Computational physics

Heiko Rieger and Adam Wysocki

Theoretical Physics
Saarland University

SS2017

Contents

Random number

What are random numbers?
It is a sequence of numbers that cannot be reasonably predicted better than by a random chance, i.e., lack of pattern or predictability in events.

Kolmogorov randomness: a string of bits is random if and only if it is shorter than any computer program (without input) that can produce that string, i.e., a random string is "incompressible".

Random number generator (RNG)

- Physical methods (non-deterministic):
  - Dice, coin flipping and roulette wheels.
  - Thermal noise from a resistor.
  - Atmospheric noise, detected by a radio receiver.
  - A nuclear decay radiation source, detected by a Geiger counter.
- Computational methods (deterministic):
  - Maybe some irrational numbers, like π, $\sqrt{2}$, ... , are good RNG. For example, π passed tests for statistical randomness, including tests for normality.
  - Recursive arithmetic RNG, will be presented in the following

Pseudorandom number generator (PRNG)

- PRNG is an deterministic algorithm for generating a sequence of numbers whose properties approximate the properties of sequences of random numbers.
- PRNG-generated sequence is not truly random, because it is completely determined by the initial seed.
- The same seed leads to the same sequence and only different seeds lead to different sequences.
- Mostly PRNG generate integer values $rand \in \{0, 1, ..., m - 1\}$ and division by m leads to $rand \in [0, 1]$.

Pseudorandom number generator (PRNG)

- Simplest method is an iterated function

$$f : \{0, ..., m-1\}^l \to \{0, ..., m-1\} \tag{1}$$

  with arithmetic operations $(+, -, \times, /)$.
  It generates successive numbers

$$i_n = f(i_{n-1}, i_{n-2}, ..., i_{n-l}) \tag{2}$$

  using an initial seed $i_0, ..., i_{l-1}$.
- The function f should be highly nonlinear and chaotic in order to generate good random numbers.
- For PRNG of the type Eq.(2) there is n_0 and p such that $i_{n+p} = i_n$ for all $n \geq n_0$, the smallest p denotes the period of the PRNG.
- If a PRNG's internal state contains n bits its period p can be no longer than $m^l = 2^{nl}$. The aim is to construct a PRNG with $p = m^l$ in order to produce the maximum possible number of random numbers.

Linear congruential generator (LCG)

The method represents one of the oldest and best-known PRNG.

$$i_{n+1} = (ai_n + c) \bmod m \tag{3}$$

with modulo operation

$$x \bmod m := x - \left\lfloor \frac{x}{m} \right\rfloor \cdot m \tag{4}$$

where $\lfloor \ldots \rfloor$ is the floor functions.

- modulus m, $0 < m$
- multiplier a, $0 < a < m$
- increment c, $0 \leq c < m$

Linear congruential generator

Hull-Dobell Theorem: the period is $p = m$ for all seed values if and only if:

- m and c are relatively prime (if the only positive integer that divides m and c is 1)
- $a - 1$ is divisible by all prime factors of m
- $a - 1$ is divisible by 4 if m is divisible by 4.

Choice of a, c, m:

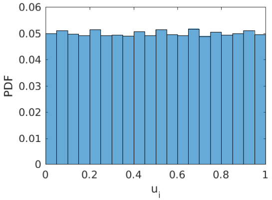
- a, c, m even number $\Rightarrow p < \frac{m}{2}$
- Do not use $a = 1$, because $i_n = (i_0 + nc) \bmod m$ is not very random!
- $m = 2^n$: the ith least significant digit repeats with at most period $2^i \Rightarrow$ alternately odd and even results
- Park and Miller propose: $m = 2^{31} - 1 = 2147483647$, $a = 16807$, $c = 0$

Linear congruential generator (LCG)

Linear congruential generator is not free of sequential correlation.

Marsaglia's Theorem:
Let be $u_n = \frac{i_n}{m} \in [0, 1]$ a number generated by $i_{n+1} = (ai_n + c) \bmod m$ and $\{u_n\}_{n \geq 0}$ a sequence of numbers. Then points $(u_0, \ldots, u_{k-1}), (u_1, \ldots, u_k), \ldots$ will NOT tend to "fill up" homogeneously the k-dimensional space, but will lie on maximal $\sqrt[k]{m \cdot k!}$ parallel $(k-1)$-dimensional hyperplanes.

