Monte Carlo Methods in High Energy Physics

WIESŁAW PŁACZEK

Institute of Physics, Jagiellonian University,

Cracow, Poland.

Lecture 2: Random number generators.

- **Random numbers.**
- **Uniform random number generation.**
- **Tests of random number generators.**
- **Non-uniform random number generation.**

⇒ http://cern.ch/placzek

"there is no such thing as a random number – there are only methods to produce random numbers"

John von Neumann

Random numbers

A **random number** – simply a particular value taken on by a random variable.

$\longrightarrow$ Sequence of truly random numbers – unpredictable and therefore unreproducible!

▶ **Sources of truly random numbers – physical generators:**

- $\ast$ e.g. tossing a coin, a roulette, radioactive decay, thermal noise in electronic devices (particularly "white noise"), cosmic ray arrival times, etc.

● **Drawbacks of physical generators:**

- $\ast$ too slow for typical calculational needs;
- $\ast$ problems with stability – particularly generators based on physical processes, e.g. small change in physical conditions of the source or its enviromnent can cause major changes in probabilistic properties of produced random numbers $\longrightarrow$ additional testing and bias-correcting devices needed.

▷ Old times: **Tables of random numbers** – not very practical!

$\longrightarrow$ Today coming back (?) – large and cheap storage devices (HDs, CDs, DVDs, etc.).

(1995: Marsaglia, CD-ROM 650MB of random numbers: electronic noise $\oplus$ rap music – "white & black noise")

"Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin"

John von Neumann

Pseudo-random numbers – numbers generated according to a strict mathematical formula (therefore reproducible and not at all random in the mathematical sense) but having the **appearance** of randomness, i.e. their statistical properties are very close to the ones of the truly random numbers (someone who does not know the formula is not supposed to be able to tell that a formula was used rather than a physical process).

▶ **Sources od pseudo-random numbers – mathematical generators:**

∗ good statistical properties of generated numbers,

∗ easy to use (simple, fast, convenient, . . .).

$\longrightarrow$ Dominated the Monte Carlo 'world' and made physical generators almost extinct!

This is why commonly pseudo-random numbers are called simply random numbers, and mathematical algorithms for their generation are called random number generators (RNG).

- **The first mathematical generator: 'mid-square' generator of John von Neumann:**

$\longrightarrow$ Formula: $$\boxed{X_n = \lfloor X_{n-1}^2 \cdot 10^{-m} \rfloor - \lfloor X_{n-1}^2 \cdot 10^{-3m} \rfloor \cdot 10^{2m}}$$

where: X_i, m – positive integers, X_0 – a constant, $\lfloor \cdot \rfloor$ – truncation to integer.

$\Rightarrow$ Generates $2m$-digit sequences of numbers – but short ones and dependent on X_0!

- **Typical scheme of a random number generator (RNG):**

  1) Set up initial constants: $X_0, X_1, \ldots, X_{k-1}$.

  2) If $(n-1)$ numbers have been generated, the number X_n calculate according to:

$$X_n = f(X_{n-1}, X_{n-2}, \ldots, X_{n-k}),\; n \geq k.$$

$\triangleright$ Most often one generates integer numbers or bits (0/1) $\Rightarrow$ they are converted to floating-point numbers of unifom distribution in the range $[0,1)$, denoted: $\mathcal{U}(0,1)$.

- **A period of RNG:**

  Sequences of numbers from mathematical generator – periodic sequences.

  Let P, ν – integers, a $X_0, X_1, \ldots$ – sequence of random numbers,

  P – **period of generator** (sequence) $\Leftrightarrow \exists_{\nu,P} : X_i = X_{i+jP}\, (j = 1,2,\ldots)\, \forall_{i\geq\nu}$.

  Usually, the period can be obtained theoretically (sometimes this might be difficult!).

  $\triangleright$ Requirements for the period of RNG:

  If N – the number of random numbers used in MC calculations, then:

$$N \ll P.$$

  $\longrightarrow$ In practice, it is required: $N \lesssim \sqrt{P}$.

  Popular today RNG: Mersenne Twister (Matsumoto & Nishimura, 1998): $P \approx 10^{6000}$.

