

The Monte Carlo method

A numerical technique for calculating probabilities & related things using sequences of random numbers.

The usual steps:

- generate sequence $r_1, r_2, \dots, r_m$ uniform in $]0,1[$.
- use them to produce another sequence $x_1, x_2, \dots, x_n$ distributed according to pdf $f(x)$ of interest
- use obtained x values to estimate some property of

$f(x)$, e.g. fraction of x values within $[a,b] = \int_a^b f(x)dx$

$\Rightarrow$ MC calculation sort of integration (at least formally)

Trivial for 1D: $\int_a^b f(x)dx$ obtainable by other methods, but MC more powerful for multi-dimensional problems.

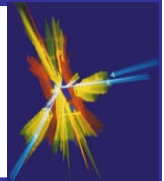
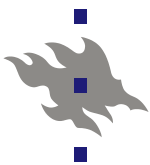
MC x values = "simulated data"

$\rightarrow$ used for testing statistical procedures

MC methods a wide field, actually own field in itself – here focus on the usage of MC for data analysis e.g. determining the statistical (& systematic) uncertainties. In such cases, MCs are used to generate different data distributions. So, let's try to answer the question "how can I generate the type of distribution I need?".

more complete & deeper discussion found in e.g.

- MATR323 "Basics of Monte Carlo simulation"-course by Prof. Flyura Djurabekova in spring 2027 (period III)



goal: to get uniformly distributed values in $]0,1[$ interval.

⇒ "random number generator"

= computer algorithm to generate $r_1, r_2, \dots, r_m$.

e.g. multiplicative linear congruential generator (MLCG)

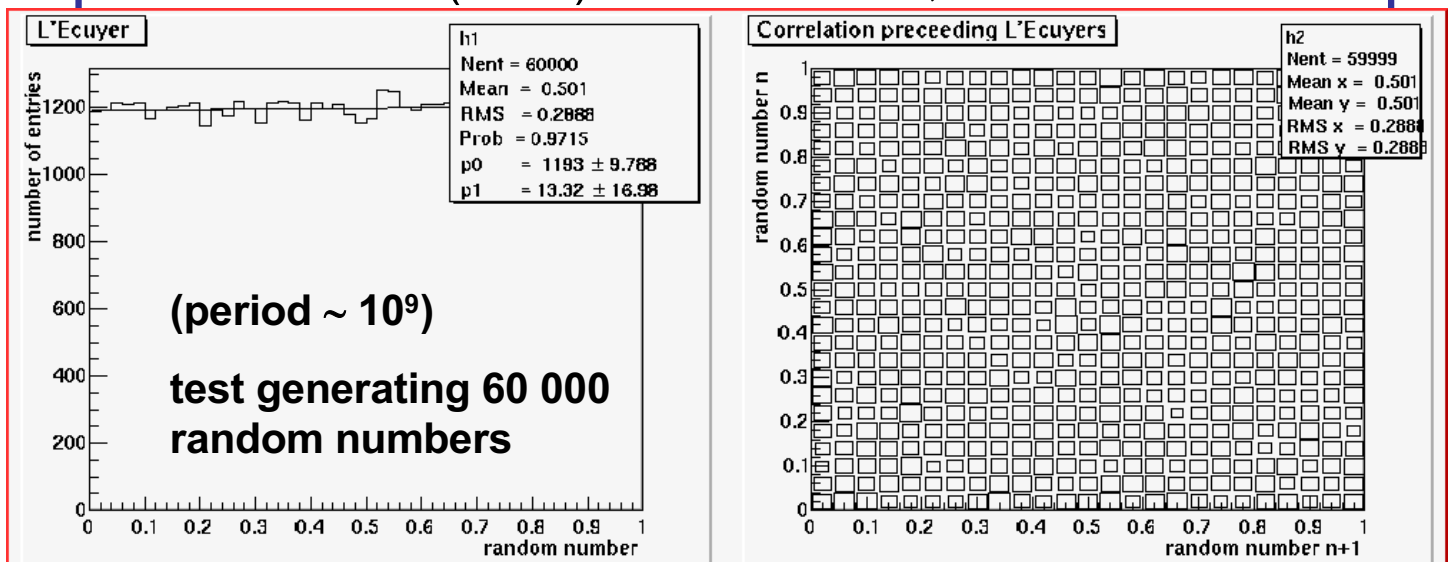
$$n_{i+1} = (an_i) \bmod m, \quad \text{where}$$

