I.2. follows to a large extent the chapter *Random Numbers* in W. Kinzel and G. Reents, *Physics by Computer* (1998).
Further information and related Mathematica and C programs are available at
`http://theorie.physik.uni-wuerzburg.de/TP3/physbc.html`

I.2. Random Number Generators

Processes like coin tossing, lotto, throwing dice, radioactive decay, ...
are generally accepted to be subject to *real randomness*

Deterministic calculations cannot yield random numbers.

but: non-linear deterministic maps can yield *effectively unpredictable* sequences of **pseudo random numbers**

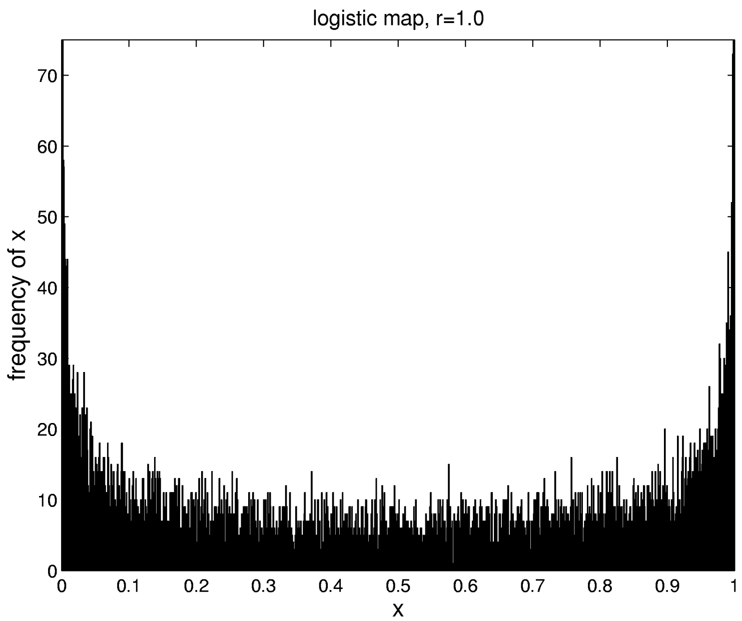

Fig. 1: the logistic map with $r = 1$,

the density of $x \in [0, 1]$ is $p(x) = \dfrac{1}{\pi\sqrt{x(1-x)}}$

Random Numbers are the basis of **Monte Carlo simulations**, see II and III

most frequently, a density $P(x) = \begin{cases} 1 & \text{for } x \in [0,1] \\ 0 & \text{else} \end{cases}$ is used

---

if a computer uses, e.g., 32 *bits* to represent numbers
then *only* $m = 2^{32} = 4.294.967.296$ different numbers can be realized

simple **non-linear iterations** of the form $k_{n+1} = f(k_n)$
are bound to be periodic with (largest possible) period m (e.g. 2^{32}).

we consider integer iterations with $0 \leq k_n < m \quad \rightarrow \quad x_n = k_n/m \in [0, 1)$

Linear congruential generators

widely used generators of the form

$k_{n+1} = f(k_n) = (a\, k_n + c)$ mod m

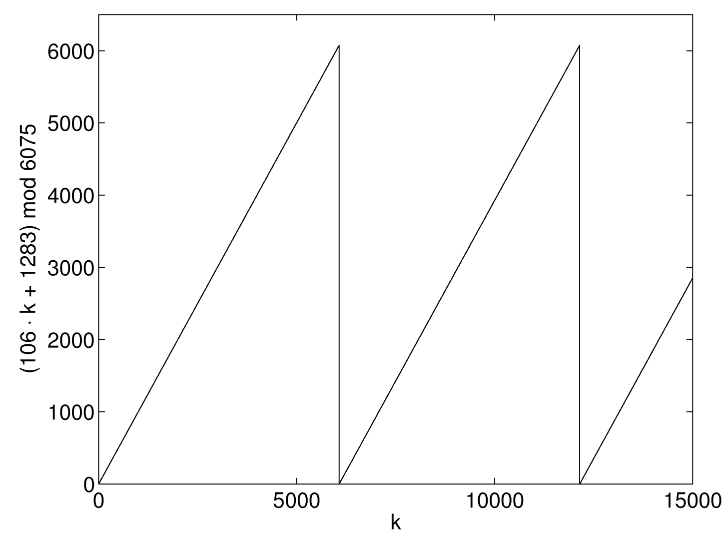

Fig. 2: *$f(k)$ of the lin. congruential generator with $a = 106, c = 1283,$ and period $m = 6075$.*

in practice: try to realize the largest possible m (e.g. 2^{32})

note : many compilers *automatically* implement the modulo-function when using 32-bit integers, e.g. in C:

```
unsigned long int k
k = 69069*k +1013904243;
```

illustrative example: a generator with rather short period $m = 6075$

it should produce

(1) all possible $m = 6075$ numbers in the sequence (maximum period)