Basic methods of uniform random number generation

1. **Linear Congruential Generators:**

   General formula: $X_n = (a_1 X_{n-1} + a_2 X_{n-2} + \ldots + a_k X_{n-k} + c) \bmod m$,

   where: $a_1, \ldots, a_k, c, m$ – parameters of the generator (fixed integers ≥ 0),

   $a \bmod b$ – denotes the integer modulo operation of a over b.

   Period: $P \leq m^k - 1$ (maximum period only for appropriately chosen parameters).

   ▷ Popular implementations (e.g. in Pascal, C/C++):

   $$k = 1 : \quad X_n = (aX_{n-1} + c) \bmod m$$

   ▶ Main drawback: "Marsaglia effect" – points lie on regular hyperplanes.

2. **Shift-Register Generators**:

   For bits: $b_n = (a_1 b_{n-1} + \ldots + a_k b_{n-k}) \bmod 2$,

   where: $a_1, \ldots, a_k \in \{0, 1\}$ – binary constants.

   Easy to implement, because: $(a + b) \bmod 2 = a \operatorname{xor} b$

   $\Rightarrow U_i \in \mathcal{U}(0,1)$ according to Tausworthe scheme: $U_i = \sum_{j=1}^{L} 2^{-J} b_{is+j}$, $s \leq L$.

   Period: $P \leq 2^k - 1$

   ▶ Drawback: Do not satisfy modern statistical tests!

- ▷ Generator of Tezuka (1995): Combination of 3 SR generators, $P \approx 10^{26}$, stat. OK.
- ▷ Mersenne Twister (Matsumoto & Nishimura): improved shift-register, $P = 2^{19937} - 1$.

3. **Lagged Fibonacci Generators**:

   General formula: $X_n = (X_{n-r} \odot X_{n-s}) \bmod m, \quad n \geq r,\, r > s \geq 1,$

   where the operator: $\odot \in \{+, -, \times, \mathrm{xor}\}$.

   Period: $P \leq (2^r - 1)\frac{m}{2}$

   Statistical properties: the best for: $\times$, the worst for: xor.

   ▷ Popular generator: RANMAR (Marsaglia, Zaman, Tsang):

   Combination of 2 generators, $P \approx 10^{43}$, very good statistical properties.

4. **SWB Generators** (subtract-with-borrow) – Marsaglia & Zaman (1991):

   Scheme: $X_n = (X_{n-r} \ominus X_{n-s}) \bmod m, \quad n \geq r,\, r > s \geq 1,$

   where: $$x \ominus y \bmod m = \begin{cases} x - y - c + m & \text{and } c = 1 \text{ when } x - y - c < 0, \\ x - y - c & \text{and } c = 0 \text{ when } x - y - c \geq 0, \end{cases}$$

   initially: $c = 0$.

   ▶ Drawbacks: Do not satisfy some recent statistical tests!

   ▷ Generator RCARRY (Marsaglia & Zaman, 1991):
   $P \approx 10^{171}$, simple and fast, but does not satisfy some recent tests.

5. **MWC Generators** (multiply-with-carry) – Marsaglia:

   Scheme: $X_n = (a_1 X_{n-1} + a_2 X_{n-2} + \ldots + a_r X_{n-r} + c) \bmod m,$

   where: $c = \lfloor (a_1 X_{n-1} + a_2 X_{n-2} + \ldots + a_r X_{n-r})/m \rfloor$ – the so-called carry value (to the next step).

   ▶ Advantages: Simple, fast, easy to implement, have long periods, very good statistical properties.

   ▷ Several generators proposed by Marsaglia.

6. **Non-Linear Generators** (since mid 1980s):

   ▷ Eichenauer & Lehn: $X_n = (aX_{n-1}^{-1} + b) \bmod m$

   where: c^{-1} = integer number: $c \cdot c^{-1} \bmod m = 1$, m – prime number.

   ▷ Eichenauer-Hermann: $X_n = [a(n + n_0) + b]^{-1} \bmod m$

   $\longrightarrow$ The number X_n can be obtained independently of the previous numbers.

   ▷ L. Blum, M. Blum, Shub: $X_n = X_{n-1}^2 \bmod m$; m – product of prime numbers

   $\longrightarrow$ Applications in cryptography.

