

## Class 12. Random Numbers

- *NRiC* §7.
- Frequently needed to generate initial conditions.
- Often used to solve problems statistically.
- How can a computer generate a random number?
  - It can't! Generators are *pseudo-random*.
  - Generators are *deterministic*: it's always possible to produce the same sequence over and over.
  - Sometimes this is a good thing!

### Random Number Generators

- User specifies an initial value, or *seed*.
- Initializing generator with same seed gives same sequence of “random” numbers.
- For a different sequence, use a different seed.
- One strategy is to use the current time, or the processor ID, to seed the generator.
  - Problem: this may have poor dynamic range, or may be correlated with when the code is run.
  - Solution: *combine* sources, e.g., `int seed = (int) time(NULL) % getpid() + getpid()`, to get a more robust seed.

### Choosing a Generator

- Since generators do not produce truly random sequences, it's possible that your results may be affected by the generator used!
- Often the supplied generators on a given machine have poor statistical properties.
- But even a statistically sound generator can still be inadequate for a particular application.
- Be wary if you ever need more than  $\sim 10^6$  random numbers, and certainly if you need more than the largest representable integer!
- Solution: always compare results using two generators.

## Guidelines

- Follow these steps to minimize problems:
  1. Always remember to seed the generator before using it (discarding any returned value).
  2. Use seeds that are “somewhat random,” i.e., have a good mixture of bits, e.g., 2731771 or 10293085 instead of 1 or 4096 or some other power of 2.
  3. Avoid sequential seeds: they may cause correlations.
  4. Compare results using at least two generators.
  5. When publishing, indicate generator used.
  6. Often it’s a good idea to make a note of the seed used for a given run, in case you need to regenerate the sequence again later.

## Uniform Deviates

- Random numbers that lie within a specified range (typically 0 to 1), with any one number in the range as likely as any other, are *uniform deviates*, i.e.,

$$p(x) dx = \begin{cases} dx & \text{if } 0 \leq x \leq 1, \\ 0 & \text{otherwise.} \end{cases}$$

- Useful in themselves, often used to generate differently distributed deviates.
- Distinguish between linear generators (discussed next) and nonlinear generators (do a web search).

## Linear Congruential Generators

- Typical of most system-supplied generators.
- Produces series of integers  $I_1, I_2, I_3, \dots$ , each between 0 and  $m - 1$ , using:

$$I_{j+1} = aI_j + c \pmod{m},$$

where  $m$  is the modulus, and  $a$  and  $c$  are positive integers called the multiplier and the increment, respectively.

- If  $m$ ,  $a$ , and  $c$  are properly chosen, all possible integers between 0 and  $m - 1$  occur at some point.
  - The choice of  $a = 7^5 = 16807$ ,  $c = 0$ ,  $m = 2^{31} - 1 = 2147483647$  is known as the minimal standard generator.
  - Often  $a$  and  $c$  chosen so as to have integer overflow on nearly every step, giving less predictable sequence and avoiding the mod operation.

- The LCG method is very fast but it suffers from sequential correlations.
- If  $k$  random numbers at a time are used to plot points in  $k$ -dimensional space, points tend to lie on  $(k - 1)$ -dimensional hyperplanes. There will be at most  $m^{1/k}$  planes, e.g.,  $\sim 1600$  if  $k = 3$  and  $m = 2^{32}$ !
- The quality of a LCG is measured by the maximum distance between successive hyperplanes: the smaller the distance, the better.
- Also, low-order bits may be less random than high-order bits, e.g., last bit alternating between 0 and 1.
  - To generate random number between 1 and 10 with `rand()`, use
 

```
j = 1 + (int) (10.0*rand()/(RAND_MAX + 1.0));
```

 and *not*

```
j = 1 + (rand() % 10);
```

 (which uses lower-order bits).

## ***NRiC* RNGs**

- *NRiC* gives several uniform deviate generators:

Generator	Speed	Notes
<code>ran0</code>	1.0	Small multiple, serial correlations.
<code>ran1</code>	1.3	General purpose, maximum $10^8$ values.
<code>ran2</code>	2.0	Like <code>ran1</code> , but longer period.
<code>ran3</code>	0.6	Subtractive method, not well studied.
<code>ranqd1</code>	0.1	Fast, machine-dependent.
<code>ranqd2</code>	0.3	Ditto.
<code>ran4</code>	4.0	Good properties, slow.