n_i = integer, a = **multiplier**, m = **modulus** & n_0 = **seed**.
(NB! mod = modulus (remainder), e.g. $27 \bmod 5 = 2$)

n_i follow periodic sequence in $[1, m-1]$ ⇒

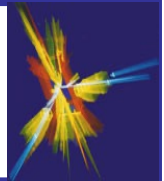
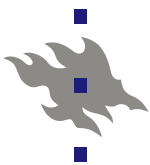
$r_i = n_i / m$ distributed in $]0,1[$.

Choose a & m so that r_i 's pass various tests of randomness: r_i 's uniform in $]0,1[$, succeeding r_i 's uncorrelated & "period" for r_i 's long (maximum = $m-1$) e.g. L'Ecuyer, *Comm. ACM* 31(1988)742: $a = 40692$, $m = 2147483399$



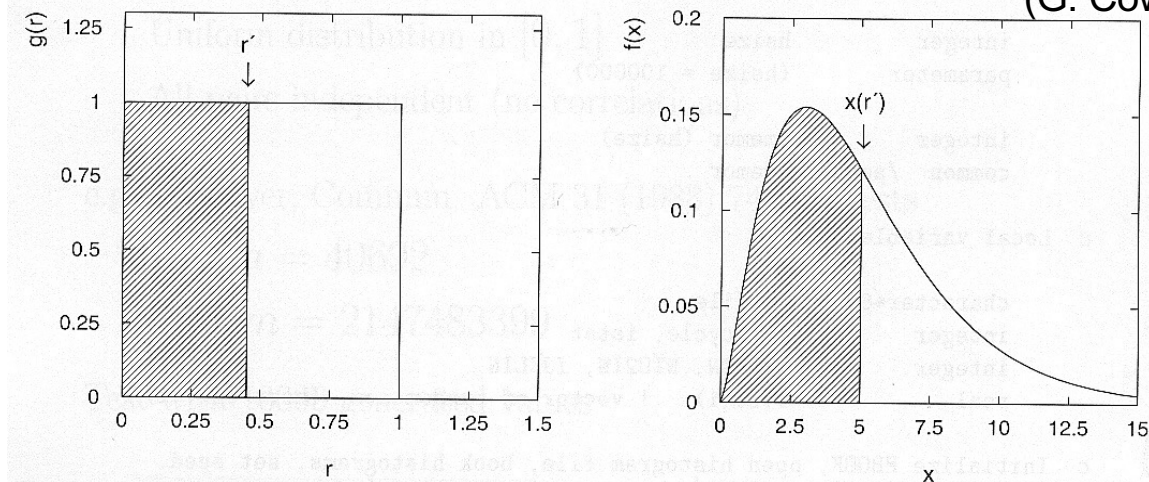
Much better algorithms exist e.g. Mersenne twister, (period $\approx 10^{6000}$) or WELL family (period up to $\approx 10^{45000}$). Many good algorithms included in free program libraries. NB! r_i 's like above in reality **pseudorandom numbers**.

See e.g. F. James, *Comput. Phys. Commun.* 60 (1990) 111



given $r_1, r_2, \dots, r_n$ uniform in $]0,1[$, find $x_1, x_2, \dots, x_n$ which follow $f(x)$ by finding a suitable transformation $x(r)$.

