CT542: Basics of Communication Systems

Lab 1

In Lab-1, you will perform statistical experiments that will provide you with a hands-on understanding of the various type of probability distributions. You will experiment with the additive white Gaussian noise (AWGN) model of the impairments affecting the communication channel. The last problem of the lab will give you an introduction to a basic, but realistic, model of a simple digital communication system, whose performance you will simulate.

In this lab, the experiments you will be performing are also called "simulations". This is an English word whose meaning, roughly, is "recreating a real-world situation by modeling it in either software (or sometimes even in hardware)". You will perform what is called Monte Carlo simulations. Do a web search to know about this method of Monte Carlo.

1. Simulation of various distributions:

   (a) *Uniform Distrbution.* Uniform PDF is used in analyzing the effect of quantization noise that arises when the analog samples of a Discrete-Time (D-T) signals are quantized. Use the following procedure to experiment with Uniformly Distributed random variables:

      i. Generate, in Matlab, $1 \times N_{experiments}$ vector of uniformly distrbuted variables over a range `[yMin,yMax]`:

      ```
      y = yMin + (yMax-yMin)*rand(1,Nexperiments);
      ```

      ii. Using Matlab's `histc` function, count the number of times the generated variable y falls in the different sub-regions of the overall range. For this, first divide the overall range of `[yMin,yMax]` into sub-regions of width `stepsize`.

      ```
      %Generate regions of size = stepsize in the range of
      %uniform variable
      stepsize = 0.01;
      yBins = yMin:stepsize:yMax;
      %Generate the boundaries, or edges, of these regions
      yEdges = [yMin-stepsize/2 yBins+stepsize/2];
      ```

      Next, use `histc` function to count the number of elements of y that fall in different regions defined by `yEdges`. This method of estimating the probability density function (PDF) of a probabilistic random variable is called *Monte Carlo* method.

      ```
      nY = histc(y,yEdges);
      %Divide nY by the number of experiments and stepsize
      %to obtain the probabilities
      pYsimulated = nY/stepsize/Nexperiments;
      ```

Always remember to use Matlab's `help functionName` feature (or do a Web search) to understand well what a particular command does. Another good way to understand the procedure well is to either plot the generated variables, or display them on Matlab command prompt.

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

iv. Evaluate the mean and the variance of the simulated random variable and compare against the theory.

v. Detailed code listing is provided below:

```
clearvars; close all;

Nexperiments = 100000;
yMin = 0; yMax = 1; %change these values and observe the effect
y = yMin + (yMax-yMin)*rand(1,Nexperiments);

% see how the uniform random variable looks like
% in the time domain
plot(1:1000,y(1:1000)); grid
xlabel('Sample Index'); ylabel('y');
title('Uniformly Distributed Random Variable (1000 samples)')

stepsize = 0.01;
yBins = yMin:stepsize:yMax;
yEdges = [yMin-stepsize/2 yBins+stepsize/2];

pYsimulated = histc(y,yEdges)/stepsize/Nexperiments;

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

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

pYtheoretical = unifpdf(yBins,yMin,yMax);


figure;
plot(yBins,[pYsimulated(1:end-1); pYtheoretical],'linewidth',2);
xlabel('y'); ylabel('pY(y)'); title('Uniform Distribution');
legend('Monte-Carlo Simulation','Theoretical');
grid;

% Evaluate the mean and the variance of the simulated
% random variable and compare against the theory

yMeanSimulated = mean(y);
yVarianceSimulated = var(y);
```

```
yMeanTheoretical = (yMin+yMax)/2;
yVarianceTheoretical = (yMax-yMin)^2/12;

% Display the simulated and analytical results
fprintf(1,'Mean value of uniformly distributed...
 random variable: %1.2f (simulated), %1.2f (theoretical)\n'...
,yMeanSimulated,yMeanTheoretical)
fprintf(1,'Variance of uniformly distributed...
 random variable: %1.2f (simulated), %1.2f (theoretical)\n',...
yVarianceSimulated,yVarianceTheoretical)
```

(b) *Gaussian Distribution.* Gaussian Distribution arises in many real-life scenarios, and is famous (and known to a layman) for its bell shape. In communication systems, 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.

