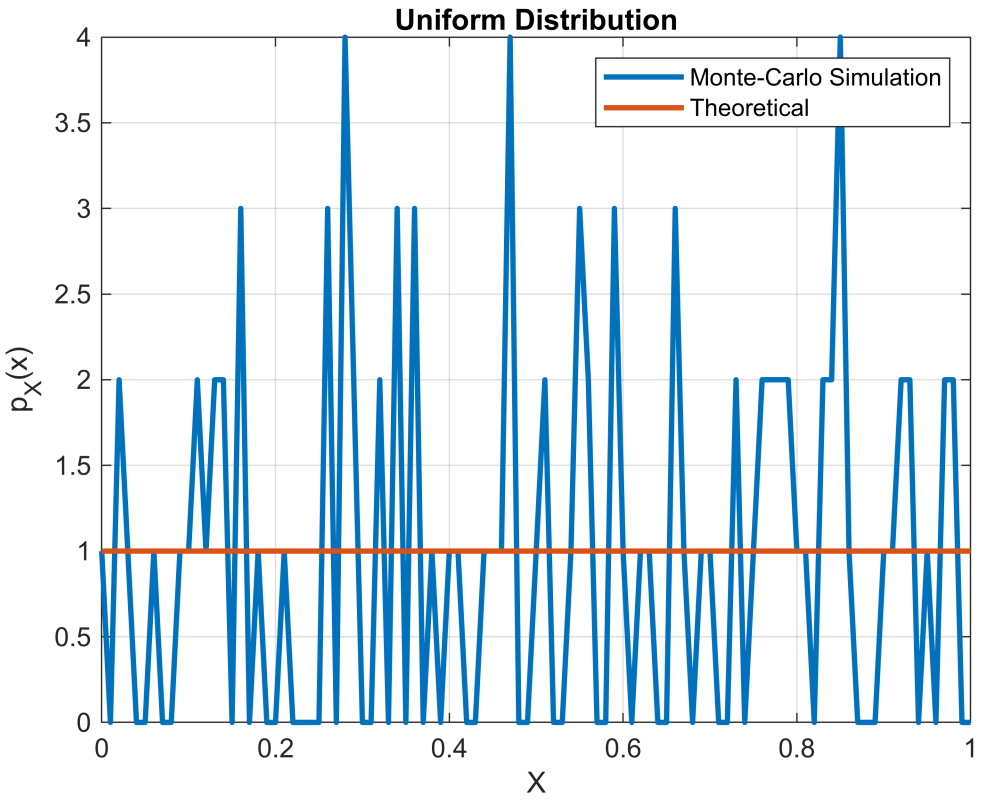


Step 4 Statistical Characterization

In this part, you will confirm that the generated RV X is indeed uniformly distributed. You will create multiple bins, specifically $(b-a)/stepsize$ number of bins in the range $[a, b]$ for the Monte Carlo histogram evaluation. You will observe that the frequency with which X falls in any one bin is roughly the same as it falls in any other

bin. As before, observe how well does the simulated PDF pdf_simulated compare against the theoretically-evaluated uniform PDF.

```
stepsize = 0.01;
xBins = a:stepsize:b;
xEdges = [a-stepsize/2 xBins+stepsize/2];

pdf_simulated = histcounts(X,xEdges)/stepsize/Nsim;

% Use Matlab's unifpdf function to generate the
% analytical (mathematical formula based) PDF.

% do a help search to know about Matlab's unifpdf function

pdf_theoretical = unifpdf(xBins,a,b);

plot(xBins,[pdf_simulated; pdf_theoretical],'linewidth',2); grid
xlabel('X'); ylabel('p_X(x)'); title('Uniform Distribution');
legend('Monte-Carlo Simulation','Theoretical');
```



Step 5 Statisics Continued

Evaluate the mean and the variance of the simulated RV and compare them against the theoretical expressions.

```
xMeanSimulated_vs_theoretical = [mean(X) (a+b)/2]
```

```
xMeanSimulated_vs_theoretical = 1×2
    0.5215    0.5000
```

```
xVarianceSimulated_vs_theory = [var(X) (b-a)^2/12]
```

```
xVarianceSimulated_vs_theory = 1×2
    0.0815    0.0833
```

The Gaussian RV

The last PDF we will study in this Lab is the Gaussian PDF. The Gaussian Distribution arises in many real-life scenarios, and it is famous (and known even to a layman) for its bell shape.

