Indian Institute of Technology Bombay

Department of Electrical Engineering

Handout 16 EE 706 Communication Networks

Assignment 3 : **40 points** **Due date**: March 15, 2010

This assignment requires the use of Scilab which can be downloaded from `www.scilab.org`. The Sharada server already has Scilab installed. The solution code should be uploaded to Moodle. Don't email the code to the instructor.

1. Suppose we want to do a computer simulation of a coin-tossing experiment where the coin is unbiased, i.e. Pr[Heads] = Pr[Tails] = $\frac{1}{2}$. One way to do this is to use a routine which generates a uniform random variable (RV) in the interval $[0, 1]$. In Scilab (`www.scilab.org`), the routine which generates a uniform RV in the interval $[0, 1]$ is `rand()`. The basic idea is to generate a uniform RV in $[0, 1]$ and compare its value with 0.5. If the value is greater than 0.5 we declare that Heads was the result of the coin toss and declare Tails as the result otherwise. The reason this is a correct simulation of an unbiased coin is that a uniform RV in $[0, 1]$ is less than 0.5 with probability $\frac{1}{2}$ and greater than or equal to 0.5 with probability $\frac{1}{2}$. The following Scilab program simulates the coin-tossing experiment 10 times.

```
clear all;
numberOfRuns = 10;
for i=1:numberOfRuns
  if rand()<0.5
    printf("Heads\n");
  else
    printf("Tails\n");
  end
end
```

   Modify the above program to simulate an arbitrarily biased coin, i.e. a coin with Pr[Heads] = p and Pr[Tails] = $1 - p$ where $0 \leq p \leq 1$. [10 points]

2. Suppose we are given a biased coin but we do not know the probabilities Pr[Heads] = p and Pr[Tails] = $1 - p$. We would like to estimate these probabilities, i.e. estimate the value of p. Suppose X is a random variable which takes the value 1 when the coin shows Heads and the value 0 when the coin shows Tails. It is easy to see that the expected value of X is $E[X] = p$. Given this observation, we can use *the law of large numbers (LLN)* to estimate p. The law of large numbers says that if $X_1, X_2, \ldots, X_n$ are instances of the random variable X, then

$$\frac{X_1 + X_2 + \cdots + X_n}{n} \rightarrow E[X] \text{ as } n \rightarrow \infty$$

   The interpretation of the LLN is that the sample mean (the LHS) of a large number of instances will be close to the expected value of the random variable.

For the case of the biased coin, we can generate a large number of instances X_i by tossing it repeatedly and estimate p which is equal to $E[X]$ by taking the sample mean of the instances, i.e. by calculating the average number of Heads. This technique is called *Monte Carlo simulation*. The following Scilab program generates instances of the coin toss experiment and stores it in the variable `instances`.

```
clear all;
numberOfRuns = 100;
trueValueOfp = 0.75;
instances = bool2s([(rand(1,numberOfRuns)<trueValueOfp)]);
```

Modify the above program to estimate the value of p. Why is the estimate different from the true value? What can you do to improve the estimate? Put your answers as comments in the code you submit. [10 points]

3. **Stop-and-wait ARQ simulation:** Suppose we want to calculate the throughput of SW ARQ by simulation. Then we need to calculate the expected value of the time taken to communicate a frame X, i.e. we need to calculate $E[X]$. Once again we want to use the law of large numbers to estimate $E[X]$. In order to do so, we need to generate instances X_i, $i = 1, 2, \ldots, n$ of the time taken to communicate a frame. Once we have the instances we can estimate $E[X]$ as

$$E[X] \approx \frac{\sum_{i=1}^{n} X_i}{n}.$$

Generating these instances is the crucial step in the simulation. The following Scilab program is missing the code which will generate `numberOfRuns` instances and store their sum in the variable `sampleSum`.

```
clear all;
numberOfRuns = 100;
probFrameError = 0.1; // Probability of frame error
probAckError = 0.1;   // Probability of ACK error
roundTripTime = 3;    // Round trip time
timeoutDuration = 5;  // Duration of timeout
sampleSum = 0;
for i=1:numberOfRuns
    //
    // Insert your code here
    //
end
avgTimeTaken = sampleSum/numberOfRuns
theoreticalAvgTime = roundTripTime + timeoutDuration*(probFrameError +
(1-probFrameError)*probAckError)/((1-probFrameError)*(1-probAckError))
```

Insert the missing code. Compare the value of $E[X]$ calculated by the simulation (which is stored in the variable `avgTimeTaken`) to the theoretical value derived in the class notes which has been calculated and stored in the variable `theoreticalAvgTime`. Why is the estimate different from the true value? What can you do to improve the estimate? Put your answers as comments in the code you submit. [20 points]