   ▶ Advantages: Very good statistical properties (satisfy all known tests).

   ▶ Drawbacks: They are a bit slower than linear generators.

- **Combinations of generators – usually give better results, but not always!**

"Random number generators should not be chosen at random."

Donald Knuth

How to check whether a given generator is good or bad?

A generator is good when it produces sequences of numbers that have properties of truly random numbers. $\longleftarrow$ How to check this?

- **Traditional approach:**

Formulate some properties of the uniform random numbers, i.e. $r \in \mathcal{U}(0,1)$, and test if the sequences of numbers from the mathematical generator posses these properties.

$\longrightarrow$ But one can formulate infinite number of such properties $\Rightarrow$ infinite number of tests!

$\triangleright$ In practice, one can only prove that the generator is bad (fails some of the tests), but one cannot prove, that the generator is good (the fact that it has passed n tests does not guarantee that it will pass the $(n+1)$th test, which could actually be our problem at hand!).

Testing of generators $\longrightarrow$ negative selection: Passing some number of tests only increases our confidence to a given generator but does not assert its complete reliability.

$\triangleright$ A lot of strict tests of various kinds have been formulated to date,

$\longrightarrow$ see e.g. D. Knuth, *"The Art of Computer Programming"*, Vol. 2.

$\longrightarrow$ E.g. a battery of tests DIEHARD by G. Marsaglia (http://stat.fsu.edu/~geo/diehard.html)

– helped to eliminate many bad generators, also the physical ones.

► In fact, there is no credible reason why recurrence formulae should produce random or even random-looking numbers! $\longrightarrow$ This is really quite amazing!

- **1993: M. Lüscher** – "Finally, a theory of random number generation" (F. James)

  Martin Lüscher – theoretical physicist, specializing in lattice field theory.

  ▷ Article: hep-lat/9309020, Comput. Phys. Commun. **79** (1994) 100:

  Operational definition of randomness in the sense required for Monte Carlo calculations, based on chaotic behaviour in classical dynamical systems (theories of Kolmogorov and Arnold) – the use of Lyapunov exponents and Kolmogorov entropy to study chaotic behaviour of numbers produced by a generator.

  ► **Generator RANLUX:** based on the SWB generator RCARRY of Marsaglia & Zaman, supplemented with an algorithm of discarding some sequences of numbers

  – in order to achieve sufficiently good 'chaotic' properties of the generated numbers.

  Period: $P \approx 10^{171}$.

  $\longrightarrow$ **No departures from randomness have been found so far!**

Non-uniform random number generation

▷ Random numbers of distributions other than uniform are usually obtained from uniformly distributed random numbers by applying some transformation methods.

I. General methods

1. **Inverse transform method:**

   Let U – uniformly distributed random number over $(0,1)$, i.e. $U \in \mathcal{U}(0,1)$, and F – some continuous and **increasing** cumulative distribution function. Then the random variable
   $$X = F^{-1}(U)$$
   is distributed according to the cumulative distribution function $F(x)$.

   Proof: $\mathcal{P}[X \le x] = \mathcal{P}[F^{-1}(U) \le x] = \mathcal{P}[U \le F(x)] = F(x).$

   ▶ Generalization: If F is any **nondecreasing** function, then one should take:
   $$X = \inf\{x\colon U \le F(x)\}.$$

   EXAMPLE 1: Exponential distribution $E(0,1) \longrightarrow$ pdf: $\boxed{\rho(x) = e^{-x},\ x > 0}$.

   $\Rightarrow$ cdf: $F(x) = \int_0^x e^{-x'}dx' = 1 - e^{-x}$.

   Let $r \in \mathcal{U}(0,1)$: $\quad r = F(x) = 1 - e^{-x} \quad \Rightarrow \quad x = -\ln(1-r)$,

   If $r \in \mathcal{U}(0,1)$, then $1 - r \in \mathcal{U}(0,1) \quad \Rightarrow \quad \boxed{x = -\ln r}$.