In the communication systems, the Gaussian PDF (also known as the Normal PDF and denoted as $N(\mu, \sigma)$, where μ is the mean and σ is the standard deviation) arises in modeling the probability distribution of the additive noise that affects the transmitted signal.

Repeat the above experiment for the Gaussian distributed Random Variables (RVs). Following changes are needed to the program listing given above.

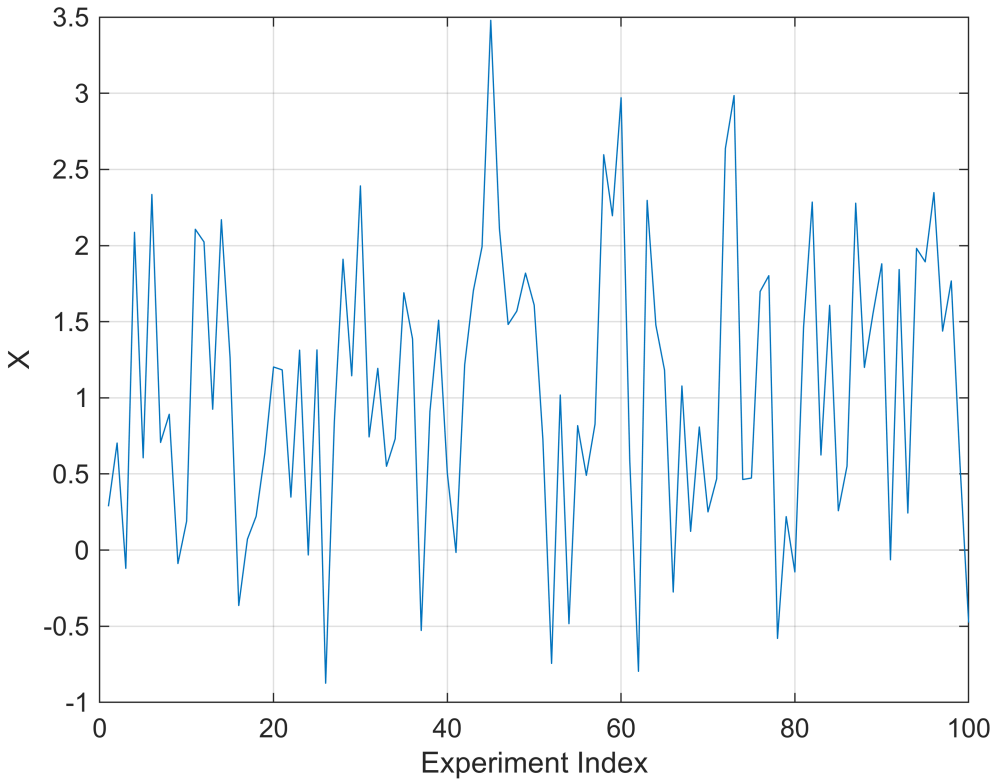
Steps 1 and 2 Initialization and Source Output Generation

The Gaussian distributed random variables $N(\mu, \sigma)$ can be generated (or simulated) as follows:

```
muGauss = 1; varGauss = 1; stdGauss = sqrt(varGauss);
X = stdGauss*randn(1,Nsim)+muGauss;
```

Step 3 TDV

```
% add code to do the TDV
plot(1:Nsim,X(1:Nsim)); grid
xlabel('Experiment Index'); ylabel('X');
```



Steps 4 and 5 Statistical Characterization

Evaluate the simulated PDF in the same manner as for the Uniform RV, except generate xBins based on the limits (or the end-points) of the generated variable.

```
stepsize = 0.01;
xBins = [muGauss-4*stdGauss:stepsize:muGauss+4*stdGauss];
xEdges = [xBins(1)-stepsize/2 xBins+stepsize/2];
```

Add your code below to compute the simulated PDF.

```
% add your code here
% add your code here
pdf_simulated = histcounts(X,xEdges)/stepsize/Nsim;
```