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

```
muGauss = 1; varGauss = 1; stdGauss = sqrt(varGauss);
y = stdGauss*randn(1,Nexperiments)+muGauss;
```

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

```
pYtheoretical = normpdf(yBins,muGauss,stdGauss);
```

Here, generate `yBins` using the same method as outlined for the Uniform PDF after changing the limits (or the end-points) of the generated variables to the following:

```
yMin = muGauss-4*stdGauss; yMax = muGauss+4*stdGauss;
```

iii. Compare the results of your Monte-Carlo simulation against the theoretical results. Do this comparison for the PDF, its mean and the variance (whose theoretical values are `muGauss` and `varGauss`, respectively).

(c) *Binomial Distribution.* Binomial Distribution, unlike the Uniform and the Gaussian, is used to model discrete-valued RVs, and finds applications in many different contexts, among which the evaluation of the bit error probability is one.

i. Binomial RVs can be generated using the following procedure:

```
Nexperiments = 1000;

N = 10;
yBins = 0:N;
yEdges = [-0.5 yBins+0.5];
p = 0.1; % probability of a zero; (1-p) is the probability of a one

y = zeros(1,Nexperiments); %placeholder for binomial variable
for kk = 1:Nexperiments
```

```
    % in each simulation trial (or experiment), generate a binary
% string b of length N bits, where 1s and 0s occur with probability
% of (1-p) and p, respectively
    b = double(rand(1,N)>(1-p));

% Sum of the elements of string b is binomially distributed
    y(kk) = sum(b);
end
```

Here, we are generating a total of N bits and putting them in the vector `b`. The main line here is `b = double(rand(1,N)>(1-p))`. Do you understand what this line does?

The simulated random variable y in the next line is simply the summation of elements of b. Since b has N elements, each of which can be either 0 or 1, the minimum value of y is 0 (if all elements of b are zero), maximum value is N (when all elements of b are 1), and y can take any integer value within this range. That is why `yBins` is a vector of integers from 0 to N.

ii. Estimate the PDF of the simulated RV using the Monte-Carlo approach, and compare the simulated PDF against the thoeretical PDF. The latter can be generated using the following function:

```
pYtheoretical = binopdf(yBins,N,p);
```

iii. Recall (see class lecture slides) that the mean and the variance of Binomially Distributed RVs is Np and $Np(1-p)$, respectively. Compare the simulated and theoretical mean and variance values.

2. In the prior exercise, you've generated RVs that follow three different distributions, and performed Monte Carlo simulations to determine the PDF, mean and the variance of the generated random variables. In this exercise, you will determine the probability that the generated random variable is below a certain limit value `yLim`. Obviously, when `yLim < yMin`, your answer should be a probability of 0, and when `yLim > yMin`, you should get a probability of 1.

   (a) As an example, set `yLim` to be at 10% of the range (i.e., `yMax-yMin`) above `yMin`, i.e., as `yLim = yMin+0.1*(yMax-yMin)`. Next evaluate the desired probability as `pMonteCarlo = sum(y<yLim)/Nexperiments`.

   (b) Repeat the above step for different values of `yLim`, i.e., change `yLim` in steps of, e.g., 1% of the range from `yMin` to `yMax`, and evaluate the corresponding probability. The result is the Cumulative Distribution Function or CDF. Plot the CDF against `yLim`.

   (c) Compare the simulated CDF against the theoretical CDF for the three types of distributions. See the lecture slides for the closed-form expressions for the various CDFs. Note that Matlab has `qfunc` and `betainc` functions that will be useful.

3. Simulation of Binary Symmetric Channel BSC(p):

   (a) Generate a total of $N = 10000$ Bernoulli($q = 0.5$) RVs X at the transmitter, pass them through a BSC(p) channel, and use the Bayes' Rule to determine the transmitted bits X given the received bits Y. Evaluate the probability of bit errors in your simulator for each value of $p \in \{0.01, 0.1, 0.2\}$. Analytically determine the probability of bit errors for this BSC(p) channel and compare the simulated result with the analytical result.

(b) Repeat the above with repetition code. Pass the generated binary RV X at the transmitter through a repetition rate $r = 1/3$ channel code. The channel is BSC(p). Use the majority-vote decoder at the receiver. Evaluate the probability of bit errors in your simulator for $p \in \{0.01, 0.1, 0.2\}$. Analytically determine the probability of bit errors for this BSC(p) channel with $r = 1/3$ repetition code and compare the simulated result with the analytical result.