(G. Cowan)



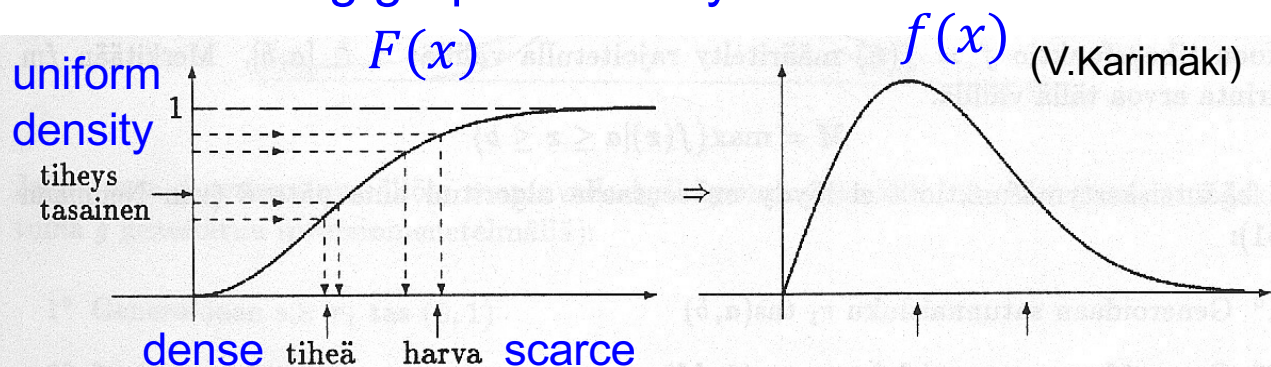
$P(r \leq r') = P(x \leq x(r'))$ i.e.

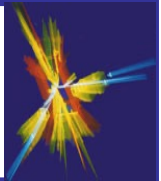
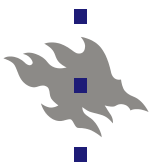
$$\int_{-\infty}^{r'} g(r) dr = r' = \int_{-\infty}^{x(r')} f(x') dx' = F(x(r'))$$

A method that always works when inverse function of cumulative distribution function $F(x)$ can be calculated or put in a table. Then **inverse transform method**:

- sample r from a uniform distribution $]0,1[$
- calculate $x = F^{-1}(r)$

then generated random numbers x that obey pdf $f(x)$. From following graph it is easy to see that it works





Discrete distributions

Inverse transform method for discrete distributions.

p_i = probability for integer i . First have to put in table cumulative distribution function $F_j = \sum_{i=0}^j p_i$, $j = 0, \dots, N$. If infinite number of possible outcomes i , N to be set so large that $F_N \approx 1$. Generation algorithm for discrete pdf:

(i) sample r from a uniform distribution $]0,1[$

(ii) find k so that $F_{k-1} < r < F_k$

(iii) accept integer $k - 1$

Resulting distribution proportional to probabilities p_i

Example of inverse transform method:

exponential pdf: $f(x; \xi) = \xi^{-1} e^{-x/\xi}$ ($x \geq 0$)

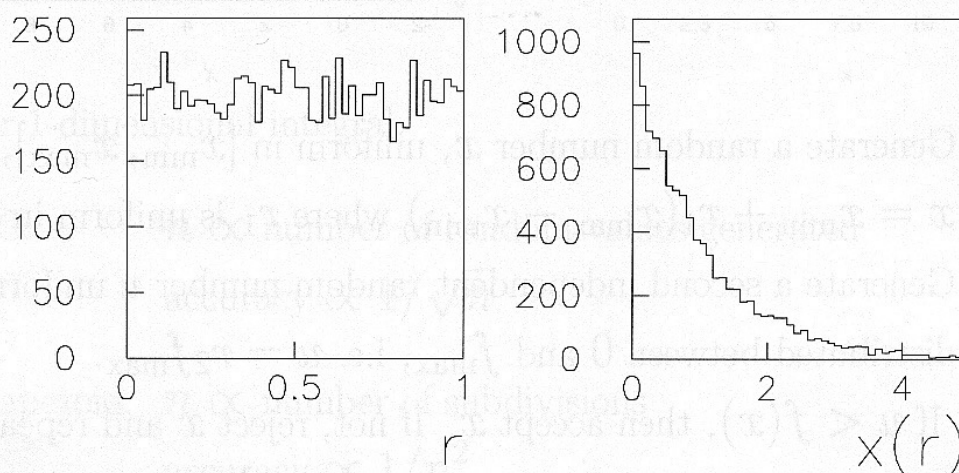
cumulative distribution function:

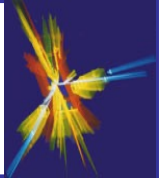
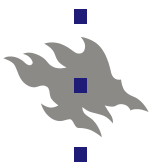
$$F(x) = \int_0^x \xi^{-1} e^{-x'/\xi} dx' = 1 - e^{-x/\xi}$$

assume $r \in]0,1[$, now set $r = F(x)$ & solve for $x(r) \Rightarrow$

$x(r) = -\xi \ln(1 - r)$ (NB! $x(r) = -\xi \ln r$ works also)

(G. Cowan)

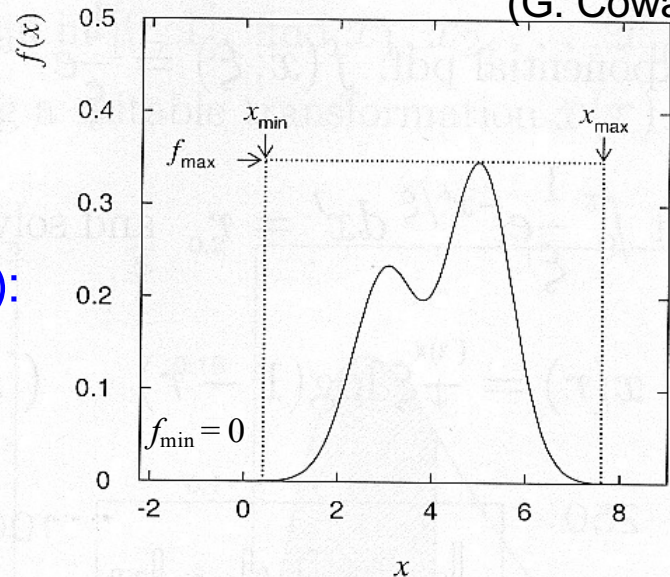




Acceptance-rejection method (von Neumann)

(G. Cowan)

Often analytic solution impossible or very impractical $\Rightarrow$ acceptance-rejection method (or **hit-or-miss**):
enclose pdf in a box
 $f_{\min} = \min(f(x))$,
 $f_{\max} = \max(f(x))$

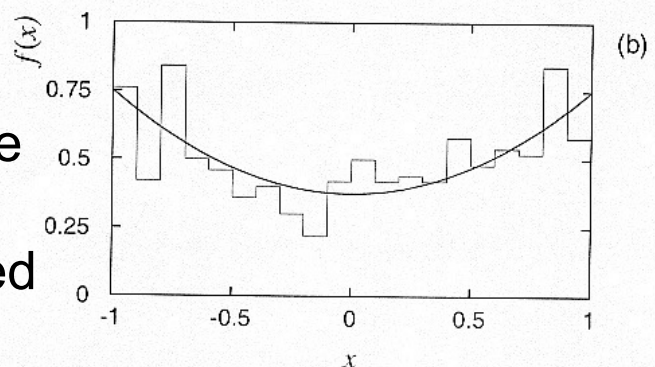
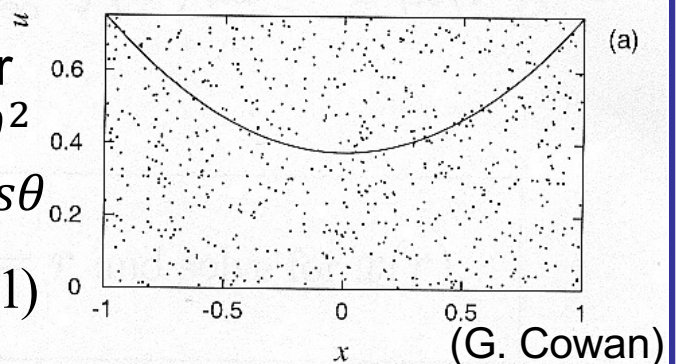