$\Rightarrow$ find a, c and m, which maximize the number of hyperplanes.



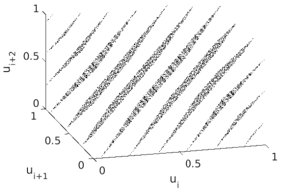


Figure 1: Histogram and spectral test of LCG: $i_{n+1} = (24298 i_n + 99991) \bmod 199017$. Maximal number of hyperplanes $\sqrt[3]{199017 \cdot 3!} \approx 106$.

Shuffling procedure

Apply shuffling procedure in order to increase the period p and to break up sequential correlations.

Generate an array $i[0], ..., i[N-1]$ filled with random numbers, i.e., $i[k] = rand() \in \{0, 1, ..., m-1\}$.

$$\begin{aligned}
\text{Initially:} \quad y &= rand() \in \{0, 1, ..., m-1\} && (5) \\
k &= \left\lfloor \frac{yN}{m} \right\rfloor \in \{0, ..., N-1\} && (6) \\
output &= i[k] && (7) \\
y &= i[k] && (8) \\
i[k] &= rand() && (9) \\
\text{GOTO} &\rightarrow (6) && (10)
\end{aligned}$$

$\Longrightarrow$ period $p = m^N$

Schrage's algorithm

Calculation of $i_n = (a \cdot i_{n-1}) \bmod m$ without overflow.

Calculate $m = a \cdot q + r$, i.e., $q = \left\lfloor \frac{m}{a} \right\rfloor$ and $r = m \bmod a$.

$$
\begin{aligned}
(a \cdot i_n) \bmod m &= (a \cdot i_n - \lfloor i_n/q \rfloor \cdot m) \bmod m && (11)\\
&= [a \cdot i_n - \lfloor i_n/q \rfloor (a \cdot q + r)] \bmod m && (12)\\
&= [a (i_n - \lfloor i_n/q \rfloor q) - r \lfloor i_n/q \rfloor] \bmod m && (13)\\
&= [a (i_n \bmod q) - r \lfloor i_n/q \rfloor] \bmod m && (14)
\end{aligned}
$$

$\Rightarrow a (i_n \bmod q) < aq < m$ and $r \lfloor i_n/q \rfloor < i_n \frac{r}{q} < i_n < m$ if $r < q$

$\Rightarrow [a (i_n \bmod q) - r \lfloor i_n/q \rfloor] \in [-m + 1, m - 1]$

$$\Rightarrow (a \cdot i_n) \bmod m = \begin{cases} a (i_n \bmod q) - r \lfloor i_n/q \rfloor & \text{if it is} \geq 0 \\ a (i_n \bmod q) - r \lfloor i_n/q \rfloor + m & \text{else} \end{cases}$$

One needs signed integer (for example: maximal $m = 2^{31} - 1$ instead of $m = 2^{32} - 1$) but avoids overflow.

Shift-Register-RNG

Works with Bit-Shift operations. L^t shifts by t bits to the left and R^s shifts by s bits to the right.

$$j_{n-1} = i_{n-1} \oplus R^s i_{n-1} \quad (15)$$
$$i_n = j_{n-1} \oplus L^t j_{n-1} \quad (16)$$

where $\oplus$ is the bitwise exclusive or (XOR). The truth table of XOR is

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

Shift-Register-RNG: Example

1	0	1	1	1	0	0	1	i_{n-1}
					s	$\longrightarrow$	$\longrightarrow$	
0	0	0	1	0	1	1	1	$R^3 i_{n-1}$
$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$	
1	0	1	0	1	1	1	0	j_{n-1}
$\longleftarrow$	$\longleftarrow$	$\longleftarrow$	$\longleftarrow$	t				
1	1	1	0	0	0	0	0	$L^4 j_{n-1}$
$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$	
0	1	0	0	1	1	1	0	i_n

Xorshift PRNG

Xorshift is state-of-the-art PRNG, it is simple and fast. Period $p = 2^m - 1$.

```
uint32_t x32 = 314159265;
uint32_t xorshift32()
{
  x32 ^= x32 << 13;
  x32 ^= x32 >> 17;
  x32 ^= x32 << 5;
  return x32;
}
```

More PRNG

- Lagged Fibonacci generator
- Mersenne twister

Non-uniform random numbers: Inverse transform sampling

So far PRNG generate uniform random numbers in $[0, 1]$.