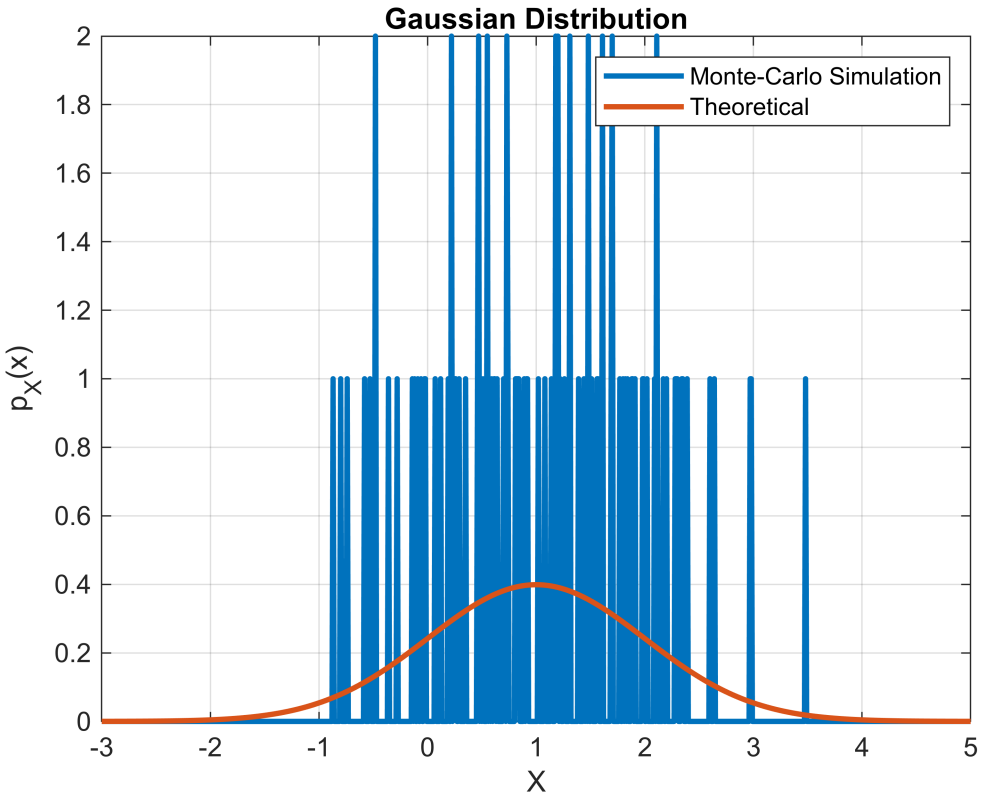
Use Matlab's normpdf function to generate the analytical (mathematical formula based) PDF (do a help search to know about Matlab's normpdf function).

```
pdf_theoretical = normpdf(xBins,muGauss,stdGauss);
```

Compare the results of your Monte-Carlo simulation against the theoretical results. Do this comparison for the PDF, its mean and the variance.

```
% add your code here
plot(xBins,[pdf_simulated; pdf_theoretical],'linewidth',2); grid
xlabel('X'); ylabel('p_X(x)'); title('Gaussian Distribution');
```

```
legend('Monte-Carlo Simulation','Theoretical');
```



CT216: Introduction to Communication Systems

Lab 4 (Experiments): Definition of Information and Information Transfer over a Communication Channel

The computer simulation based experiments in the Lab ask you to perform experiments to obtain understanding how much information can be transmitted reliably (i.e., without errors) even when the channel over which the information is transmitted is noisy. One of the main purposes of this Lab is for you to experiment with the values of q and p parameters of the C-disease problem, i.e., the BSC(p) channel, and observe and plot the resulting entropies numerically and graphically.

You are asked to perform the experiments using the following code (extra credits if you think of and devise new experiments and develop the Matlab/Python code). Show the results and make your observational/insightful comments regarding these results.

This entire programming-based experiments section of Lab 4 carries 40 marks.

