Demonstration

// Show program random_LCG.cpp

$x_0 = 1$, $a = 16807$, $b = 2^{31} - 1 = 2\,147\,483\,647$

$c = 0$ (Lewis-Goodman-Miller parameters)

Also $a = 3$, $b = 32$

$c = 0$, 3, 4 observe periodicity

from Klein and Godunov

$a = 65$, $b = 65\,537$, $c = 319$

```
./random_LCG a b c n | awk '{print $1, $3}'
                                  | xmgrace -p p.par -
```

or

```
./random_LCG a b c n | xmgrace -p p.par -block - -bxy 1:3
```

- series looks good - show histogram (deviations should be Gaussian)
- show correlations when plotting
  x_{n+1} vs x_n (./example.sh)

- periodicity: a short periodicity is bad
  longest periodicity is b
  periodicity kicks in as soon as a number is repeated
  e.g. <u>5 7</u> 9 13 <u>5 7</u> 9 ...

However a truly random sequence can have repeats

randomlists.com/random_numbers random.org/sequences

- setting the seed allows us to reproduce results
- changing seed gives another realization of the simulation //

Another approach: Generalized Feedback Shift Register Method

n, p and q are integers, $p > q$

-> using some approach, generate

e.g. $p = 5$, $q = 3$

$x_1 =$

$x_6 =$

where $\oplus$ is operation on

representing

e.g. $4 \oplus 7 = ?$

$4_{10} =$

$7_{10} =$

4

7

one of, but not both

result of exclusive or

$x_6 =$

This is an example of - requires bit manipulation f$^{\underline{n}}$s

Must choose p and q carefully (521 and 168 work well)

In C/C++ $4 \oplus 7$ is

Fortran

show xor.cpp

& NR-based programs

Can combine two generators

e.g. use 2 LCGs or 1 LCG and 1 bit shuffler

- use RNG1 to get, say, $N = 256$ RNs
- store in a list
- use RNG2 to pick a random number M between 1 and 256
- replace M^{th} element by RN generated by RNG1

→ " "

In the end, simulation results must be independent of the RNG - can confirm this by getting same results

Even if RNG passes statistical tests, results may be

e.g. If process is sensitive to numbers in the range 0.7112 ± 0.0001, small bias in RNG will have

Note from Numerical Recipes

- Never use
- Never use method with
- Never use method that

- Never use

UPSHOT: It's hard to be random!

Monte Carlo (MC) Integration

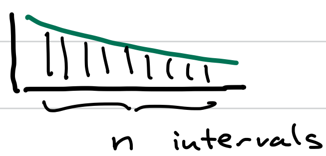
Usual methods like Simpson's rule work well in

For d-dimensional integration, trapezoid rule has error , where N is # of integrand evaluations

For MC method, error - independent of
→ MC converges faster for

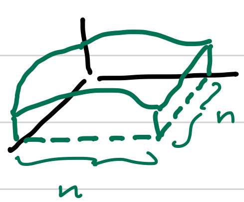
Classical approaches

1-D



$t_{cpu} \propto$

2-D

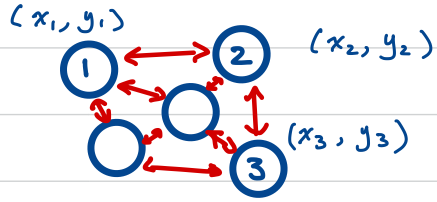


$t_{cpu} \propto$

d-D $t_{cpu} \propto$

E.g. from statistical mechanics

Consider 50 atoms on a surface



U = potential energy of whole system

=

Average energy at temperature T is given by

$\langle U \rangle = \frac{1}{Q}$

this is a d=

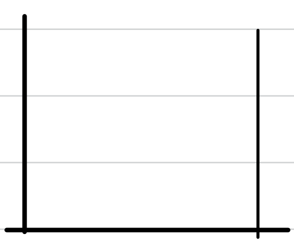
For classical method $t_{cpu} \sim$ (let)

$\sim$

For processor $\rightarrow$

that does

MC Approach #1



Generate pairs of RNs (x_i, y_i)

s.t.

and

probability that point (x_i, y_i) =

falls under $f(x)$ curve