Random Numbers

STAT221

Random Number Generators

Most of the topics covered on this course require simulation of random numbers.
True random numbers can be 'generated' by running a physical experiment, which has random outcomes.
Computational algorithms can also be used to generate sequences of numbers which `behave similarly' to random numbers. Of course, any computer algorithm is deterministic, so that if you know the initial state and the calculations the algorithm undertakes then you would be able to exactly reproduce the exact sequence of so called random numbers. Therefore, they are not truely random as they are 100% predictable, so are called pseudo-random numbers.
The challenge is creating a deterministic algorithm that can mimic randomness, or more precisely can fool statistical tests of randomness, so that for any practical purpose the user can treat them as though they are random

Linear Congruential Number Generator

Generating a sequence of integers via the recurrence equation:

$$X_i = (aX_{i-1} + c) \mod m$$

for suitable choices of:

- a - the multiplier
- c - the increment
- m - the modulus
- X_0 - the seed or starting value

The integers produced by this equation are modulo m, which means they are between $0, \ldots, m-1$, so there are only m different numbers. We will see below that a good CRNG will uniformly cover all possible values $0, 1, \ldots, m-1$ over a sufficiently long run. Therefore, `realisations' of uniform random variables on the range $[0, 1)$ are produced by dividing the sequence of pseudo-random numbers by m:

$$U_i = \frac{X_i}{m} \quad \text{for} \quad i = 1, 2, \ldots$$

Though some implementations divide by $m-1$ to give a closed intervals $[0, 1]$, i.e. including 0 and 1. Further, some implementations give values in $1, 2, \ldots, m$, which if the divisor is m will give an interval of $(0, 1]$. For most practical purposes these slight variants are of little concern as m is usually chosen to be very large, so inclusion of the end points is irrelevant (individual values are irrelevant from a probabilistic viewpoint).

Example

$$X_0 = 1 \qquad m = 31 \qquad a = 3 \qquad c = 5$$

```r
X <- numeric(length=31)
X[1] <- 1
for (i in 2:length(X)){
  X[i] <- (3 * X[i-1] + 5) %% 31
}
X/31
```

```
##  [1] 0.03225806 0.25806452 0.93548387 0.96774194 0.06451613 0.35483871
##  [7] 0.22580645 0.83870968 0.67741935 0.19354839 0.74193548 0.38709677
## [13] 0.32258065 0.12903226 0.54838710 0.80645161 0.58064516 0.90322581
## [19] 0.87096774 0.77419355 0.48387097 0.61290323 0.00000000 0.16129032
## [25] 0.64516129 0.09677419 0.45161290 0.51612903 0.70967742 0.29032258
## [31] 0.03225806
```

Period of a CRNG

Due to the modulus operation CRNG' will always repeat themselves. A key property of such algorithms is their period, i.e. the time length at which they repeat. If they repeat quickly then this will make a poor set of random numbers.

The period of a CRNG is at most m. If such a CRNG goes through all the integers $0, 1, \ldots, m-1$ before repeating itself then it is described as a 'full cycle' generator. If two values in $\{X_0, X_1, \ldots, X_m\}$ are identical, then $X_i = X_{i+p}$ for some $p < m$ and so the generator will have a shorter period than length m. Ideally, you want the parameters to be chosen to maximise this period. It can be shown (not proven here) that if the increment $c \neq 0$ a CRNG will have the maximum period of length m for any seed value if and only if:

- c and m are relatively prime (i.e. the only integer common divisor of c and m is 1)
- $a - 1$ is divisible by all prime factors of m and
- $a - 1$ is a multiple of 4 if m is a multiple of 4

Due to their cyclical nature a neat way to think about CRNG's is that the sequence goes around a circular dial in one direction (usually drawn clockwise), with as many points as the period (i.e. upto a maximum of m points). If it is a full cycle CRNG, then the starting seed could be anywhere on the dial, but the CRNG will always follow the sequence from the seed in a clockwise direction.

Examples

Have a look at the following sequences:

$$X_0 = 1 \qquad m = 16 \qquad a = 5 \qquad c = 1$$

$$X_0 = 1 \qquad m = 16 \qquad a = 3 \qquad c = 1$$

$$X_0 = 2 \qquad m = 16 \qquad a = 3 \qquad c = 1$$

$$X_0 = 1 \qquad m = 2^{11} \qquad a = 1229 \qquad c = 1$$

$$X_0 = 1 \qquad m = 244944 \qquad a = 1597 \qquad c = 51749$$

Take a look at a histogram `hist()` and plot X_i vs X_{i+1} with `plot()` .

The really good properties of CNRG algorithms are that they require little memory (to obtain X_i, you only need to know a, c, m and previous value X_{i-1}) and are very fast. But the structure of the algorithm leads to dependence/correlation between values separated by certain lags in time (called serial correlation).

They should also not be used for cryptography (e.g. internet encryption protocols) as it is easily broken by standard frequency analysis (i.e. it is easy to determine a, c and m). One good thing about computer based random number generators in general is that they are reproducible. So provided you record the seed (and the algorithm parameters) you can always reproduce exactly the same sequence of random numbers.

Reproducibility is a key concept in scientific exploration.

Alternative RNGs

Combinations of multiple runs of this random number generator, with different choices of these parameters are commonly combined to provide better performance (giving the so called combined linear congruential random number generation).

One of the best random number generators which is used as the default in many computer packages (including R) is the Mersenne Twister (Matsumoto and Nishimura, 1998, see Wikipedia (https://en.wikipedia.org/wiki/Mersenne_Twister)), which uses the binary representation of numbers on computers but is beyond the scope of this course.

Many implementations in computer packages (like R) have sensible default values for the parameters and set the seed depending on the system date/time. So in order to produce reproducible results, users often just store the initial seed and keep the default parameter settings.

```
# generate 5 uniform random numbers (Mersenne-Twister)
set.seed(42)
runif(5, min=0, max=1)
```

```
## [1] 0.9148060 0.9370754 0.2861395 0.8304476 0.6417455
```

```
# change the RNG
RNGkind("Knuth-TAOCP-2002")
set.seed(42)
runif(5, min=0, max=1)
```

```
## [1] 0.1276630 0.5334160 0.4485457 0.1315829 0.6034311
```

True Random Numbers in R

The random package provides several functions that access the true random number service at http://random.org (http://random.org). The http://random.org (http://random.org) services uses atmospheric noise sample via a radio tuned to an unused broadcast frequency together with a skew correction originally due to John von Neumann.

```
library(random)
# generating 10 random integers between 1 and 100
randomNumbers(n=10, min=1, max=100, col=10, base=10)
```

```
##      V1 V2 V3 V4 V5 V6 V7 V8 V9 V10
## [1,] 10 24 28 29  6  4 36 65 40  45
```