(2) in a *pseudo random* ordering (as to be checked by statistical tests)

requirement (1) is easy to meet: $a = 1, c = 1$ yields $k = 0, 1, 2, 3, \ldots, 6074, 0, 1, 2 \ldots$

(not *quite* random...)

number theoretical methods provide *good* choices $\{a, c\}$

example from *Numerical Recipes:* [1] $a = 106,\ c = 1283$

[1] W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery, *Numerical Recipes in C: The Art of Scientific Computing*, Cambridge Univ. Press (1992)

Matlab routines

```
   function mat = simplerand(M,N,initial)
%  simplerand(M,N,initial) returns an M x N  matrix
%  of pseudo random numbers obtained from the lin.
%  congruential generator a= 106, c=1283, m=6075
%  with initial value x(0) = initial {default: 1234}
%  x-values are rescaled to the interval (0,1)
   x    = 1234;
   if nargin ==3
      x = initial;
   end
   mat  = ones(M,N);
   a    = 106; c    = 1283; m    = 6075;

   for i =1:M
   for j =1:N
          x = mod(a*x+c,m);
          mat(i,j) = x/m;
   end
   end
```

```
%    cov.m  (no arguments) generates 2 sequences of
%    6075 pseudo random numbers, one from simplerand
%    the other using the built-in matlab rand fct.
%    For each generator, the covariances
%    <x(i) x(i+n)> - <x(i)><x(i)>  are plotted
%    versus  n =1,2,.... nmax
  nmax =3037; m=6075;
  xsimple = simplerand(1,m); sm= sum(xsimple)/m;
  xbuilt = rand(1,m); bm= sum(xbuilt )/m;
  covs = ones(1,nmax); covb = ones(1,nmax);
  ii = [1:nmax];

  for i=1:nmax
    xsimpli = [xsimple(i+1:m) xsimple(1:i)];
    xbuilti = [ xbuilt(i+1:m) xbuilt(1:i)];
    covs(i) = sum(xsimpli .* xsimple)/m - sm*sm;
    covb(i) = sum(xbuilti .* xbuilt )/m - bm*bm;
  end
  figure(1); plot(ii, covs); axis([0 nmax -.06 .06]);
  figure(2); plot(ii, covb); axis([0 nmax -.06 .06]);
```

```
% randcompare (no arguments) generates 2 sequences
% of 6075 pseudo random numbers (from simplerand
% and the built in rand function respectively).
% For each generator, all triples of the form
% (x(i),x(i+1),x(i+2)) are plotted in 3d

  xsimple = simplerand(1,6075);
  ysimple = [ xsimple(2:6075) xsimple(1)];
  zsimple = [ xsimple(3:6075) xsimple(1:2)];
  figure(1); scatter3(xsimple,ysimple,zsimple);

  xbuilt = rand(1,6075);
  ybuilt = [ xbuilt(2:6075) xbuilt(1)];
  zbuilt = [ xbuilt(3:6075) xbuilt(1:2)];
  figure(2); scatter3(xbuilt,ybuilt,zbuilt);
```