EXAMPLE 2: Discrete distribution: $\mathcal{P}[X = k] = p_k,\ k = 0, 1, \ldots;\ \sum_k p_k = 1.$

If $r \in \mathcal{U}(0, 1)$, then $\boxed{X = \min\{k : r \leq \sum_{i=0}^{k} p_i\}}$.

▷ Algorithm in C/C++:

```
int DiscreteGen(double rn, double* p) {
// Generation of discrete distribution P{X = k} = p[k],
// rn - random number of uniform distribution over (0,1).
  int k = 1;
  double sum = p[0];
  while (sum < rn) sum += p[k++];
  return k - 1;
}
```

Limitations of inverse transform method:

- It is usually required that a cumulative distribution function is known analytically and can inverted analytically – only a small number of functions satisfy these conditions!
- In principle, one can use numerical integration of pdf or tabulated cdf (histogram) and, instead of analytical, perform numerical inversion of cdf – this is usually slower and less accurate, therefore not so often realized in practice.

2. **Rejection (hit-or-miss) method (von Neumann, 1951):**

Let $f(x)$ – our desired probability density function, x can be n-dimensional variable.

A. Find a pdf $g(x)$ for which random point generation is simple and fast (in the simplest case $g(x) =$ const) and adjust a constant $c > 0$ such that:

$$f(x) \leq c\, g(x), \; \forall_x .$$

B. Generate point X according to $g(x)$ and a random number $U \in \mathcal{U}(0,1)$.

C. If: $c\,U g(X) \leq f(X)$ – accept X, otherwise reject it and go back to step B.

▷ Alternative way:

Step C. Calculate: $w(X) = \frac{f(X)}{g(X)}$ – event weight; find maximum weight: w_{max}.

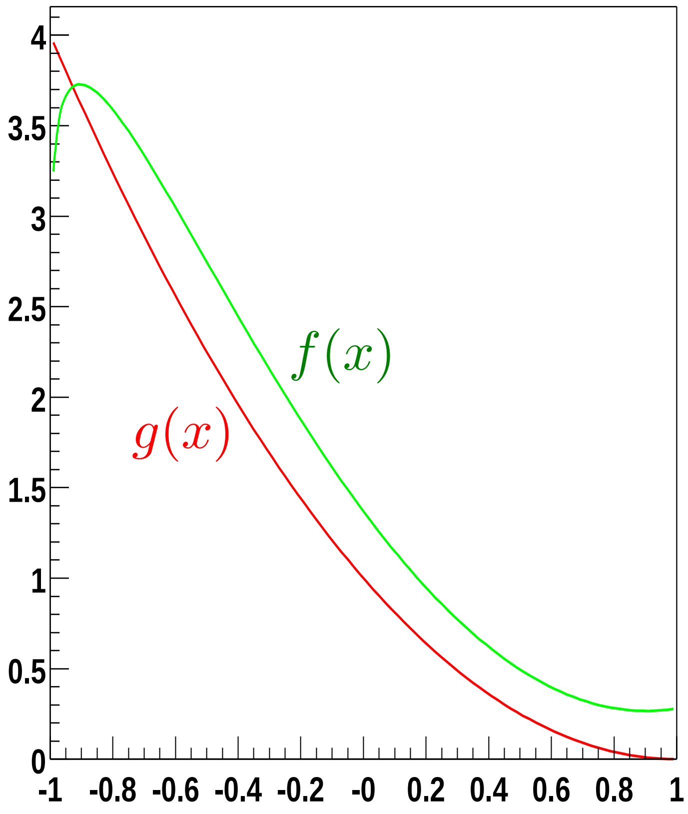
If: $w(X) \geq U w_{max}$ – accept event, otherwise reject it and return to step B.

