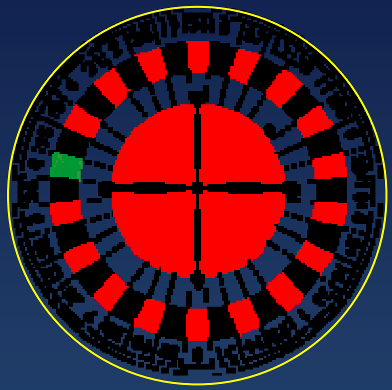
Simulating Randomness (Monte-Carlo Techniques)

(major scientific use)

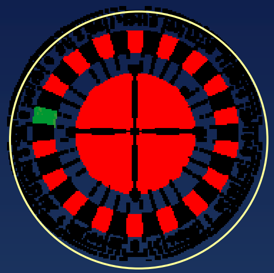


Rubin H Landau
With
Sally Haerer and Scott Clark

Computational Physics for Undergraduates
BS Degree Program: Oregon State University

"Engaging People in Cyber Infrastructure"
Support by EPICS/NSF & OSU

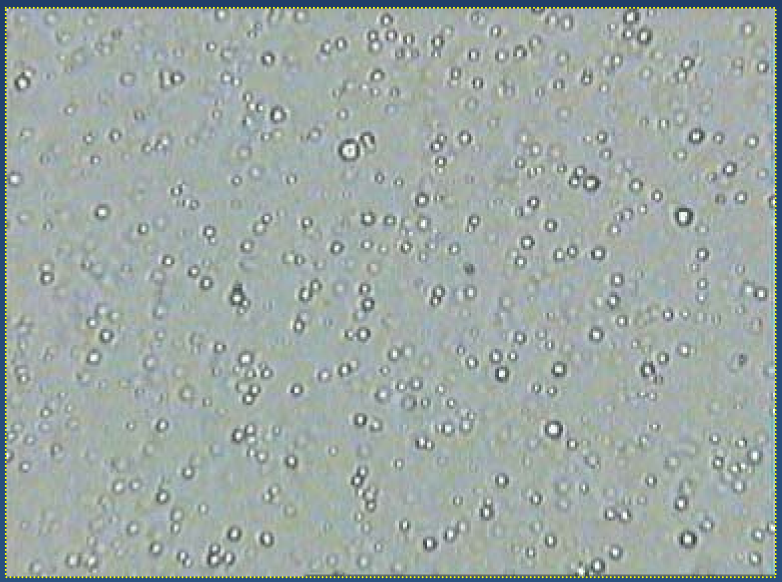
Deterministic Randomness



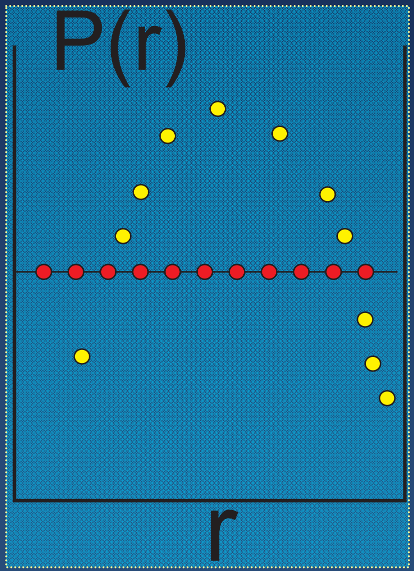
- Computers are *deterministic*; no chance involved
- Always same output for same input; unless error
- Generate pseudo-random numbers
- Monte Carlo calculations: simulate random events

Examples:

1. thermal motion
2. games of *chance (*meaning?)
3. radioactive decay
4. solve equations statistically
5. solve intractable problems



Theory: Random Sequences



- *Random Sequence*: r_1, r_2, ...
  - no correlations among numbers (predict)
- *Not* $\Rightarrow$ equally likely (can be)
- *Uniform Sequence*: r_i equally likely
- *e.g,:* 1, 2, 3, 4, ... = uniform √, random X
- *e.g.:* 3, 1, 4, 2, ... random ?, uniform √
- Distribution function $\mathcal{P}(r)$ (fig)
  - $\mathcal{P}(r)\, dr$ = probability $r \le r \le r+dr$
  - uniform: $\mathcal{P}(r)$ = constant
- Random number generator: also uniform [0,1]
- Can use tables or nature (deck of cards)