How to generate random numbers from a given distribution $p(x)$? One possibility is the inverse transform sampling.

Cumulative distribution function,

$$F_X(x) = \Pr(X \leq x) = \int_{-\infty}^{x} dt\, p(t), \tag{17}$$

is the probability that the random variable X takes on a value less than or equal to x.

Claim: If U is a uniform random variable on $[0, 1]$ then $F_X^{-1}(U)$ follows the distribution F_X.

Inverse transform sampling:

Proof: Consider random variable $Y = F_X^{-1}(U)$.

$$\begin{aligned} F_Y(y) &= \Pr(Y \le y) = \Pr(F_X^{-1}(U) \le y) && (18) \\ &= \Pr(U \le F_X(y)) = F_U(F_X(y)) && (19) \\ &= F_X(y), && (20) \end{aligned}$$

using $F_U(x) = \Pr(U \le x) = x$ for all $x \in [0, 1]$.
$\Rightarrow$ Y and X have the same distribution.

Examples:

- Exponential distribution: $p(x) = \lambda e^{-\lambda x}$
  $\Rightarrow F(x) = 1 - e^{-\lambda x}$
  $\Rightarrow$ generate $U \in [0, 1]$ and calculate $X = \frac{-ln(1-U)}{\lambda}$
- Lorentz distribution: $p(x) = \frac{1}{\pi} \frac{\Gamma}{\Gamma^2 + x^2}$
  $\Rightarrow F(x) = \frac{1}{2} + \frac{1}{\pi} \arctan(\frac{x}{\Gamma})$
  $\Rightarrow$ generate$U \in [0, 1]$ and calculate $X = \Gamma \cdot \tan(\pi(U - \frac{1}{2}))$

Rejection sampling

Inverse transform sampling works only if $F_X(x)$ is invertible. An alternative is rejection sampling. Works if the distribution $p(x)$ fulfills: $p(x) = 0$ for $x \notin [x_0, x_1]$ and $p(x) \leq p_{max} \forall x$.

`rand()` is uniform in $[0, 1]$
Pseudo code:

```
true=1;
while (true==1)
 {x=x0+(x1-x0)*rand();
  y=pmax*rand();
  if (y<=p(x)) {true=0;}
 }
return(x);
```

`x` is distributed according to $p(x)$

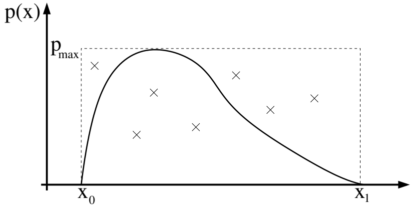


Figure 2: Only samples in the region under the graph are accepted.

Rejection sampling

It works, because p_{gen} (the distribution corresponding to `rand()`), p_{accept} (probability of acceptance a random number at x) and $p(x)$ obey

$$p_{gen}(x) = \frac{1}{x_1 - x_0} \quad \text{and} \quad p_{accept}(x) = \frac{p(x)}{p_{max}} \tag{21}$$

and, therefore, generated distribution is

$$\tilde{p}(x) = p_{gen}(x) \cdot p_{accept}(x) = \frac{p(x)}{p_{max}(x_1 - x_0)} \tag{22}$$

equal to $p(x)$ up to a normalization constant.
The method is not very efficient due to a large number of rejected random numbers. The average number of calls of `rand()` can be estimated as

$$N_{calls} = \frac{2 \cdot p_{max}(x_1 - x_0)}{\int_{x_0}^{x_1} p(x)dx} \tag{23}$$