A Model of Causes and Their Influence on the Effects

In this exercise we will consider the binary case, i.e., the cause is binary and the observed effect is also binary

Step 1: Define a Bernouilli Source

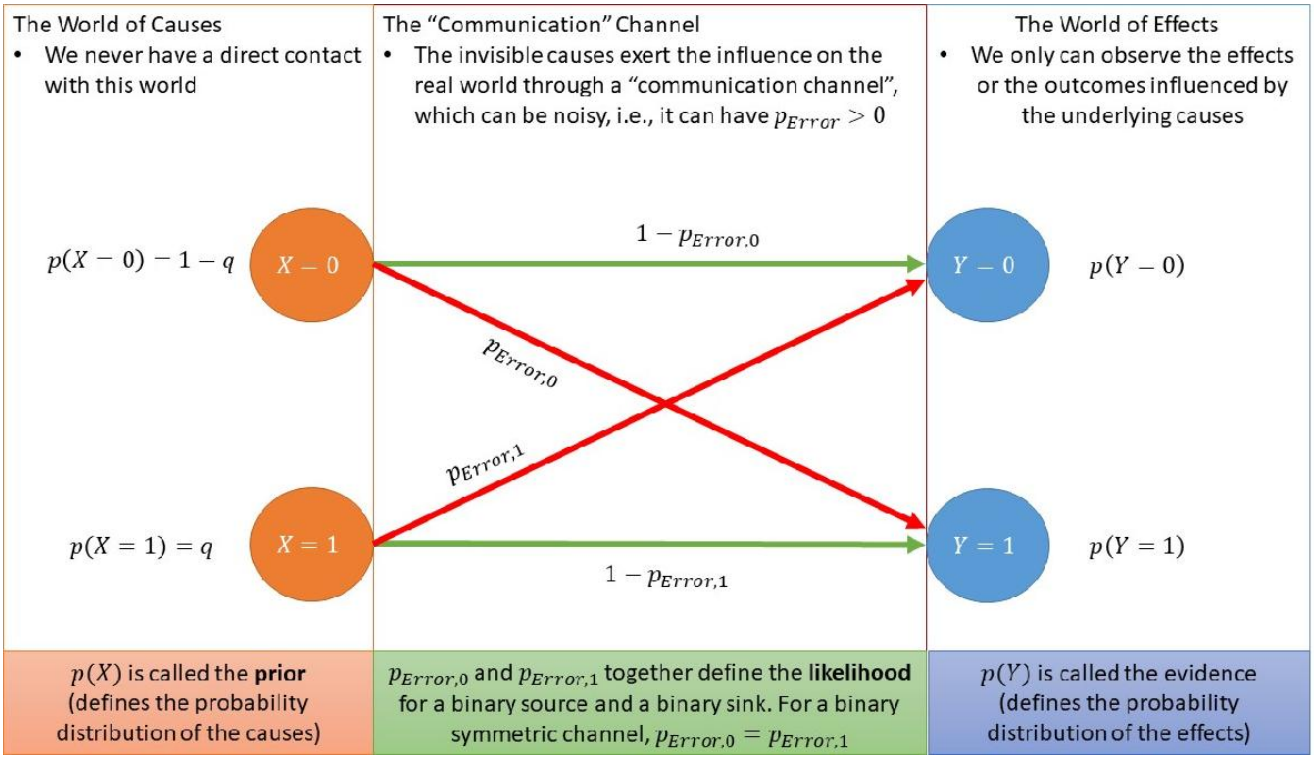
Define the Bernouilli(q) source X and calculate its entropy.

```
q =0.7;
pX = [q 1-q];
H_X = entropy_function(pX)
```

```
H_X = 0.8813
```

Step 2: Define a the channel and its likelihood function

Define the likelihood function of the binary noisy channel.



The first element of pYgivenX0 corresponds to $Y = 0$ and the second element corresponds to $Y = 1$, and likewise, the first element of pYgivenX1 corresponds to $Y = 0$ and the second element corresponds to $Y = 1$.

```
pError0 =0.21;
pYgivenX0 = [1-pError0 pError0]; % the likelihood function, i.e., the prob of [Y=0,
Y = 1] when X = 0

pError1 =0.42;
pYgivenX1 = [pError1 1-pError1]; % the likelihood function [Y=0, Y = 1] when X = 1
```