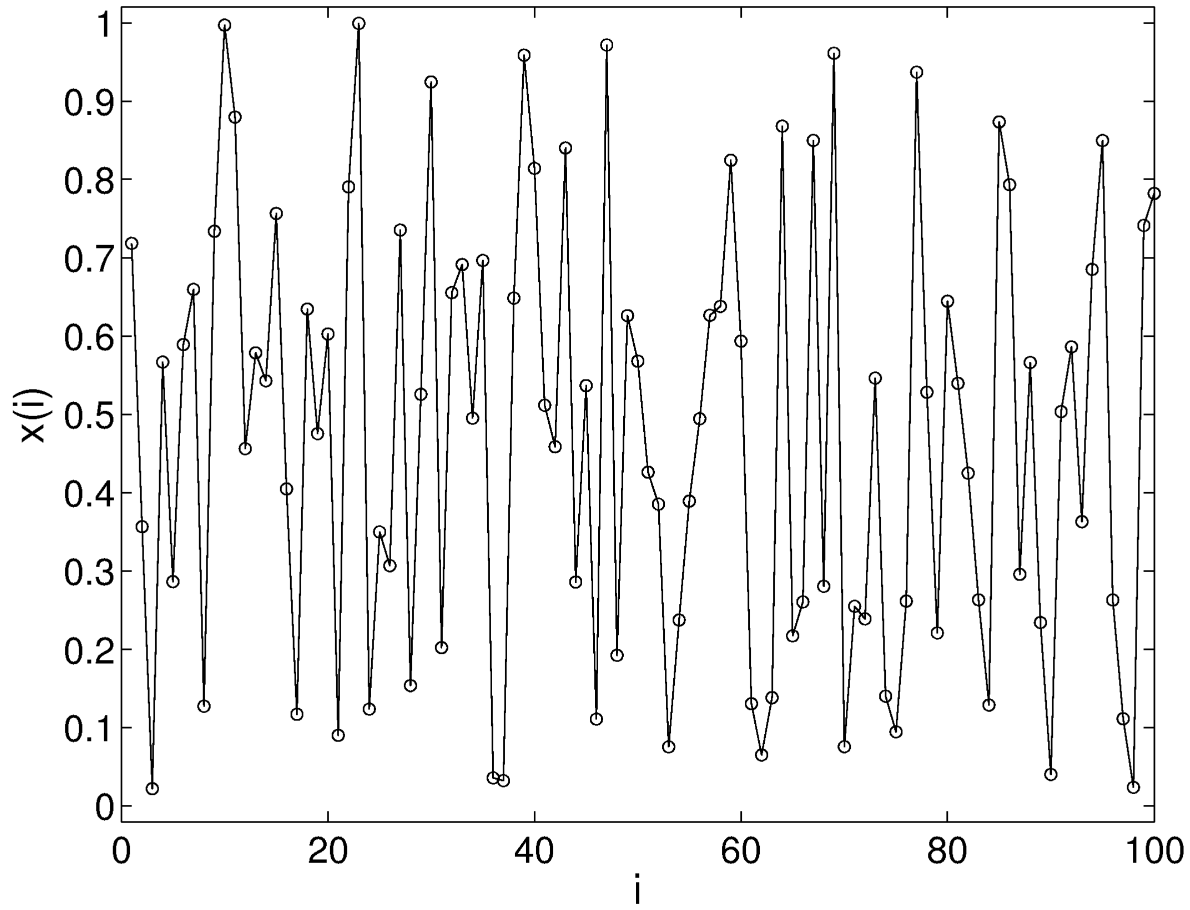

Fig. 3: *A sequence of 100 values* $x_i = k_i/m$ *for the generator with* $a = 106, c = 1283, m = 6075$*.*

Covariances

true statistical independence of the random numbers would imply ($\Rightarrow$)

$$\text{cov}(n) \equiv \langle x_i x_{i+n} \rangle - \langle x_i \rangle^2 = 0 \qquad \text{for all} \quad 1 \leq n \; (\leq m/2)$$

where $\langle x_i \rangle = \frac{1}{m} \sum_{j=1}^{m} x_j = 1/2$ and $\langle x_i x_{i+n} \rangle = \frac{1}{m} \sum_{j=1}^{m} x_j \, x_{j+n} \overset{?}{=} 1/4$

Significant non-zero cov(n) indicate a *memory* over n steps in the sequence

Compare: the complete sequence of the simple generator with
6075 random numbers obtained with the Matlab built-in fct. `rand()`