Linear Congruent Rand Generator

$$r_i \stackrel{\text{def}}{=} (a\, r_{i-1} + c) \bmod M \qquad (1)$$

$$= \text{remainder}\left(\frac{a\, r_{i-1} + c}{M}\right) \qquad (2)$$

- Most common method
- Interval [*0, M-1*]
- r_1 = *seed*; supplied by user;
  - *M = very large (cycle)*
  - a (large), c: black magic
- mod = remainder function (amod, dmod),
  - *e.g. 4 mod 2 = 0, 5 mod 2 = 1*
- Effectively: r_i = least significant part (≈ round-off error)

Linear Congruent Example

$$r_i = (a\, r_{i-1} + c) \bmod M \qquad (1)$$

- Start: $c = 1,\ a = 4,\ M = 9,\ r_1 = 3$ (2)

  $r_1 = 3$

  $r_2 = (4 \times 3 + 1) \bmod 9 = 13 \bmod 9 = R(13/9) = 4$ (3)

  $r_3 = (4 \times 4 + 1) \bmod 9 = 17 \bmod 9 = 8$

  $r_4 = (4 \times 8 + 1) \bmod 9 = 33 \bmod 9 = 6$

  $r_{5\text{-}10} = 7,\ 2,\ 0,\ 1,\ 5,\ 3$

- Sequence length M = 9 (repeats)
- For range [0, 1]: r / M (= 9) (4)

  0.333, 0.444, 0.889, 0.667, 0.778, 0.222,

  0.000, 0.111, 0.555, 0.333

Random Facts of Life

algorithm

- $r_i = (a\, r_{i-1} + c)$ mod $M \quad \Rightarrow$ Range $0 \cdot \; r_i \cdot \; M - 1$ (1)
- r repeats $\Rightarrow$ cycle $\Rightarrow$ large M & a
- $\Rightarrow \geq$ 48-bit integers

(good) $2^{48} \simeq 3 \times 10^{14}$ (2)

(small = bad) $M = 2^{31} \simeq 2 \times 10^{9}$

- Methods: rand, rn, random*, srand, erand, drand*, drand48* (3)