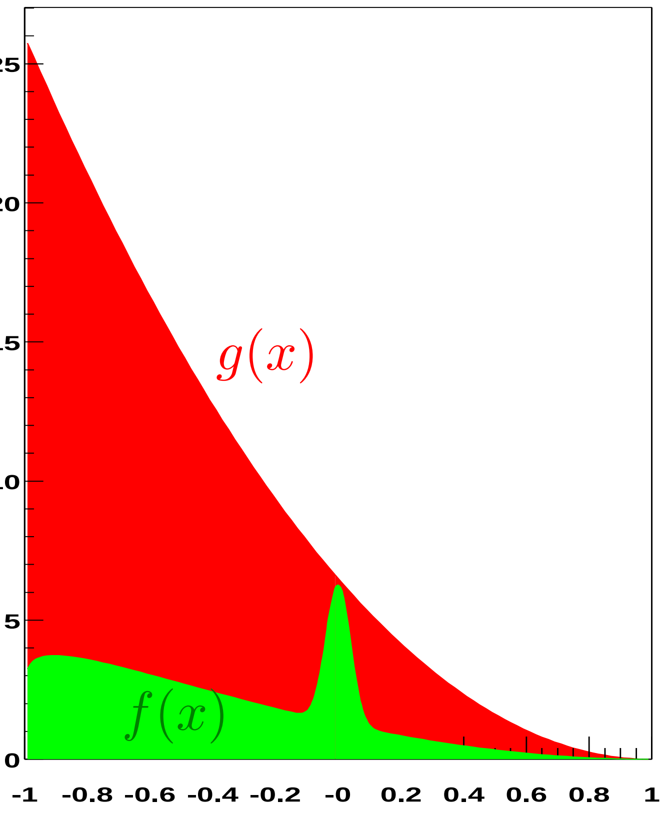
► Whenever weighted events are acceptable one can skip a rejection loop (as well as generation of auxiliary random number U); in such a case each point (event) X is accompanied with the weight $w(X)$.

Limitations of the rejection method:

- Zeros of the function $g(x)$ are dangerous if at the same time $f(x) \neq 0$!
- Spikes of $f(x)$ can degrade efficiency if they are not well approximated by $g(x)$!

Limitations of the rejection method





'Blind spots' (zeros) of $g(x)$.
$\longrightarrow$ **'infinite tail' of the weight!**

The narrow spike of $f(x)$
$\longrightarrow$ **huge rejection rate!**

3. **Composition (superposition) method (Butler, 1956):**

∗ CONTINUOUS COMPOSITION

Let X – a random variable of the probability density function $f(x)$:

$$f(x) = \int g_y(x) h(y) dy,$$

where: $g_y(x)$ – some pdf depending on the parameter y; $h(y)$ – some other pdf.

Generation scheme:

A. Generate Y according to the pdf $h(y)$.

B. For a given value Y, generate X according to the pdf $g_Y(x)$.

EXAMPLE: $f(x) = n \int_1^{+\infty} y^{-n} e^{-xy} dy, \quad x, y > 0, \; n \geq 1.$

Let's define the functions: $g_y(x) = ye^{-xy}$ and $h(y) = ny^{-(n+1)}$.

A. Random numbers Y of the pdf $h(y)$ can be generated using the inverse transform method: $Y = (1 - U)^{-1/n}$, where $U \in \mathcal{U}(0,1)$.

B. Random numbers X of the pdf $g_y(x)$ can be generated as for the exponetial distribution $E(0, \frac{1}{y})$: $X = -\frac{1}{Y} \ln V$, where $V \in \mathcal{U}(0,1)$.

∗ DISCRETE COMPOSITION

Let:

$$f(x) = \sum_{i=1}^{\infty} p_i \, g_i(x),$$

where: p_i – density of some discrete distribution, i.e. $p_i \geq 0, \; \sum_{i=1}^{\infty} p_i = 1$;

$g_i(x)$ – some continuous pdfs.

Generation scheme:

A. Generate a number i according to the density p_i, e.g. using the inverse transform.

B. For a given value i, generate X according to the pdf $g_i(x)$.

► This technique is also called a **branching method**.

EXAMPLE: Polynomial probability density functions

Let:

$$f(x) = \sum_{i=1}^{n} c_i x^i, \qquad 0 \leq x \leq 1, \; c_i \geq 0; \quad \sum_{i=1}^{n} \frac{c_i}{i+1} = 1.$$

A. Generate the index $i \in \{1, 2, \ldots, n\}$ according to the pdf $p_i = \frac{c_i}{i+1}$.

B. For a given value i generate X according to the pdf $(i+1)x^i$, e.g. using the inverse transform method: $X = U^{1/(i+1)}$, where $U \in \mathcal{U}(0,1)$.