Step 3: Calculate the posterior on the source: the direct method

The (marginal) probability of the effect: i.e., the evidence or the probability $p(Y)$, is calculated using the rule of marginalization of the joint probability.

```
pY(1) = pYgivenX0(1)*pX(1)+pYgivenX1(1)*pX(2); % This is prob of Y = 0
pY(2) = pYgivenX0(2)*pX(1)+pYgivenX1(2)*pX(2); % This is prob of Y = 1
```

The posterior is calculated using the Bayes' rule:

```
pXgivenY0 = [pX(1)*pYgivenX0(1) pX(2)*pYgivenX1(1)]/pY(1)
```

```
pXgivenY0 = 1×2
    0.8426    0.1574
```

```
pXgivenY1 = [pX(1)*pYgivenX0(2) pX(2)*pYgivenX1(2)]/pY(2)
```

```
pXgivenY1 = 1×2
    0.5075    0.4925
```

Step 4: Calculate the Conditional Entropy and the Information Transfer

The entropy of the RVs $\widetilde{X}_n$, where $n = 0$ or 1, is called the conditional entropy (the entropy of X conditioned on an observation $Y = n$). This topic of conditional entropy is explained on **Slides 68 onward of Lecture 2 Probability Theory slides posted on Google Classroom**. Please read these slides before experimenting with the following code.

Note that $H(X)$ signifies the uncertainty at the receiver regarding the state $X = 0$ or $X = 1$ of the transmitter. The purpose of communication is to reduce this uncertainty at the receiver after observation of Y (e.g., we say that we have received information only if the output of the communication channel reduces our uncertainty about the state of the transmitter. Can you think of some real-world real-life examples of this). However, $H(\widetilde{X}_n)$ is the metric (a single scalar value) which succinctly captures the uncertainty at the receiver regarding X after observation of Y_n. Using this argument, Shannon defined the information transfer over a noisy communication channel as the *difference* between the unconditional and the conditional entropies. The specific name is *Mutual Information* and it is defined as $I(X;Y) = H(X) - H(\widetilde{X})$, where $H(\widetilde{X})$ is the average of $H(\widetilde{X}_n)$ over all values $Y = n$.

```
H_X
```

```
H_X = 0.8267
```

```
H_XgivenY0 = entropy_function(pXgivenY0)
```

```
H_XgivenY0 = 0.6280
```

```
InformationTransfer_givenY0 = H_X - H_XgivenY0
```

```
InformationTransfer_givenY0 = 0.1987
```

```

H_X
```

```
H_X = 0.8267
```

```
H_XgivenY1 = entropy_function(pXgivenY1)
```

```
H_XgivenY1 = 0.9998
```

```
InformationTransfer_givenY1 = H_X - H_XgivenY1
```

```
InformationTransfer_givenY1 = -0.1731
```

```

H_X
```

```
H_X = 0.8267
```

```
H_XgivenY = pY(1)*H_XgivenY0 + pY(2)*H_XgivenY1
```

```
H_XgivenY = 0.7419
```

```
InformationTransfer = H_X - H_XgivenY
```

```
InformationTransfer = 0.0849
```

Step 5: Evaluate the information transfer for varying values of $p_{Error} = p_{Error,0} = p_{Error,1}$