$$e.g.\ M = 2^{48}, \quad c = B_{16} = 13_8$$

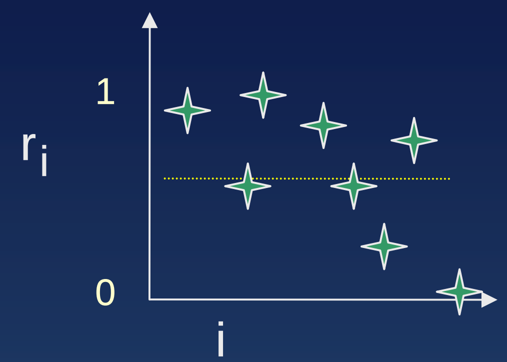
$$a = 5DEECE66D_{16} = 273673163155_8$$

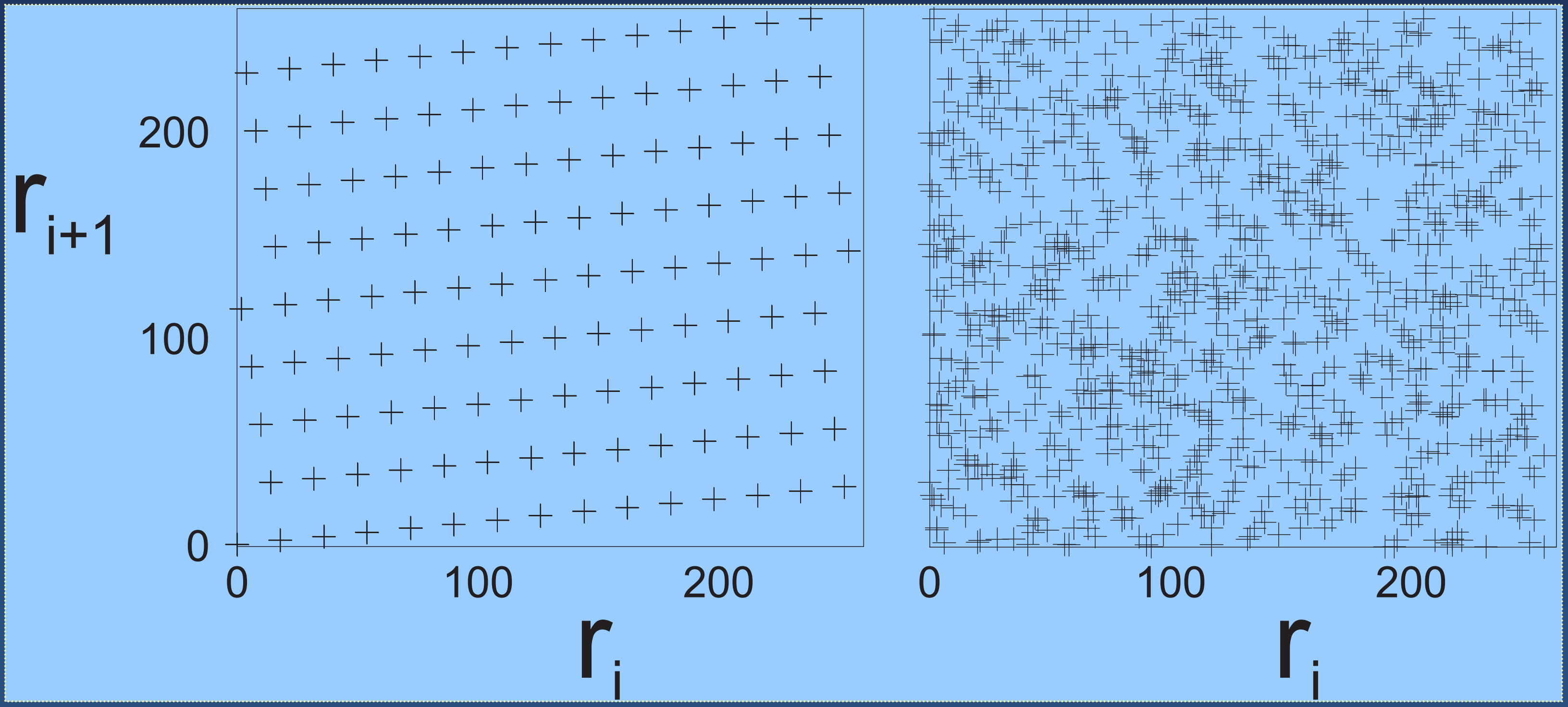
Scale for range $A \cdot \; x_i \cdot \; B$

$$x_i = A + (B - A) r_i, \quad 0 \cdot \; r_i \cdot \; 1 \qquad (4)$$

Tests for Randomness (ESTD)

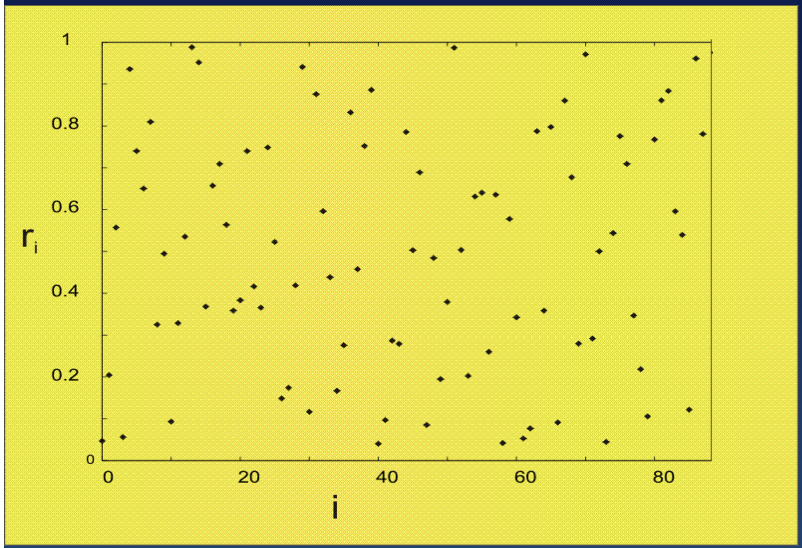
- Always check before use (war stories)
  - print (random, range), plot
- Use grey matter to "see" correlations
- Plot (x, y) = (r_i, r_{i+1})