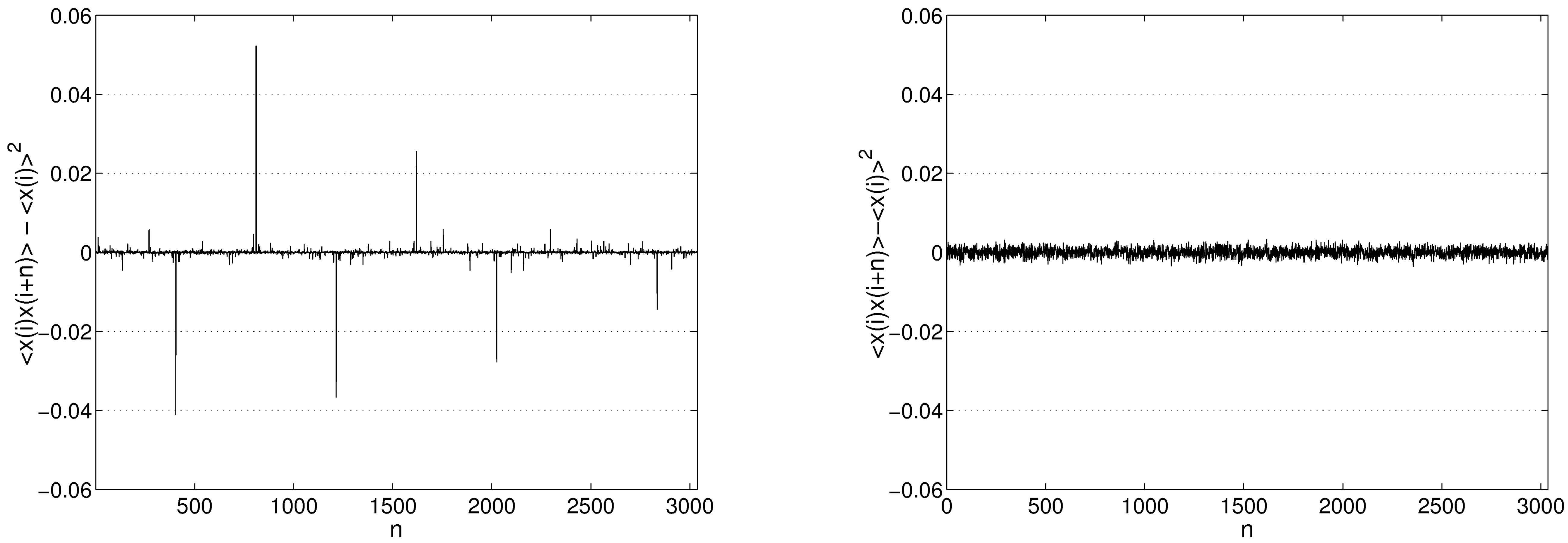

Fig. 4: Covariances $cov(n)$ for the simple generator (left panel) and Matlab function `rand()` (right panel). The simple generator yields relatively large absolute values of $cov(n)$ at characteristic n, e.g. at multiples of 405

3D-Visualization

consider triples of subsequent values (x_i, x_{i+1}, x_{i+2})

a truly random sequence should evenly distribute points in the cube $[0,1]\times[0,1]\times[0,1]$.

Compare: the complete sequence of the simple generator with
6075 random numbers obtained with the Matlab built-in fct. `rand()`