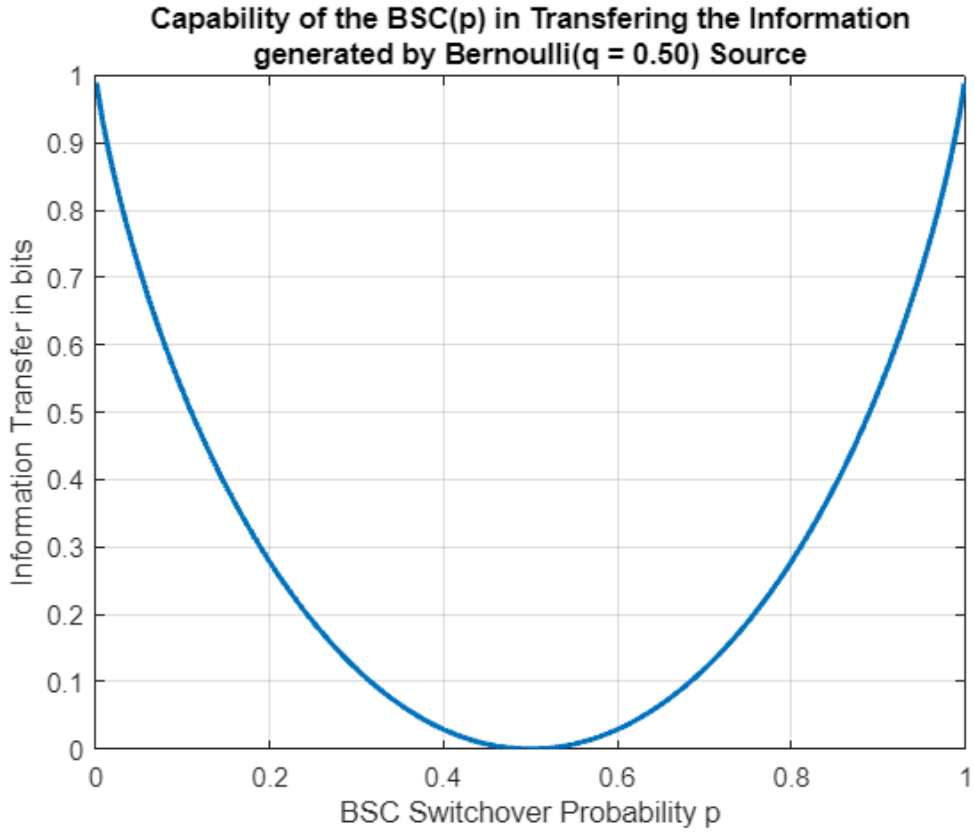
This step is similar to what we have done earlier for the Entropy function. There we varied q parameter and evaluated the Entropy of the Information source for each value of q. The evaluated Entropy was plotted as a function of q.

In this section, we now repeat the same procedure, except the goal is to plot the information transfer capability of the BSC(p_{Error}) in transferring the information generated by the Bernouilli(q) source. This plot is obtained as a function of p_{Error} for a specified q value.

```
q =0.5;
pX = [1-q q];
H_X = entropy_function(pX)
```

```
H_X = 1
```