4. **Combination of composition and rejection (Butcher, 1960):**

Let X – a random variable of the probability density function:

$$f(x) = \sum_{i=1}^{\infty} p_n f_n(x), \qquad p_n \geq 0, \ \sum_{i=n}^{\infty} p_n = 1,$$

where: $f_n(x)$ – some n-dependent pdfs.

For each f_n find a pdf $g_n(x)$ and a constant c_n $(c_n > 0)$, such that:

$$f_n(x) \leq c_n\, g_n(x)\ \forall_x.$$

Generation scheme:

A. Generate a number n according to the distribution $\mathcal{P}[n = i] = p_i$.

B. For a given value n, generate X according to the pdf $g_n(x)$.

C. Generate $U \in \mathcal{U}(0,1)$.

D. If: $c_n\, U g_n(X) \leq f_n(X)$ – accept X, otherwise reject it and return to step A.

▷ Alternative way:

Step D. Calculate: $w_n(X) = \frac{f_n(X)}{g_n(X)}$ – event weight; find maximum weight: w_n^{max}.

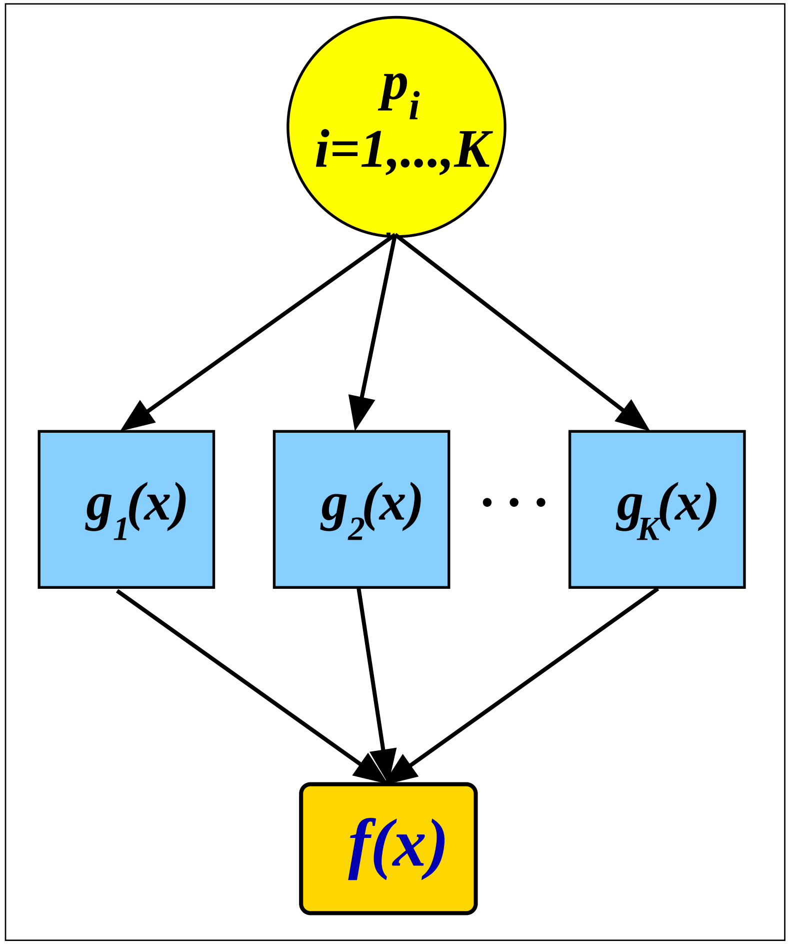
If: $w_n(X) \geq U w_n^{max}$ – accept event, otherwise reject it and return to step A.

Note: Here we need the maximum weight for each branch n independently. Their values can be estimated analytically or numerically (e.g. by histogramming the weights in a trial MC run).

