PROJECT TITLE : UNIFORM RANDOM NUMBER GENERATORS & THEIR EVALUATIONS

[redacted]

ABSTRACT

This paper will basically explore a few commonly used uniform random number generators and will evaluate their generated results & performance on a variety of performance metrics to help us understand how well they perform.

BACKGROUND & DESCRIPTION

To start off with this report does not explore what is called True Random Number Generators, which requires sophisticated systems & can take a lot of time. Pseudo random numbers which have been explored here, are deterministic form of generators and thus can be replicated & behave close to random numbers.

I have structured this to first build out an algorithm to generate a set of pseudo-random numbers using the below algorithms and then based on the PRN set generated, I perform a set of independence and Goodness of fit tests on them based on 200 PRN's each. I have also measured the amount of time it takes for the generator to generate a set of 100,000 uniform PRN's.

LINEAR CONGRUENTIAL GENERATORS

This class of generators is the simplest form that take the form below and produces a set of numbers in a cyclic manner which, the higher it is the better. The range of integer bits for a 32-bit architecture can be at a max $M = 2^{31} - 1$ and numbers range from 0 – (M-1)

$$\mathrm{G(x) = (xa + b) \bmod M}$$

MULTIPLE RECURSIVE NUMBER GENERATOR

A multiple recursive generator (MRG) of order k is defined by the linear equation below. By choosing the parameters used in the equation wisely, the generator can be made to have a period of any magnitude.

$$x_n = (a_1 x_{n-1} + \cdots + a_k x_{n-k}) \bmod m;$$
$$u_n = x_n / m,$$

According to research papers online L'Ecuyer, Blouin and Couture (1993) performed an extensive search for good MRG's. one that they found was the below which has a period of

~ $2^{217.}$ This is also one of the generators I have implemented & tested out against the others.

$$z^{[k]} = 1{,}071{,}064 z^{[k-1]} + 2{,}113{,}664 z^{[k-7]} \ (\text{mod } (2^{31} - 19))$$

Now, the purpose of the chi-squared test is to check for goodness of fit of the generated PRN's. The test basically checks if the generated values fall evenly into the intervals given as (k) splits between [0,1].

The purpose of the runs test however, is to check for independence between the adjacent observations. Ideally there should not be too little or too many runs, but within a tolerance level which is defined by the alpha level (I have chosen 0.05 in both cases here).

I have attached the python code that I have used to generate the series of PRN's and have tried to put in comments wherever needed so it can be experimented with to generate different outputs.

What I observed was that, while we ideally need both the test cases to pass its not always the case. By playing around with some of the parameters I was able to find combinations which were independent but not uniform and vice versa. For instance, in the Multiple recursive generator implementation I have performed – I need 7 initial values to start off with and depending of what kind of values I choose [11,0,1,71,6,7,99] – it is independent but not uniform.