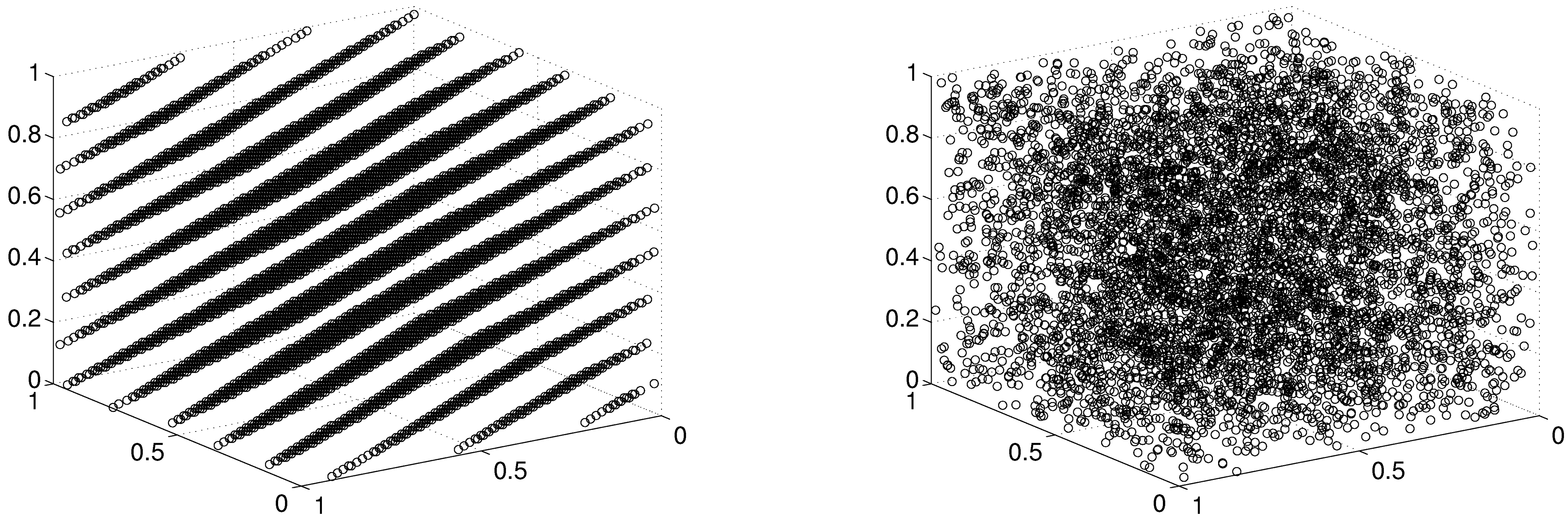

Fig. 5: All triples (x_i, x_{i+1}, x_{i+2}) from the simple generator (left) fall into 14 planes. For the Matlab function `rand()`, the data is obviously more evenly distributed.

Realization of larger periods

- more bits for representing numbers (compiler, hardware)

- combine several generators, e.g.: switch between two functions f, g,

  triggered by a third (independent) generator h $\quad k_{n+1} = \begin{cases} f(k_n) & \text{if } \hat{x}_n \leq 1/2 \\ g(k_n) & \text{else} \end{cases}$

  where $\tilde{x}_n$ is obtained from a sequence $\hat{k}_n = h(\hat{k}_{n-1})$

- use functions of *several* previous numbers

$$k_{n+1} \;=\; f(k_{n-t}, k_{n-s}) \qquad \text{with integers } t > s$$

  The sequence repeats exactly, if the t numbers

  $\{k_{n-t}, k_{n-t+1}, \ldots, k_{n-1}\}$ are encountered again

  $\rightarrow$ period of length $2^{32\,t}$ is possible (problem: find f)

An example: “mzran” [Marsaglia and Zaman, 1994] [2]

subtract-with-borrow generator (period $\sim 2^{95}$)

$$k_n^{(1)} = \left(k_{n-2}^{(1)} - k_{n-3}^{(1)} - c_{n-1}\right) \mod (2^{32} - 18) \quad \text{where}$$

$$c_n = \begin{cases} 0 & \text{if} \quad (k_{n-2}^{(1)} - k_{n-3}^{(1)} - c_{n-1}) > 0 \\ 1 & \text{else} \end{cases}$$

combined with the linear congruential generator (period $\sim 2^{32}$)

$$k_n^{(2)} = (69069\, k_{n-1}^{(2)} + 1013904243) \text{ mod } 2^{32}$$

output:

$$k_n = \left(k_n^{(1)} + k_n^{(2)}\right) \text{ mod } 2^{32}$$

This sequence of k_n has a period of about 2^{127}

(1000 iterations per second $\rightarrow$ $\sim 10^{24}$ CPU-years for 1 period)

[2] G. Marsaglia, A. Zaman, *Some portable very-long period random number generators.* Computers in Phys. 8 (1994) 117.

A few remarks

- even the best generator produces a deterministic, reproducable sequence of – at best – *pseudo random numbers*

- advantage: Monte Carlo results can be exactly reproduced later (important: initialization should be recorded with the simulation results)

- there is no *ultimate statistical test* that could *prove* the usefulness of a generator

- (higher order) correlations can be *hidden* very well, yet corrupt the outcome of Monte Carlo simulations

- recommended: use different generators and compare results

A random number generator is $good$ until somebody[3] discovers that it is not.

---

[3] Two examples:
A.M. Ferrenberg, D.P. Landau, Y.J. Wong, Phys. Rev. Lett. 69 (1992) 3382.
F. Schmid and N.B. Wilding, Intn. Journ. Mod. Phys. C 6 (1995) 781.