Branching algorithms

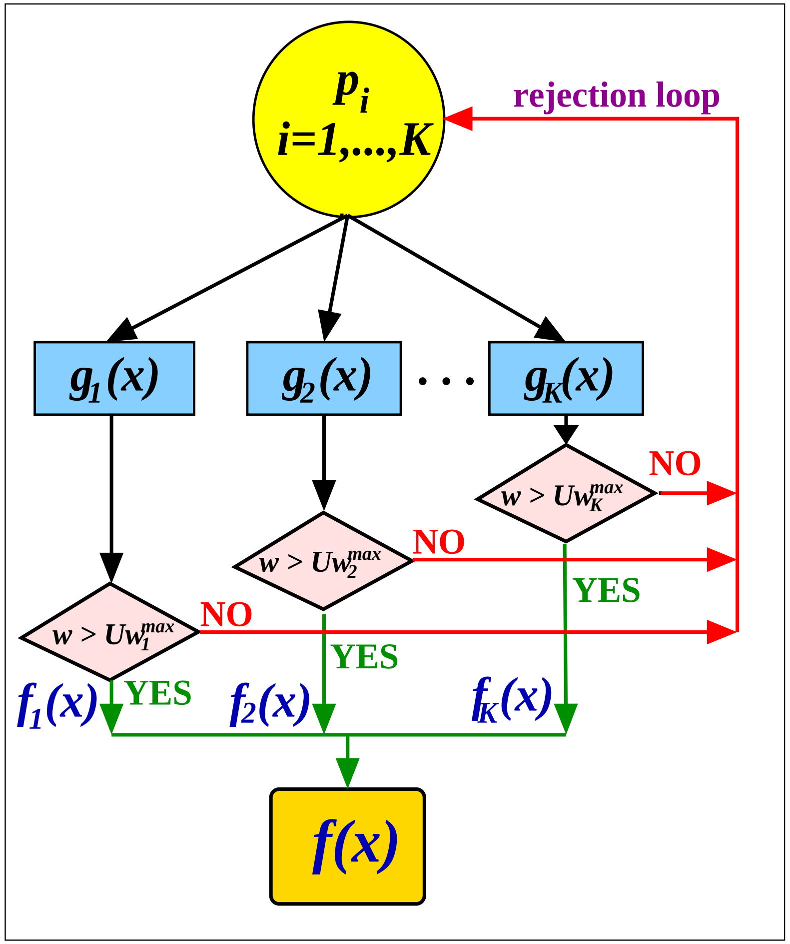
The simple branching method:

$$f(x) = \sum_{i=1}^{K} p_i \, g_i(x)$$



Combination of branching and rejection:

$$f(x) = \sum_{i=1}^{K} p_i \, f_i(x)$$



II. Random number generators for basic distributions

1. **Gaussian (normal) distribution $N(0,1)$**

$$\text{pdf:}\quad \rho(x) = \frac{1}{\sqrt{2\pi}}\, e^{-\frac{x^2}{2}}\,.$$

a) The method based on the Central Limit Theorem, see Lecture 1.

$\longrightarrow$ **Drawback:** Lack of inifinite tails of Gaussian distribution!

b) Inverse transform method:

$\triangleright$ The cumulative function of one-dimensional Gaussian distribution cannot be expressed in terms of elementary functions!

$\blacktriangleright$ Go to $\mathbf{2}$ dimensions:

$$\text{pdf:}\quad \varrho(x) = \frac{1}{2\pi}\, e^{-\frac{x^2+y^2}{2}}\,.$$

and make transformation to polar coordinates, then invert the cumulative function:

$$x = \sqrt{-2\ln r_1}\,\cos(2\pi r_2), \qquad y = \sqrt{-2\ln r_1}\,\sin(2\pi r_2)$$

where $r_1, r_2 \in \mathcal{U}(0,1)$.

$\Rightarrow$ General Gaussian distribution $N(\mu,\sigma)$: $\quad x' = \mu + \sigma x$.

2. **Exponential distribution $E(\theta,\lambda)$**

$$\text{pdf:}\;\; \rho(x) = \frac{1}{\lambda}\, e^{-\frac{x-\theta}{\lambda}}\,, \qquad x \geq \theta\,.$$

▶ Transformation: $x \to x' = \frac{x-\theta}{\lambda} \Rightarrow E(\theta,\lambda) \to E(0,1)$: $\rho(x') = e^{-x'}$, $x' \geq 0$.

