CMPE 320/Spring 08/Project 1: Generating Random Numbers and Applications

Samir Chettri

February 18, 2008

Abstract

In class we will do much of the theory of probability and statistics. Throughout it we will assume that a source of random numbers with a given probability distribution exists. When polled, all students had, for example, heard about the “normal” distribution or ”bell–curve.” This density is ubiquitous in probability theory. Even more fundamental than that is a random number generator for uniform random variables. The uniform density is simply a box function extending along the x–axis in the range [0 1] and since area of a density must be unity, the height of this box is also one. This project investigates methods of generating such types of random numbers using MATLAB. Some of the programs to be generated are easy, while others are more difficult.

Students will first do some derivations followed by actual numerical calculations for filter design in MATLAB. They will hand in both their derivations and program that does the calculations. Upto fifteen percent of the points for the project will be given for derivations and upto eighty–five percent for the MATLAB programs. The maximum a student can get is one hundred points.

Keywords: MATLAB, uniform random numbers, chi–square test, other random distribution tests.

Contents

1 Uniform Random Variates **1**

2 Testing random number generators **2**

2.1 Graphical tests 2

2.2 The chi–squared test 2

3 What to do? **3**

1 Uniform Random Variates

Consider m, a large prime integer and a, a large fixed integer such that $a < m$. Random numbers can be easily generated using a very simple iterative equation -

$$x_{i+1} = (ax_i) \bmod m. \tag{1}$$

Here, “mod” refers to the remainder after we divide by m and x_0 is the initial seed with the property that $x \in [1, 2, \ldots, m-1]$.

Question 1: To understand how the above random number generator works, use $m = 31$, $x_0 = 1$ and $a = 13$. Here are the first three values of the sequence: $1, 13, 14, \ldots$. Create 64 random numbers starting with $x_0 = 1$. Repeat the calculation using $x_0 = 2$ with all other values remaining the same. Plot the resulting random numbers using MATLAB (there will be two graphs). Comment on the graphs with notes in the graphs itself. Hand in your graphs.

ht

Table 1: Parameters for two uniform random number generators that use Equation (1) as their basis

Parameter	Generator #1	Generator #2
a	65539	16807
m	2^{31}	$2^{31}-1$

Question 2: Starting with $x_1 = (ax_0) \bmod m$, show that

$$x_n = a^n x_0 \bmod m, n = 1, 2, 3, \ldots. \tag{2}$$

Hint: In order to show this, draw a number line and see the relationships between the quantities in equation (2). Also, rewrite "mod" using other mathematical operators. Hand in your derivations.

Let us consider two random number generators. Both random number generators use Equation (1) as their basis. The parameters of each random number generator are given in Table 1.

Question 3: For generator # 1 show that

$$x_{k+2} = (6x_{k+1} - 9x_k) \bmod 2^{31}, \text{for all } k. \tag{3}$$

Comment on what this relationship tells you regarding this random number generator. Do not just write an answer, provide some justification. Hand in your derivations and comments.

2 Testing random number generators

The big issue with random number generators is - are they truly random? At some level, the graph you drew in the previous section with $m = 31$, $x_0 = 1$ and $a = 13$ that had starting sequence $1, 13, 14, \ldots$, should convince you that these are not truly random, but pseudo–random. Accordingly, all our tests and comments below are for pseudo–random numbers.

2.1 Graphical tests

Some interesting things happen when you take three consecutive random numbers from generators 1 and 2 in Table 1 and use them as cartesian coordinates. Use your generators 1 and 2 (that will produce random numbers in [0 1]) to produce 10^6 random numbers each in $[-1\ 1]$, take three consecutive numbers and use them as X, Y, Z coordinates. Plot each triplet using the MATLAB function `plot3` so that the cube is filled with these points. Plot two volumes, one for each generator and use the interactive features of the graphics window in MATLAB to find an optimum viewing angle for both. Once you find an optimum viewing angle by trial and error, use `view` to force the plots to come to the particular azimuth and elevation view.

2.2 The chi–squared test

We have seen the chi–squared distribution in class as the sum of squares of normal distributions. This sub–section will elaborate on how chi-squared tests can be used to test the quality of random number generators. Remember, in this case we are working with uniform random number generators as defined by Equation 1, but the test we describe below will be applicable to non–uniform random number generators, e.g., normal distribution, gamma distribution etc.

The procedure is simple.