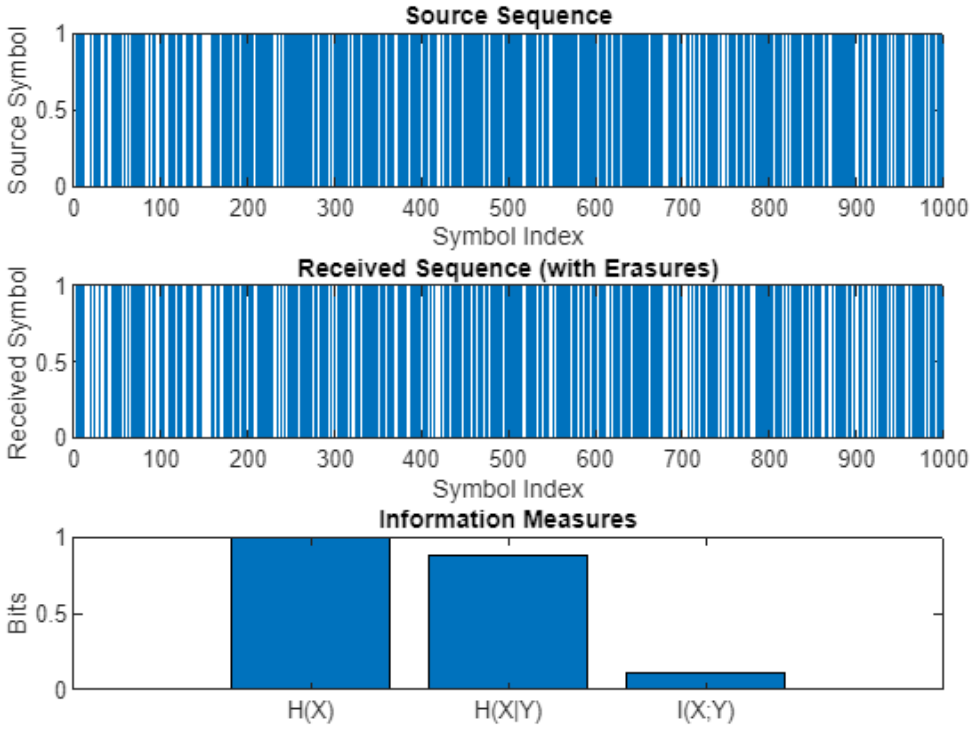
```
pErrorVec = 0:1e-3:1;
Info_X_Y = 0*pErrorVec;
kk = 0;
for pError = pErrorVec
    kk = kk+1;
    pYgivenX0 = [1-pError pError];
    pYgivenX1 = [pError 1-pError];
    pY(1) = pYgivenX0(1)*pX(1)+pYgivenX1(1)*pX(2);
    pY(2) = pYgivenX0(2)*pX(1)+pYgivenX1(2)*pX(2);
    pXgivenY0 = [pX(1)*pYgivenX0(1) pX(2)*pYgivenX1(1)]/pY(1);
    pXgivenY1 = [pX(1)*pYgivenX0(2) pX(2)*pYgivenX1(2)]/pY(2);
    H_X_givenY0 = entropy_function(pXgivenY0);
    H_X_givenY1 = entropy_function(pXgivenY1);
    H_X_givenY = pY(1)*H_X_givenY0 + pY(2)*H_X_givenY1;
    Info_X_Y(kk) = H_X - H_X_givenY;
end
plot(pErrorVec,Info_X_Y,linewidth=2)
xlabel('BSC Switchover Probability p'); ylabel('Information Transfer in bits')
title({'Capability of the BSC(p) in Transfering the Information',sprintf('generated
by Bernoulli(q = %1.2f) Source',q)}); grid
```




```
%{
In this section we can play with the values of p and q and perror and
qerror and can realise the graph with the analytic result that we got from
the analytical questions.
%}
```

```
% Binary Erasure Channel Simulation and Information Measures

% Parameters
p_erasure = 0.1; % Probability of erasure

% Generate random binary source
source_length = 1000;
source = randi([0, 1], 1, source_length);

% Simulate Binary Erasure Channel
received = source;
erasure_indices = rand(1, source_length) < p_erasure;
received(erasure_indices) = NaN; % Erasure symbol

% Calculate Entropy H(X)
prob_source_0 = sum(source == 0) / source_length;
prob_source_1 = sum(source == 1) / source_length;
```

```
entropy_X = - (prob_source_0 * log2(prob_source_0 + eps) + prob_source_1 *
log2(prob_source_1 + eps));

% Calculate Conditional Entropy H(X|Y)
prob_erasure = sum(~isnan(received)) / source_length;
conditional_entropy_X_given_Y = prob_erasure;

% Calculate Mutual Information I(X;Y)
mutual_information = 1 - prob_erasure;

% Plotting
figure;

subplot(3, 1, 1);
stem(source, 'Marker', 'none');
title('Source Sequence');
xlabel('Symbol Index');
ylabel('Source Symbol');

subplot(3, 1, 2);
stem(received, 'Marker', 'none');
title('Received Sequence (with Erasures)');
xlabel('Symbol Index');
ylabel('Received Symbol');

subplot(3, 1, 3);
bar([entropy_X, conditional_entropy_X_given_Y, mutual_information]);
xticks(1:3);
xticklabels({'H(X)', 'H(X|Y)', 'I(X;Y)'});
title('Information Measures');
ylabel('Bits');
```




```
% Display values
fprintf('Entropy H(X): %.4f bits\n', entropy_X);
```

```
Entropy H(X): 0.9992 bits
```

```
fprintf('Conditional Entropy H(X|Y): %.4f bits\n', conditional_entropy_X_given_Y);
```

```
Conditional Entropy H(X|Y): 0.8840 bits
```

```
fprintf('Mutual Information I(X;Y): %.4f bits\n', mutual_information);
```

```
Mutual Information I(X;Y): 0.1160 bits
```

CT216 (20220142G)

1. (Using given values)