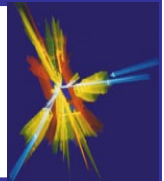
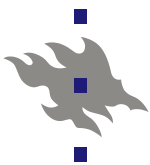
- (i) generate a random number x , uniform in $[x_{\min}, x_{\max}]$, i.e. $x = x_{\min} + r_1(x_{\max} - x_{\min})$, where r_1 uniform in $]0,1[$
- (ii) generate 2nd random number u uniformly distributed between $f_{\min}$ & $f_{\max}$, i.e. $u = f_{\min} + r_2(f_{\max} - f_{\min})$.
- (iii) if $u < f(x)$, then accept x . If not, reject x & repeat.

Example: generate a polar angle distribution $1 + \cos\theta^2$ ($-1 \leq \cos\theta \leq +1$). $x = \cos\theta$

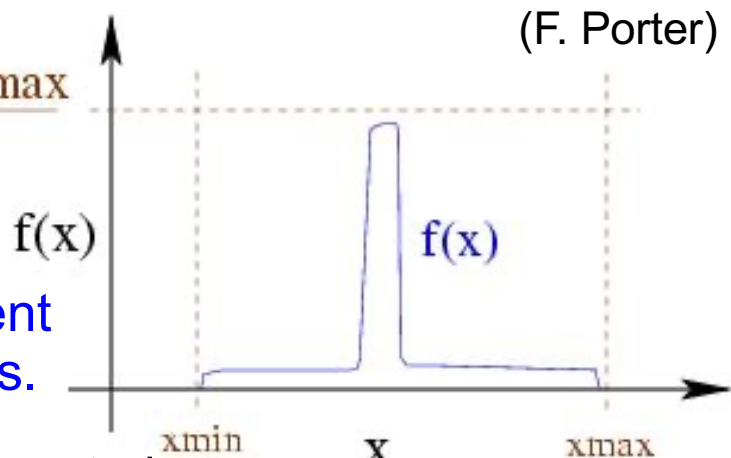
$$f(x) = \frac{3}{8}(1 + x^2) \quad (-1 \leq x \leq 1)$$

$f_{\min} = 0$, $f_{\max} = 3/4$; x values of pairs of random numbers that lie below the curve are accepted. The distribution of the accepted x values shown below.





Acceptance–rejection method simple to apply but efficiency of algorithm depends on area ratio of pdf to enclosing box. Inefficient for "peaky" distributions.



Fraction of trial points accepted:

$$\varepsilon = \frac{\text{Area under curve}}{\text{Area of box}} = \frac{\int_{x_{\min}}^{x_{\max}} f(x) dx}{(f_{\max} - f_{\min})(x_{\max} - x_{\min})}$$

Importance sampling:

To improve acceptance-rejection method efficiency, use importance sampling method. First random numbers generated according to $g(x)$ satisfying $g(x) > f(x)$ all over whole x -interval. x 's generated according to $g(x)/\int g(x)$ (with e.g. inverse transform method), and x accepted if $u < f(x)$, where u random number $]0, g(x)[$, or weighted with a factor $f(x)/g(x)$

e.g. to estimate the integral of the tail of Gaussian distribution. $P(Y > 3)$ if Y Gaussian $N(\mu = 0, \sigma = 1)$: truth ≈ 0.00135 . $h(Y_i) = 1$ if $Y_i > 3$; $h(Y_i) = 0$ if $Y_i \leq 3$

Instead of using the original Gaussian, use an equivalent Gaussian with larger acceptance rate to reduce uncertainty of the estimate and converge much faster.