1. Generate a sample by calling the random number generator - in this case it is the uniform random number generator.
2. Compute a test statistic. In this case it is the chi–square test (to be defined shortly).
3. Assess the likelihood that the data generated by the random number generator is close to the ideal distribution.

Here is what the chi–square statistic looks like.

$$v = \sum_{x=0}^{k-1} \frac{(o[x] - e[x])^2}{e[x]}. \tag{4}$$

Intuitively, equation (4 gives us an idea of how close the observed distribution $o[x]$ is to the expected distribution $e[x]$. What is the interpretation of $o[x]$? $o[x]$ is a histogram of the generated random values with $k = 1000$ bins. Let n be the total number of random numbers generated by either method given in Table 1. $e[x]$ is easily calculated from our previous data.

We define a null hypothesis H_0 which postulates that the data $o[x]$ comes from a uniform distribution. The alternative hypothesis H_1 is that it is not from a uniform distribution. Now we would like $Pr[v_1^* \le v \le v_2^*] = 1 - \alpha$ where α is a small value and v_1^* and v_2^* represent lower and upper bounds on the value v can take. This means that if $v < v_1^*$ or $v > v_2^*$, the test failed and hypothesis H_0 is false. v_1^* represents the case that the histogram is too uniform while v_2^* is the case that it is far from uniform. Remember $v = 0$ means that the expected and observed counts are identical. In any random test we should be very surprised if the chi–squared test statistic is too low as well as if it is too high. If it is too low, it possibly means that the test was fudged. It is typical to choose $\alpha = 0.05$ which means that $Pr[v_1^* \le v \le v_2^*] = 0.95$. Through some calculations we do not show here, it is assumes that the chi–square "degreees–of–freedom," (DOF) are $k - 1$. The number of random numbers chosen are $n = 10k$.

How do we calculate either v_1^* or v_2^* given α? Consider v_2^*: It is the value of the chi–square distribution on the x–axis that has DOF $= k - 1$ and for which the probability (area) to the right of the graph is $\alpha/2$. Similar comments apply to v_1^*.

Remember, the chi–squared statistic is not the chi–squared distribution as defined in our texbook (Therrien and Tummala) on page 148, rather, equation (4) approaches the chi–squared distribution as n and k become very large.

3 What to do?

This project has two parts. The first part deals with performing derivations in order to understand the basic theory while the latter one deals with programming in MATLAB.

Derivations - 15%.

1. Answer Question 1, Question 2 and Question 3 in the text.
2. Show all derivations clearly and neatly. Do not skip any detail.
3. **Due date**: 26 February 2008, directly or by email to our TA (handwritten or scanned are both OK). Your submissions should be neatly written (or typed). If you are using tables or data from elsewhere, copy the data or mathematical fact rather than make a reference to it.
4. **Late submissions will not be accepted**.

MATLAB - 85%

- You will write several MATLAB programs and/or functions for random number generation and testing.

  1. Create two random number generator functions with the following interfaces: `r = randgen1(p,q)` and `r = randgen2(p,q)`. `randgen1`, `randgen2` are functions that create the random generators listed in Table 1. `p, q` are integers denoting size of the random array. For example `p = q = 1` indicates we will generate a single random number, while `p = 1000, q = 2` indicates that `r` is a random matrix with `1000` rows and `2` columns. **NOTE:** All random numbers generated by both functions should be in [0 1]. Clearly the parameters of each random generator listed in Table 1 will be within the routine itself. Consider carefully what you need to do if you have successive calls to the random number generator and how you will solve that problem in software.
  2. Write a program called `v = mychi(obs, exp, k, n)` . This will return the v value defined in equation (4).
  3. Write a main program called `Project01DriverProgram.m` that will set up and call the functions listed above. Your code will look as below:

```
% Test 1 - Graphical correlation between three consecutive points

  call first RG and scale rvs to be in [-1 1], use plot3 and view
  call first RG and scale rvs to be in [-1 1], use plot3 and view

% Test 2 - chi-square test

  Use alpha = 0.05 to compute critical values

  for i = 1:256

    generate 10^6 random numbers each using method 1 and 2
    calculate chi--square stat. v using k=1000, n=10000 with mychi()

  end

  plot two graphs, with v (and critical v) on y-axis and test # i on x-axis
  print total # of times each method failed, i.e., went beyond critical v.
```

- **Due date**: 11 March 2006 by email only to our TA.
- **Late submissions will not be accepted**.