Random number generator [01M6F]

Dávid Ádám,
lecture Monday 12:45, summer term 2005/2006

June 4, 2006

Contents

1 Introduction

The aim of this project is to test the quality of two random number generators. First is the one implemented in standard library of Java programming language. The other is a "home-made" random number generator by the author.

1.1 The linear congruential method

For implementing author's home-made generator of random[1] numbers the following algorithm was chosen. We can obtain a sequence of random numbers X_n by:

$$\mathrm{X}_{n+1} = (a\mathrm{X}_n + c)\mathrm{mod}\, m \tag{1}$$

where

$n \geq 0$, m is *modulus*, $m > 0$

[1]The introduced sequence is *pseudorandom*, this means generating sequence with a certain period (Real random sequence would have ideally infinite period).

a means *multiplier*, $0 \leq a < m$,
c stands for *increment*, $0 \leq c < m$,
X_0 is the starting value, $0 \leq \mathrm{X} < m$.

2 Experiments

The author tried several methods to discover the quality of his random number generator, in the following subsections we introduce them. There were generated 333 random numbers from 0 to 1, each of them had 8 digits (e.g. 0,3141479).

2.1 χ^2 test

From statistics [1] we introduce:

$$V = \sum_{s=1}^{k} \frac{(Y_s - np_s)^2}{np_s} \tag{2}$$

where n is number of independent observations, in our case it's the count of random numbers we generated.

p_s means the probability of random number being in category s.

Y_s is the number of observations that *do* fall into category s.

If we take into consideration expansion $(Y_s - np_s)^2 = Y_s^2 - 2np_sY_s + n^2p_s^2$, and use the facts $Y_1 + Y_2 + \ldots + Y_k = n$ and $p_1 + p_2 + \ldots + p_k = 1$ we get from (2):

$$V = \frac{1}{n} \sum_{s=1}^{k} \left(\frac{Y_s^2}{p_s} \right) - n \tag{3}$$

2.1.1 Results with χ^2 method

Random numbers were sorted them to 9 categories with the step of 0,1 e.g. intervals $[0; 0, 1)$, $[0, 1; 0, 2) \ldots [0, 9; 1)$. V is counted according to (3) considering that $p_s = 1/9$ for every s, for more information see Section 4. The result for V counted with respect to the type of random-generator:

- home-made $V = 9, 72$
- Java's $V = 9, 47$

What is a reasonable value of V? The answer can be acquired from the following equation, left part is derived from (2).

$$\begin{aligned} V &> \chi^2_{1-\alpha}(k-1) \\ \frac{1}{n} \sum_{s=1}^{k} \left(\frac{Y_s^2}{p_s} \right) - n &> \chi^2_{1-\alpha}(k-1) \end{aligned} \tag{4}$$

Now it's time to find χ^2 in a statistical table[2].

[2] for example from [1]

$$\chi^2_{1-\alpha}(\nu) = \chi^2_{1-\alpha}(k-1) = \chi^2_{1-5\%}(9-1) = \chi^2_{0,95}(8) = 15,51 \tag{5}$$

where $\nu = k-1$, k represents the degrees of freedom. We've got $k = 9$ as *number of categories*. And at the and $\alpha = 5\%$ is the significance level.

2.1.2 Summary of χ^2 test

In both cases V is much more less ($V_{home} = 9,72$, $V_{java} = 9,47$) than $\chi^2_{0,95}(8) = 15,51$. So it is false that $V > \chi^2$, this implicates, that both of random generators generate fairly good random numbers.

We propose a hypothesis that output of our random generators is close to ideal, random and equally distributed. This hypothesis can be *accepted* but *cannot be confirmed*.

2.2 Serial correlation test

We use the following equation (6) derived in [1], it represents the *serial correlation coefficient*, the extent to which U_{j+1} depends on U_j.

$$C = \frac{n(U_0U_1 + U_1U_2 + \ldots + U_{n-2}U_{n-1} + U_{n-1}U_0) - (U_0 + U_1 + \ldots + U_{n-1})^2}{n(U_0^2 + U_1^2 + \ldots + U_{n-1}^2) - (U_0 + U_1 + \ldots + U_{n-1})^2} \tag{6}$$

Coefficient C lies between -1 and +1. When it's zero it means, that quantities U_j and V_j are relatively independent of each other. On the other hand a value of ± 1 indicates total dependence.

2.2.1 Results – serial correlation test

Results of random generators:

- home-made, $C \simeq 0.69737$
- Java's, $C \simeq 0.68332$

Now we can conclude, that our home-made generator is a not as good as Java's in terms of correlation between U_j, U_{j+1}.

3 Conclusion

Both of our tested algorithms were based on linear congruential formula defined at [1], Section 3.2.1. In two of the disciplines, Java's algorithm had better behaviour.

On the whole the two results were really tight to each other. My further plans would be to do a spectral test on these sequences, that would show, if there are some hyperplanes, that can show us nonrandom behaviour[3].

[3] (for further studies the author recommends [1] Section 3.3.4)

4 Appendix A.

4.1 Equalized distribution

One of typical behaviours of random sequences are, that they are randomly distributed. This means that there's a same probability that the next random number would be any of numbers from interval zero to one,in our case.

$$P[X = a_k] = \frac{1}{n}, k = 1, 2, \ldots, n. \tag{7}$$

$$EX = \frac{1}{n} \sum_{k=1}^{n} a_k \tag{8}$$

4.2 Java's generator source code

Here's a preview of Java's source code[4], that gathers the next random number.

```
protected int next(int bits) {
    long oldseed, nextseed;
    AtomicLong seed = this.seed;
    do {
oldseed = seed.get();
nextseed = (oldseed * multiplier + addend) & mask;
    } while (!seed.compareAndSet(oldseed, nextseed));
    return (int)(nextseed >>> (48 - bits));
}
```

References

[1] Knuth, D.: The art of computer programming (second edition, volume 2.), 1981, Addison-Wesley

[2] Rogalewicz, V.: Pravděpodobnost a statistika pro inženýri, 1998, CTU publishing house

[4] available after registration on *http://java.sun.com*