Gaussian distribution

$$P_\sigma(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{(x-m)^2}{2\sigma^2}\right] \tag{24}$$

Central limit theorem: $u_1, u_2, ..., u_N$ are N independent and identically distributed random numbers with mean m and variance σ^2. $\Rightarrow P(x = \sum_{i=1}^{N} u_i) \stackrel{N\to\infty}{\longrightarrow} P_{\tilde{\sigma}}(x)$ with mean $\tilde{m} = Nm$ and variance $\tilde{\sigma}^2 = N\sigma^2$.
Example: choose $N = 12$ uniform random numbers $u_i \in [0,1] \Rightarrow \tilde{m} = 12 \cdot 0.5 = 6$ and $\tilde{\sigma}^2 = \frac{12}{12} = 1 \Rightarrow x = \sum_{i=1}^{12} u_i - 6$ is normally distributed.
Disadvantage: 12 random number must be generated and x have a limited range of $[-6, 6]$.

Note: A Gaussian random number x' with m and σ^2 can be generated form a Gaussian random number x with $m = 0$ and $\sigma^2 = 1$ via

$$x' = m + \sigma x \tag{25}$$

Gaussian distribution

Box–Muller method: generate two random numbers $u_1, u_2 \in [0,1]$, then the two random variables

$$\begin{aligned} x_1 &= r\cos(\varphi) = \sqrt{-2\log(u_1)}\cos(2\pi u_2) &(26)\\ x_2 &= r\sin(\varphi) = \sqrt{-2\log(u_1)}\sin(2\pi u_2) &(27) \end{aligned}$$

will both have the normal distribution ($m = 0$ and $\sigma^2 = 1$), and will be independent.
Using inversion sampling to transform u_1 and u_2 into polar coordinates r and φ leads to

$$\frac{1}{2}e^{-\frac{1}{2}r^2}\mathrm{d}(r^2)\frac{1}{2\pi}\mathrm{d}\varphi = \frac{1}{2\pi}e^{-\frac{1}{2}r^2}r\mathrm{d}r\mathrm{d}\varphi = \frac{1}{2\pi}e^{-\frac{1}{2}(x_1^2+x_2^2)}\mathrm{d}x_1\mathrm{d}x_2 \quad (28)$$

Discrete probability distribution

Finite number of states with probabilities $p_1, p_2, ..., p_N$ and $\sum_{i=1}^{N} p_i = 1$.

Production of random number via naive modification of rejection sampling.

rand() is uniform in $[0, 1]$.
Pseudo code:

```
pmax = max(p[1],...,p[N]);
true=1;
while (true==1)
 {i=1+(int) N*rand();
  y=pmax*rand();
  if (y<=p[i]) {true = 0;}
 }
return(i);
```

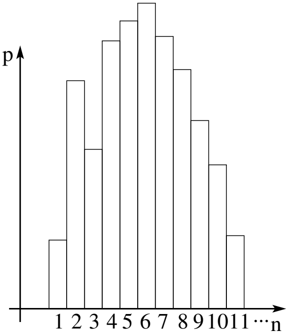


Figure 3: Discrete probability distribution.

Tower sampling

Naive rejection sampling is not efficient. Better is tower sampling, calculate cumulative sum of $p_1, p_2, ..., p_N$ as $q_j = \sum_{i=1}^{j} p_i$ ("Tower").

Pseudo code:

```
input p[1],...,p[N]
q[0]=0;
for (i=1,i<N+1,i++)
 {q[i]=q[i-1]+p[i];}
x=rand();
find j with q[j-1]<x<q[j]
return(j);
```

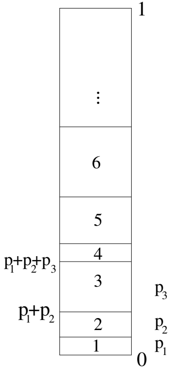


Figure 4: The "Tower".