▷ Inverse transform method: $x' = -\ln r,\;\; r \in \mathcal{U}(0,1)$.

3. **Cauchy (Breit-Wigner) distribution $C(\theta,\lambda)$**

$$\text{pdf:}\;\; \rho(x) = \frac{\lambda}{\pi}\,\frac{1}{(x-\theta)^2+\lambda^2}, \quad -\infty < x < +\infty\,.$$

▶ Transformation: $x \to x' = \frac{x-\theta}{\lambda} \Rightarrow C(\theta,\lambda) \to C(0,1)$: $\rho(x') = \frac{1}{\pi}\,\frac{1}{1+x'^2}$.

▷ Inverse transform method: $x' = \tan(\pi[r-\frac{1}{2}]),\;\; r \in \mathcal{U}(0,1)$.

4. **Power-law distributions**

pdfs: $\rho_1(x) = nx^{n-1},\;\; \rho_2(x) = n(1-x)^{n-1},\;\; 0 \leq x \leq 1,\; n = 1,2,\ldots\,.$

a) Inverse transform method: $x = r^{1/n},\;\; r \in \mathcal{U}(0,1)$.

b) Let: $r_1, r_2, \ldots, r_n \in \mathcal{U}(0,1)$ – independent random numbers.

$x = \max\{r_1, r_2, \ldots, r_n\}$ – is distributed according to $\rho_1(x)$,

$x = \min\{r_1, r_2, \ldots, r_n\}$ – is distributed according to $\rho_2(x)$.

5. **Binomial distribution** $b(n,p)$

pdf: $\mathcal{P}[X=m] = \binom{n}{m} p^m\,(1-p)^{n-m}, \quad 0<p<1,\; m=0,1,\ldots,n\,.$

- Algoritm 1: Rejection method (like ‘hit-or-miss’ for the Buffon’s needle problem)

```
long BinomialGen1(long n, double p) {
  double r; long m = 0;
  for (long i = 0; i < n; i++){
    r = RNG();         // random number generation
    if (r <= p) m++; }
  return m;
} //
```

Drawback: Many random numbers needed!

- Algoritm 2: Only one random number needed!

```
long BinomialGen2(long n, double p, double r) {
// r - random number of uniform distribution over (0,1)
  long m = 0;
  for (long i = 0; i < n; i++)
    if (r <= p) { m++; r /= p;}
    else r = (1 - r)/(1 - p);
  return m;
} //
```

Drawback: More floating-point operations!

6. **Poisson distribution** $P(\mu)$

pdf: $\mathcal{P}[X = k] = \frac{\mu^k}{k!}\, e^{-\mu}\,, \quad k = 0, 1, \ldots\,; \qquad E(k) = V(k) = \mu\,.$

▶ Algoritm 1: Popular in HEP applications

```
int PoissonGen1(double mu) {
  int k = -1;
  double r, s = 1.0, q = exp(-mu);
  while (s > q) { r = RNG(); s *= r; k++; }
  return k;
} //Many random numbers needed, but can be used e.g. to conctruct particles 4-momenta
```

▶ Algoritm 2: Inverse transform method

```
int PoissonGen2(double mu, double r) {
// r - random number of uniform distribution over (0,1)
  int k = 0;
  double q = exp(-mu), s = q, p = q;
  while (r > s) { k++; p *= mu/k; s += p; }
  return k;
}// Only one random number needed, but more floating-point operations!
```

Summary

- Monte Carlo calculations are based on **random numbers**.
- There are three types of random numbers: **truly random numbers** (from physical generators), **pseudo-random numbers** (from mathematical generators) and **quasi-random numbers** (special correlated sequences of numbers, used only for integration – give faster convergence than the standard MC integration).
- In real-life Monte Carlo calculations **pseudo-random** numbers are used most often.
- Use only well tested random number generators! The popular RNGs are: RANMAR, RANLUX, Mersenne Twister. Do not trust generators provided with compilers, operating system, programming-language libraries, etc.
- Having an uniform random number generator and using basic MC generation techniques one can construct a random number generator for almost any distribution.
- The art of Monte Carlo calculations is to use appropriate combinations of various generation methods in order to construct an efficient MC algorithm being solution to a given problem.