- Draw an iid sample $Y_1, \dots, Y_{100}$ from a $N(0, 1)$, then the estimator is

$$\text{integral } \hat{I} = \frac{1}{100} \sum_{i=1}^{100} h(Y_i)$$

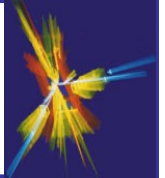
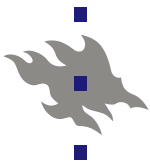
(J. Cisewski)

- Draw an iid sample $Y_1, \dots, Y_{100}$ from a $N(4, 1)$, then the estimator is

$$\hat{I} = \frac{1}{100} \sum_{i=1}^{100} \frac{h(Y_i)f(Y_i)}{g(Y_i)} \quad \text{importance sampling}$$

where f is the density of a $N(0, 1)$ and g is the density of $N(4, 1)$

$N = 10^5$	Expected Value	Variance
Truth	0.00135	0
Monte Carlo	0.00136	1.3×10^{-5}
Importance Sampling	0.00135	9.5×10^{-8}



MC methods often used to "mimic" data in e.g. particle physics, a two-step process: simulation of physics, "**event generator**" & simulation of response of experimental apparatus, "**detector simulation**".

MC event generator:

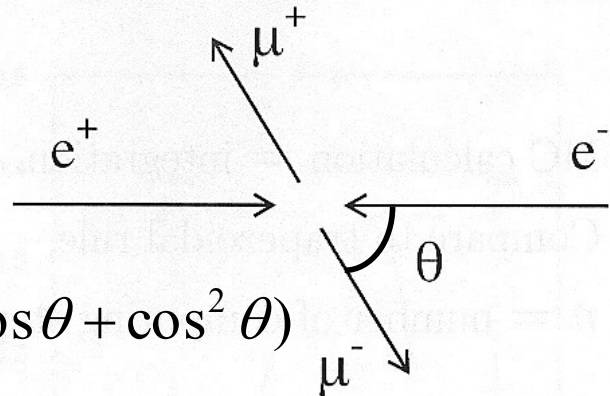
simple example:

$$e^+e^- \rightarrow \mu^+\mu^-$$

generate θ and ϕ :

$$f(\cos \theta; A_{FB}) \propto (1 + \frac{8}{3} A_{FB} \cos \theta + \cos^2 \theta)$$

$$g(\phi) = 1/2\pi$$



In reality implemented into program packages that accounts for all (known) effects (hadronisation, initial and final state radiation, longlived particles etc...)

MC detector simulation (built on e.g. GEANT4):

Input: particle list & momenta from event generator

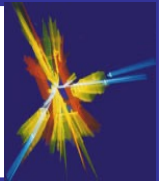
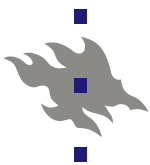
simulate particle interactions with detector material & detector response: multiple Coulomb scattering (generate particle scattering angle), ionization energy loss (generate energy loss dE/dx), electromagnetic & hadronic showers, produce detector signals, electronics response ...

Output: simulated "raw" data $\rightarrow$ input to reconstruction

Usage:

Predict what should be seen at "detector level" given a hypothesis at "generator level". Compare with real data.

- optimize measurement & experiment sensitivity.
- estimate efficiency & purity (expected signal & background)
- simulate measurement many times ("gedanken" experiment)



Standard Gaussian distributed random numbers

cumulative distribution function $F(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-x'^2/2} dx'$

Square evaluation method: Generate numbers in polar coordinates & afterwards transform them to cartesian

$$f(x)f(y)dxdy = re^{-r^2/2}dr \frac{d\phi}{2\pi} = g(r)h(\phi)drd\phi$$

r & ϕ generated according to $g(r)$ & $h(\phi) \Rightarrow$

x & y independent Gaussian distributed random numbers.