Tower sampling: bisection method

Tower sampling needs only one random number, however, the search for index `j`, which fulfills the condition `q[j-1]<x<q[j]`, can be expensive (no free lunch theorem). An efficient search can be performed with bisection method (terminates after $\log_2(N)$ steps).

```
input x,q[0],q[1],...,q[N]
nmin=0;
nmax=N+1;
true=1;
while(true==1)
 {n=(int) (nmin+nmax)/2;
  if(q[n]<x)         {nmin=n;}
  else if(q[n-1]>x) {nmax=n;}
  else               {true=0;}
 }
return(n);
```

Simplest stochastic process: random walk

Consider a random walk on a line, which starts at 0 and at each step moves $+\delta x$ or $-\delta x$ with equal probability.

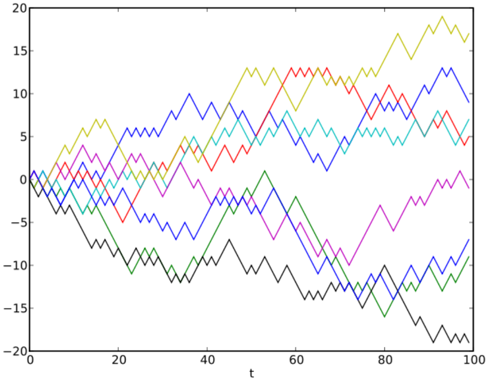


Figure 5: Independent realisations fo a random walk. Vertical axis: position x. Horizontal axis: time t

Random walk

$P(x, t)$ is the probability to find the walker at position x at time t steps and the transition probability is

$$w(x' \to x) = \begin{cases} \frac{1}{2} & \text{, if } x' = x \pm \delta x \\ 0 & \text{, else} \end{cases} \tag{29}$$

Master equation

$$\begin{aligned} P(x, t + \delta t) &= P(x, t) - \sum_{x'} w(x \to x') P(x, t) \\ &\quad + \sum_{x'} w(x' \to x) P(x', t) \qquad (30) \\ &= P(x, t) - P(x, t) + \frac{1}{2} \left[P(x - \delta x, t) + P(x + \delta x, t) \right] \end{aligned}$$

Random walk

- The position of a walker $x(t = n\delta t)$ after n steps is a stochastic variable.
- $x(t = n\delta t) = \sum_{i=1}^{n} S_i$ is a sum of n independent steps $S_i \in \{-\delta x, +\delta x\}$ with probability $\Pr(S_i = \pm\delta x) = \frac{1}{2}$.
- It is $\langle S_i \rangle = 0$ and $\langle S_i^2 \rangle = \delta x^2$.
- This leads to binomial distribution

$$P(x = k\delta x, t = n\delta t) = \frac{1}{2^n} \binom{n}{[n-k]/2}, \tag{31}$$

where $(n-k)/2$ is the number of steps to the left.
- Eq.(31) converges to a normal distribution for large n

$$\lim_{n \to \infty} P(x, t) = \frac{1}{\sqrt{2\pi D t}} \exp\left(-\frac{x^2}{2Dt}\right) \tag{32}$$

using the central limit theorem and taking the limit $\delta x, \delta t \to 0$ such that $\delta x^2 / \delta t = 2D$, where D is called diffusion coefficient.

Random walk

Random walk is a diffusion process (Brownian motion): $\langle x^2 \rangle = 2Dt$

The master equation

$$\frac{P(x, t + \delta t) - P(x, t)}{\delta t} = \frac{\delta x^2}{2\delta t} \frac{P(x + \delta x, t) - 2P(x, t) + P(x - \delta x, t)}{\delta x^2} \quad (33)$$

becomes in the limit $\delta t, \delta x \to 0$

$$\frac{\partial P(x, t)}{\partial t} = D \frac{\partial^2 P(x, t)}{\partial x^2} \quad (34)$$

the well known *diffusion equation* and Eq.(32) is its fundamental solution.

Literature

- Pierre L'Ecuyer: "Random Number Generation" In *Handbook of Computational Statistics* (pp. 35-71)
- William H. Press, Saul Teukolsky, William T. Vetterling und Brian P. Flannery: *Numerical Recipes in C. The Art of Scientific Computing*
- Edward A. Codling *et al.*: *Random walk models in biology*, J. R. Soc. Interface (2008) 5, 813–834