- On the department machines, see `rand()`, `random()`, and `drand48()`.
- There is much discussion on the web of relative merits of RNGs. Recommended generators include [TT800](#) and the [Mersenne Twister](#).
- Bottom line: test it yourself, or use web-published testing routines, e.g., spectral methods.

## **Transformation Method**

- Suppose we want to generate a deviate from a distribution  $p(y) dy$ , where  $p(y) = f(y)$  for some positive and normalized function  $f$ , with  $y$  ranging from  $y_{\min}$  to  $y_{\max}$ .
- Let  $F(y)$  be the *cumulative* distribution of  $f(y)$ , from  $y_{\min}$  to  $y$ , i.e.,  $F(y) = \int_{y_{\min}}^y f(y') dy'$ .

- Set a uniform deviate  $x = F(y)/F(y_{\max})$  and solve for  $y$ : this is the new generation function.
- Only useful if  $F^{-1}(x)$  is easy to compute.

### Example: Exponential deviates

- Suppose we want  $p(y) dy = e^{-y} dy$ ,  $y \in [0, \infty)$ .
- Apply the transformation method:
  - Have  $f(y) = e^{-y}$ ,  $F(y) = e^{-0} - e^{-y} = 1 - e^{-y}$ .
  - Set  $x = F(y)/F(\infty)$  and solve  $x(1 - e^{-\infty}) = 1 - e^{-y}$  for  $y$ .
  - Get  $y(x) = -\ln(1 - x) = -\ln(x)$  (since  $1 - x$  is distributed the same as  $x$ ).
- So if  $x$  is a uniform deviate between 0 and 1,  $y(x)$  ( $x > 0$ ) will be an exponential deviate.
- See *NRiC* §7.2 for Gaussian deviates.

### Another example: A simple IMF

- Suppose we want to generate particle masses according to  $M dM = M^\alpha dM$ ,  $M \in [M_{\min}, M_{\max}]$ .
- From the transformation method we get:

$$M = M_{\min} \left\{ 1 + x \left[ \left( \frac{M_{\max}}{M_{\min}} \right)^{\alpha+1} - 1 \right] \right\}^{\frac{1}{\alpha+1}},$$

or

$$M = \left[ (1 - x)M_{\min}^{\alpha+1} + xM_{\max}^{\alpha+1} \right]^{\frac{1}{\alpha+1}}.$$

- Notice that for a flat distribution ( $\alpha = 0$ ), get expected result.
- What happens if  $\alpha = -1$ ? EFTS...

### Initial Conditions

- Often want to generate random initial conditions for a simulation, e.g., initial position and velocity.
- Must take care when using transformations, since may not get distribution you expect.
- For example, to fill a flat disk of radius  $R$  with random points is it better to:
  1. Choose random  $\theta$  and  $r$  then set  $x = r \cos \theta$ ,  $y = r \sin \theta$ ?
  2. Fill a square and reject points with  $x^2 + y^2 > R^2$ ?

Answer: 2, but 1 will work if  $r^2$  (instead of  $r$ ) has a uniform random distribution.

## Application: Cryptography

- A simple encryption/decryption algorithm can be constructed using random number generators.
- If both parties know the initial seed, they can both reproduce the same sequence of values.
- However, communicating the *seed* between parties carries risk.
- One popular technique is to combine *public* and *private* keys for secure communication (the example below is called Diffie-Hellman Key Exchange).
- How do public and private keys work?

Step	You	Your Friend
1	Public: choose large prime $p$ .	Public: choose $b$ , no common factors with $p - 1$ .
2	Private: choose $x$ .	Private: choose $y$ .
3	Compute $b^x \bmod p$ and send.	Compute $b^y \bmod p$ and send.
4	Compute $k = b^{yx} \bmod p$ .	Compute $k = b^{xy} \bmod p$ .

- $k$  is the encryption key. This procedure relies on the fact that it is very difficult to factor large numbers.
- Also uses the handy relationship:

$$(b^y \bmod p)^x \bmod p = (b^y)^x \bmod p, \text{ for any } x, y.$$