

CS 475/575  
Slide set 4

M. Overstreet  
Old Dominion University  
Spring 2005

---

---

---

---

---

---

---

---

Random Number Generators

Techniques to generate **random numbers**:

(very old) random number tables

seldom done

even cheap calculators likely to rnd function

though who knows how good they are

observation of physical phenomena

slow, expensive, not reproducible

computer generated:

find a function  $f$  such that given a seed  $x_0$ ,

$$\{x_i \mid x_i = f(x_{i-1}), 0 \leq i \leq n\}$$

---

---

---

---

---

---

---

---

computer generated (cont.)

⌘ thus

$$x_1 = f(x_0)$$

$$x_2 = f(x_1)$$

$$x_3 = f(x_2)$$

$$x_4 = f(x_3)$$

etc.

once you pick  $x_0$ , everything is determined and always produces the same sequence

---

---

---

---

---

---

---

---

## desirable properties

⌘ ideally, f should be:

- small (& easy to code)
- fast
- reproducible
- produce as many values as desired.

⌘ infinite length is impossible. why?

⌘ for practical and historical reasons, we usually generate  $u(0,1)$  random numbers.

---

---

---

---

---

---

---

---

## what is "random"?

⌘ suppose we have a black box which supposedly produces the digits 0 - 9 in random order.

⌘ if we turn the box on and the 1st digit is 6, is this random?

- can't be answered, not enough information

⌘ "randomness" is a statistical property of a sequence of numbers

- does not depend on how numbers are generated
- choice of essential characteristics not scientific
  - but no "patterns"

---

---

---

---

---

---

---

---

## tests for randomness

⌘ most seem to be along the line of eliminating undesirable characteristics:

- frequency counts; about the same number in equal sized subintervals.

(what if the box produced 100 integers, but in the order 10 0's, then 10 1's ..., etc.)

- serial tests, the pairs

$(x_i, x_{i+1}), (x_i, x_{i+2}), (x_i, x_{i+3}), etc.$

should be uniformly distributed.

---

---

---

---

---

---

---

---

## tests (cont.)

### ⌘ more tests

- gap test
- poker test
- coupon collector's test
- permutation test
- runs up and runs down test
- monkey test

### ⌘ requirements for test:

- have understood statistical distribution (so you can measure)
- "strength" of test: if fail test a => fail test b, don't do b

---

---

---

---

---

---

---

---

## bad ideas:

### ⌘ VonNeumann's midsquare method (40's):

$$x_{n+1} = \text{middledigits}(x_n^2)$$

### ⌘ Knuth's unpredictable code method (50's):

alg. with "random" (at least hard to anticipate from reading the code) iterations, jumps, shifts, flips, etc.

---

---

---

---

---

---

---

---

## rng (from Park & Miller, CACM, 10/88)

### ⌘ rng goals

- full period
- random
- efficient
- reproducible

### ⌘ proposed by Lehmer in 1951: prime modulus multiplicative linear congruential generator:

$$f(z) = a * z \text{ mod } m,$$

where m is prime

---

---

---

---

---

---

---

---

## rng (more)

⌘ usually want a float in  $[0,1]$ , so

$$u_i = z_i / m$$

⌘ this is good because:

- 1) since  $m$  is prime,  $\forall (z) \neq 0, \Delta z \in [1, \dots, m-1]$
- 2)  $1/m \leq u_i \leq 1 - 1/m \forall i$
- 3) randomness of  $u_i$  is the same as the randomness of  $z_i$

---

---

---

---

---

---

---

---

## rng mod arith example

1)  $f(z) = 6z \bmod 13$ : if start with  $z = 1$ , get

1, 6, 10, 8, 9, 2, 12, 7, 3, 5, 4, 11, 1, ...

if start with  $z = 2$ , what happens?

2)  $f(z) = 7z \bmod 13$ , start with 1, to get

1, 7, 10, 5, 9, 11, 12, 6, 3, 8, 4, 2, 1

3)  $f(z) = 5z \bmod 13$ , start with 1, to get

1, 5, 12, 8, 1, ...

start with 2, to get

2, 10, 11, 3, 2, ...

---

---

---

---

---

---

---

---

## rng (cont.)

⌘ common choice for modulus is  $2^b$

(why?)

⌘ better choice is  $2^{31} - 1 = 2147483647$

(is this prime?)

⌘ with this choice of  $m$ , still need:

☐ a choice for  $a$  (P&M recommend 16807)

☐ an efficient implementation

⌘ so  $f(z) = 16807 * z \bmod 2147483647$ .

(this generator has been extensively tested)

---

---

---

---

---

---

---

---

## theory support RNGs

⌘ using number theory, can ensure max period

⌘ assume generator is *mixed linear congruential* of form:

$$x^{l+1} = ax^l + c \text{ mod } m$$

here "mixed" means both add & mult. if  $c=0$ , then this is a *multiplicative linear congruential generator*

---

---

---

---

---

---

---

---

## max period - 1

⌘ max period achieved by proper choice of  $a$ ,  $c$ ,  $m$ , and  $x_0$ :

⌘ if  $m = 2^b$ ,  $c=0$ , longest period,  $P$ , is  $P = m/4 = 2^{b-2}$  provided:

- 1)  $x_0$  is odd,
- 2)  $a = 3 + 8k$  for some integer  $k$

---

---

---

---

---

---

---

---

## max period - 2

⌘ For  $m$  prime and  $c=0$ , longest period  $P = m-1$  provided:

$a$  has the property that the smallest integer  $k$  such that  $a^k - 1$  is divisible by  $m$  is  $k=m-1$ .

---

---

---

---

---

---

---

---

## problem: maximal period not only issue

### ⌘ Problems with serial correlation

- ☒ If  $k$  random numbers at a time are used to plot points in  $k$ -space, the points will not “fill up” the space, but will fall in  $k-1$  planes.
- ☒ At most about  $m^{1/k}$  planes.
- ☒ Randu is an infamous example provided by IBM
  - ☒  $a=65539, c=0, m=2^{31}$
- ☒ See `~cs475/plotting`. Type “gnuplot” then at prompt type  
load “plot.triples”

---

---

---

---

---

---

---

---

## some code: randu

```
/* Bad Example! Do not use!! */
/* Returns values between 0 and 1. */
/* Depends on 32 bit representation for integers. */
double randu( seed )
long int *seed;
{
    long int a = 16807,
            mod = 2147483647; /* 2^31 - 1 */
    double dmod = 2147483647.0; /* 2^31 - 1 */

    *seed = *seed * a;
    if ( *seed < 0 )
        *seed += mod;
    return( ( double ) *seed / dmod );
}
```

---

---

---

---

---

---

---

---

## better code: random

```
/* C implementation Park & Miller's random function */
double random ( seed )
long int *seed;
{
    long int a = 16807, /* 7^5 */
            m = 2147483647, /* 2^31 - 1 */
            q = 127773, /* m / a (int divide) */
            r = 2836, /* m % a */
            lo, hi, test;
    double dm = 2147483647;

    hi = *seed / q;
    lo = *seed % q;
    test = a * lo - r * hi;
    *seed = test > 0 ? test : test + m;
    return( (double ) *seed / dm );
}
```

---

---

---

---

---

---

---

---

## Reference

⌘Leemis, chapter 2.

---

---

---

---

---

---

---