Random Numbers

- *NRiC* Chapter 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.
  - WARNING: this may have poor dynamic range, or may be correlated with when the code is run.

Choosing a Generator

- Since generators do not produce truly random sequences, it is 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.
- 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. 2731774 or 10293082 instead of 1 or 4096 or some other power of two.
  3. Avoid sequential seeds: they may cause correlations.
  4. Compare results using at least two generators.
  5. When publishing, indicate generator used.

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. $p(x)\,dx = dx$ if $0 < x < 1$, 0 otherwise.

- 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, ..., 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 increment.
- If m, a, and c are properly chosen, all possible integers between 0 and m -1 occur at some point.

LCGs, Cont'd

- 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. ~1600 if k=3 & m=2^{32}!
- The quality of a LCG is measured by the maximum distance between successive hyperplanes: the smaller the distance, the better.

NRiC RNGs

- *NRiC* gives several uniform deviate generators:

Generator	Speed	Notes
`ran0`	100	Small multiple, serial correlations
`ran1`	130	General purpose, maximum 10^8 values
`ran2`	200	Like `ran1`, but longer period
`ran3`	060	Subtractive method, not well studied
`ranqd1`	010	Fast, machine-dependent
`ranqd2`	025	Ditto
`ran4`	400	Good properties, slow

- There is much discussion on the web of relative merits of RNGs. Recommended generators include TT800 and the Mersenne Twister.

Transformation Method

- Suppose we want to generate a deviate from a distribution $p(y)\,dy$, where $p(y) = f(y)$, with y ranging from y_{min} to y_{max}.
- Let $F(y)$ be the cumulative distribution of $f(y)$, from y_{min} to y.
- Set a uniform deviate $x = F(y)/F(y_{\mathrm{max}})$ 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 $y(x) = -\ln(1 - x)$.
- So if x is a uniform deviate between 0 and 1, $y(x)$ ($x < 1$) 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

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

- What happens if $\alpha = -1$? EFTS...

Initial Conditions

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

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.

Cryptography, Cont'd

- How do public & 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} (mod p) and send	Compute b^{y} (mod p) and send
4	Compute $k = b^{yx}$ (mod p)	Compute $k = b^{xy}$ (mod 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:

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

Simple Monte Carlo Integration

- Can use RNGs to estimate integrals.
- Suppose we pick N random points x_1, ..., x_N uniformly in a multidimensional volume V.
- Basic theorem of Monte Carlo integration:

$$\int_V f\,dV \approx V\langle f\rangle \pm V\sqrt{\frac{\langle f^2\rangle - \langle f\rangle^2}{N}}$$

where $\langle f\rangle \equiv \frac{1}{N}\sum_{i=1}^{N} f(x_i)$ & $\langle f^2\rangle \equiv \frac{1}{N}\sum_{i=1}^{N} f^2(x_i)$

Monte Carlo, Cont'd

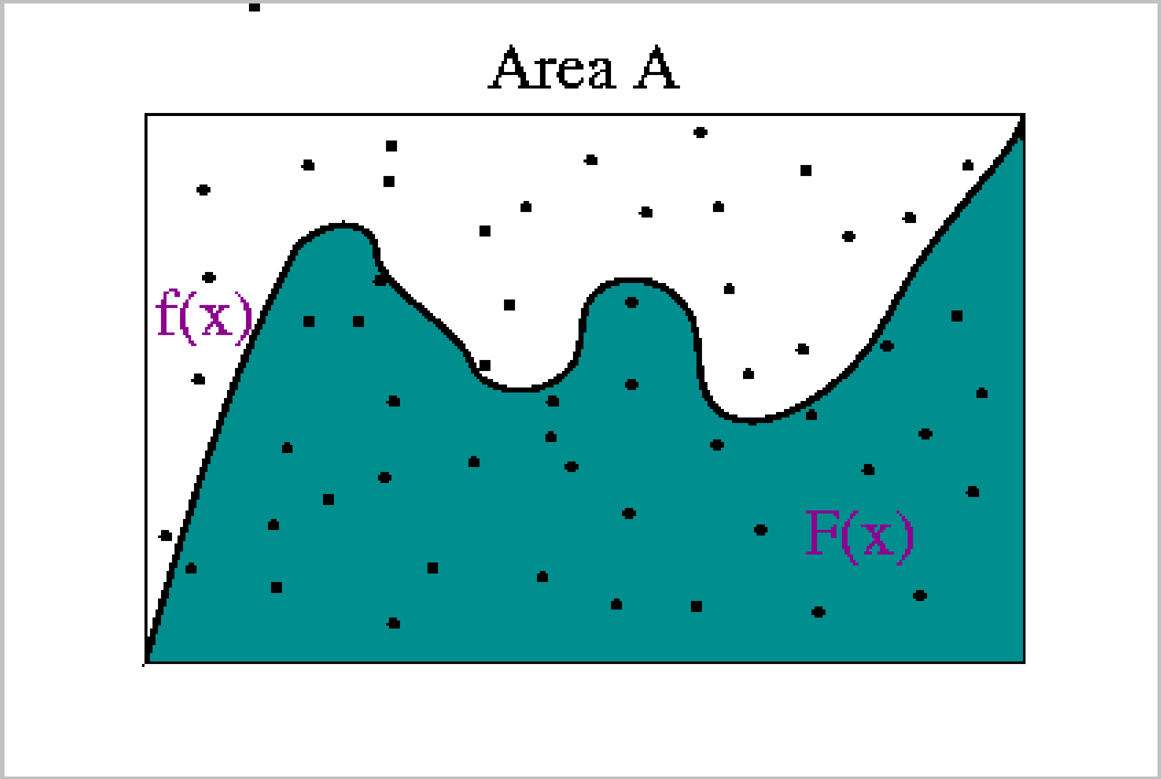
- The error term is 1σ, not a rigorous bound.
- Previous formula works fine if V is simple.
- What if we want to integrate a function g over a region W that is *not* easy to sample randomly?
- Solution: find a simple volume V that *encloses* W and define a new function $f(\mathbf{x})$, $\mathbf{x} \in V$ such that:

$$f(\mathbf{x}) = g(\mathbf{x}) \quad \text{for all } \mathbf{x} \in W$$

$$f(\mathbf{x}) = 0 \qquad \text{otherwise}$$

Monte Carlo, Cont'd

- Strategy: make *V* as close as possible to *W*, since zero values of *f* will increase the error estimate.



Monte Carlo, Cont'd

- Principal disadvantage: accuracy increases only as square root of *N*.
- Fancier routines exist for faster convergence.

  Cf. NRiC §7.7-7.8.

- Monte Carlo techniques used in a variety of other contexts: anywhere statistical sampling is useful.

  e.g. Predicting motion of bodies with short Lyapunov times if starting positions & velocities poorly known.