Marsaglia polar method: very efficient algorithm for generating Gaussian random numbers

u_1 & u_2 uniform in $]0,1[$

construct $v_1 = 2u_1 - 1$ & $v_2 = 2u_2 - 1$ (uniform in $]-1,1[$),

if $r^2 = v_1^2 + v_2^2 > 1$ start over again, otherwise

$$z_1 = v_1 \sqrt{-2 \ln r^2 / r^2} \quad \text{and} \quad z_2 = v_2 \sqrt{-2 \ln r^2 / r^2}$$

z_1 & z_2 independent Gaussian numbers with $\mu = 0$ & $\sigma = 1$

$z_i' = \mu + \sigma z_i$ Gaussian numbers with mean μ & variance σ^2

Poisson random numbers: efficient algorithm for small ν :

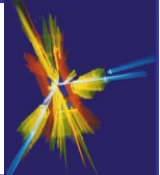
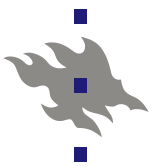
Set $k = 1$ & $A = 1$ at start, then iterate until successful choice

(i) generate u (uniform in $]0,1[$), replace A with uA

(ii) if $A < \exp(-\nu)$, where ν mean of Poisson distribution, accept $n_k = k-1$ & stop.

(iii) replace k by $k+1$ and repeat (i).

for large ν ($> \sim 10$), faster & easier to generate Gaussian random numbers since Poisson distribution $\approx$ Gaussian.



χ^2 random numbers: for even n , generate $n/2$ uniform random numbers u_i ; then $y = -2 \ln \left(\prod_{i=1}^{n/2} u_i \right)$ follows $\chi^2(n)$.
for odd n , generate $(n-1)/2$ uniform numbers u_i & one Gaussian z ; then $y = -2 \ln \left(\prod_{i=1}^{(n-1)/2} u_i \right) + z^2$ follows $\chi^2(n)$.

Binomial random numbers: principle same as for any discrete distribution. Use inverse transform method and tabulate cumulative distribution $F(x)$
Most computer libraries include generators for most common distributions like Gaussian, Poisson, χ^2 etc...

Precision of Monte Carlo methods:

MC = "integration".

compare to trapezoidal rule,
 n = # of computing steps

for 1D integral: MC: $n \propto$
number of accepted random
values, precision $\propto 1/\sqrt{n}$

trapezoid: $n \propto$ number of
subdivisions, precision $\propto 1/n^2$

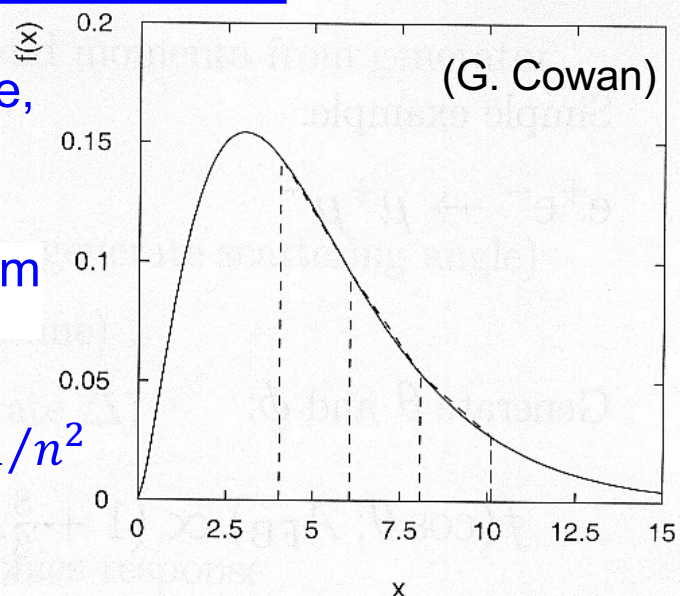
in 1D trapezoid wins!!!

but in d dimensions:

MC: precision $\propto 1/\sqrt{n}$

trapezoid: precision $\propto 1/n^{2/d}$

MC wins for $d > 4$. Gaussian quadrature better than trapezoid but for high enough d , MC always wins!!
(see e.g. F. James, *Rep. Prog. Phys.* 43 (1980) 1145).



← independent of d !