Tests for Uniformity (large N)

- Here is the rub



k th moment of distribution

$$\langle x^k \rangle = \frac{1}{N}\sum_{i=1}^{N} x_i^k \simeq \int_0^1 dx\, x^k \mathcal{P}(x) \quad (1)$$

$$\simeq \frac{1}{k+1} \quad (2)$$

- Uniformity via near-neighbor correlation

(3)

$$C(k) = \frac{1}{N}\sum_{i=1}^{N} x_i x_{i+k}, \quad (k = 1, 2, \ldots) \simeq \int_0^1 dx \int_0^1 dy\, xy\, \mathcal{P}(x,y) = \frac{1}{4}$$

(4)

- Randomness test: relative deviations $\simeq 1/\sqrt{N}$

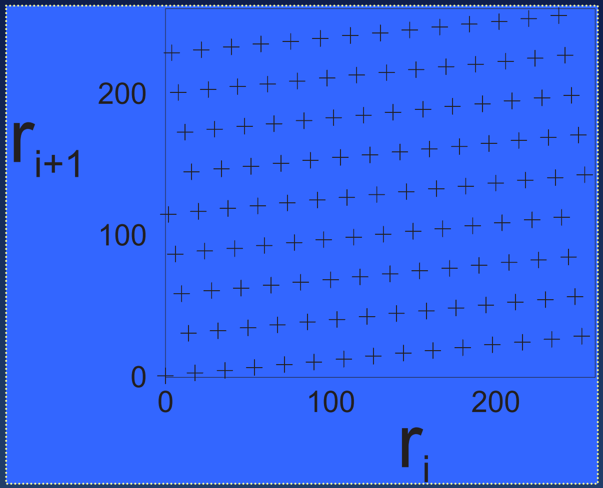
Implementation: RandNum.java

```
//  RandNum.java:     random numbers via Java utilities
import java.io.*;                       //Location of PrintWriter
import java.util.*;                          //Location of Random
public class RandNum
{  public static void main(String[] argv)
        throws IOException, FileNotFoundException {
 PrintWriter q = new PrintWriter(
        new FileOutputStream("RandNum.DAT"),true);
 long seed = 899432;                         // Initialize, seed
 Random randnum = new Random(seed);
 int imax = 100;
 int I = 0;
 for(i = 1; i <= imax; i++)    // Generate random sequence
   q.println( randnum.nextDouble() );
System.out.println( " “ );
System.out.println( "RandNum Program Complete.“ );
System.out.println( "Data stored in RandNum.DAT“ );
System.out.println( " “ );
} }
```

Time for Lab!

- **Time to "play games"**
- **Look inside the (X) box**

Random Exercises



1. Write <u>your own</u> random number generator

   (<u>not for prime time</u>)

   a. Linear congruent method

   b. Unwise choice: (a, c, M, r_1) = (57, 1, 256, 10)

   c. Period = ?

   d. Plot points (x_i, y_i) = (r_{2i-1}, r_{2i})

2. Repeat for built-in generator (industrial strength?)

Lab Exercises: cont

3. Test linear congruent method with reasonable constants.

4. Test built-in generator for uniformity (and randomness)

k=1, 3, 7, N = 100, 10,000, 100,000.

$$\left| \frac{1}{N} \sum_{i=1}^{N} x_i^k - \frac{1}{k+1} \right| \simeq \mathcal{O}(1/\sqrt{N}) \qquad (1)$$