(a) $P(Y=1) = P(Y=1|X=0)P(X=0) + P(Y=1|X=1)P(X=1)$

$= p(1-q) + (1-p)q = \boxed{p+q-2pq}$

(b) $P(X=1|Y=0) = \tilde{q}_0 = \tilde{x}_0$

$= \frac{P(X=1 \cap Y=0)}{P(Y=0)} = \frac{P(Y=0|X=1)P(X=1)}{P(Y=0)}$

$= \frac{pq}{1-p-p+2pq}$

$\Rightarrow$ Note $\rightarrow$ $P(Y=0) = P(Y=0|X=0)P(X=0) + P(Y=0|X=1)P(X=1)$

$= (1-p)(1-q) + p \cdot q = 1-p-q+2pq$

(c) $P(X=1|Y=1) = \tilde{q}_1 = \tilde{x}_1$

$= \frac{P(Y=1|X=1)\,P(X=1)}{P(Y=1)} = \frac{(1-p)q}{p+q-2pq}$

2. $\lambda_0 = \frac{\tilde{q}_0}{1-\tilde{q}_0}$

$\therefore \lambda_0 = \frac{pq}{(1-p)(1-q)}$

$\lambda_1 = \frac{\tilde{q}_1}{1-\tilde{q}_1}$

$= \frac{(1-p)q}{p(1-q)}$

3.

(a) $q = 0.01$ & $p = 0.05$

$\tilde{q}_1 = \frac{19}{118}$

$\lambda_1 = \frac{19}{99}$

} Using above solved equations

(b) $q = 0.01$ & $p = 0$

$\tilde{q}_1 = 1$

$\lambda_1 = \infty$

(c) $q = 0.5$ & $p = 0.1$

$\tilde{q}_1 = \frac{9}{10}$

$\lambda_1 = 9$

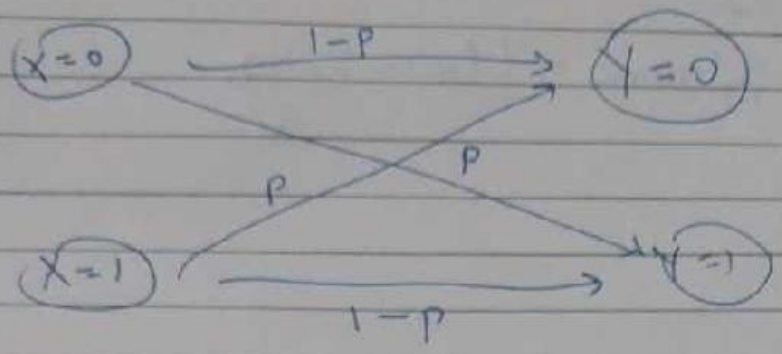
(d) $q = 0.1$ & $p = 0.5$

$\tilde{q}_1 = \frac{1}{10}$

$\lambda_1 = \frac{1}{9}$

5. X → bit by transmitter
Y → bit by receiver at output

(a)



X represent having disease (1=T; F=0)
Y represent test result (1=+ve; 0=-ve)

P → Test error → False +ve or False -ve

(b) Least Noisy → where $p=0$ → Highly reliable

Most Noisy → where $p=1/2$ → Spammer in work.

⇒ $p=0.05$

$q \to \tilde{q}_1$ facts $\frac{19-1}{118} \frac{100}{100} \to 15$ times $\lambda_1 = \frac{19}{99}$

$\frac{1}{1,00}$

Transmitter has strong influence on outcome Y.
→ Evidence Convincing for X.

⇒ $q \to \tilde{q}_1 \Rightarrow \frac{1-0.01}{0.01} = \frac{99}{1} \quad \lambda \to \infty$

Very strong ; Very Convincing

(3) $\frac{\frac{9}{10} - \frac{1}{2}}{\frac{1}{2}} = \frac{4}{5}$ & $1 - \frac{9}{10}$

Weak Influence & Unconvincing.

(4)

(a) Fairly accurate test + rare disease

+ve test increases our belief abt having the disease. We may need more tests to conform diagnosis.

(b) Perfect test & rare disease

→ We can be certain that we have the disease. No need for other test.

(c) Fairly Accurate + Common disease

+ve test increases our belief by large amnt. We can be fairly confident we have the disease. We would want to confirm w/ other test.

(d) Inaccurate test + uncommon disease

+ve Test barely increase belief abt having disease & not by meaningful amount

We can't trust the test.