Lecture 12

Monte Carlo Techniques

Monte Carlo

- many applications
  - modeling e.g. cars
  - modeling detectors
    - CERN GEANT
      - graphical interface
      - particle interactions
      - material properties
      - High-energy physics, nuclear physics, astrophysics
      - Philips medical detector modeling
      - /home/henkjan/Hisparc/sincpr14xz03.gif

/home/henkjan/Hisparc/trafix14prhme.gif

summary: particle data group, PRD,
http://pdg.web.cern.ch/pdg

Monte Carlo technique:

- multi-dimensional integrals
  - complicated boundaries
  - large grids needed
- problems with many parameters
  - e.g. 20 variables, 10 grid points : 10^{20} calculations
  - with Monte-carlo: 10^{6} calculations gives about 0.1-1% accuracy.

Random number generators

- System-supplied (ANSI C, C++)
  - #define RAND_MAX ....
  - void srand(int seed) – start random sequence
  - int rand()
  - returns integer on interval [0,Rand_max-1]
- Rand_max is typically 2^{32}, but sometimes much smaller (like 32767). This is allowed in ANSI C.
- pseudo-random range. However, no standard on how long this range is. To be tested/avoided.
- typically, one uses *linear congruential* generators:

$$I_{j+1} = (aI_j + c)\%m$$

Linear congruential generators

- typically sequence of RAND_MAX numbers. The constants a, c, and m should be chosen with care, else the sequence is shorter, or there will be correlations between consecutive numbers. E.g. least significant bit alternates, or very small numbers are followed by small numbers.
- seed: I0. Same seed will give same sequence: program can be repeated.
- if implementation of rand() returns 2-byte integer: reject it.
- random values in small range : (e.g. 1-10)

$$j = 1 + \mathrm{int}(10.0 * rand() / (RAND\_MAX + 1));$$

$$wrong \quad j = 1 + (rand()\%10)$$

Linear congruential generators

- sequential correlations on different calls:
  - k numbers are not uniformly distributed in k-dimensional space, but tend to lay on (k-1)-dimensional planes. There will be most about m1/k of such planes.
- Parks and Miller:

$$I_{j+1} = aI_j \qquad a = 7^5 = 16807 \qquad m = 2^{31} - 1$$

  - good minimal standard. period: 2^{31}-2. No 0 generated.
  - only 1 multiplication involved: extremely fast.
  - very small numbers are followed by a small number.
  - better with additional shuffle: NR function ran1()

Larger sequences:

- L'Ecuyer: combination of two random generators with slightly different seeds. Add the numbers and divide by either of the moduli.

$$m_1 = 2147483563 = 2\,x\,3\,x\,7\,x\,631\,x\,81031 + 1$$
$$m_2 = 2147483399 = 2\,x\,19\,x\,1031\,x\,1789 + 1$$
$$period = 2.3\,10^{18}$$

- Nowadays: RANLUX library (CERN). Based on chaos theory, (James, Comp. Phys. Comm. 79, 111; Luescher, Comp. Phys. Comm. 79, 100). Different quality levels, trading off speed and quality.
- Alternatively: Fibonacci-based generator by Marsaglia, Zaman, Tsang (10^{43} period) (CERNLIB)
- Passes DIEHARD battery of tests.

Probability density function

- random numbers gnerated in [0,1]. If p.d.f f(x) on the range x [-inf,inf], then the integrated probability up to a = F(a) runs from 0 to 1.
- generate u between 0 and 1:

$$u = F(x)$$

$$x = F^{-1}(u)$$

Probability density function

$$u = F(x)$$

$$x = F^{-1}(u)$$

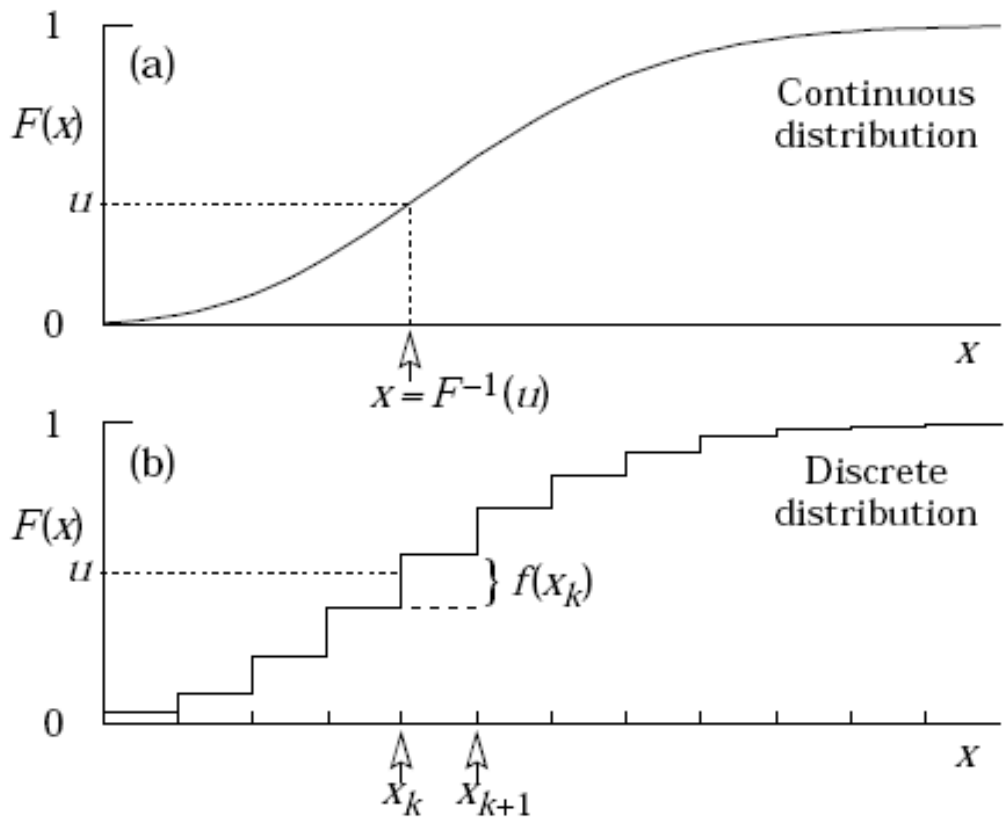


Figure 33.1: Use of a random number u chosen from a uniform distribution (0,1) to find a random number x from a distribution with cumulative distribution function $F(x)$.

Exponential deviates

- e.g. time between clicks of a Geiger-Mueller counter (random Poisson-distributed events).

$$y(x) = -\ln(x) \qquad x \;\; normal \;\; deviate \;\; on \;\; [0,1]$$

$$p(x)dx = \begin{cases} dx & 0 < x < 1 \\ 0 & otherwise \end{cases}$$

$$p(y)dy = \frac{dx}{dy}dy = e^{-y}dy$$

- generate decays between t1 and t2 according to f(t)=exp(-t/tau):

$$r_2 = e^{-t_2/\tau} \quad ; \quad r_1 = e^{-t_1/\tau}$$

$$t = -\tau \ln(r_2 + u(r_1 - r_2))$$

Isotropic direction

3 dimensions:

$$d\Omega = d\cos\theta d\phi$$

$$\cos\theta = 2u_1 - 1$$

$$\phi = 2\pi u_2$$

$$r \quad from \quad r^2 dr:$$

$$r = \sqrt[3]{y}, \quad dy = 3r^2 dr$$

sine and cosine in 2 dimensions:

$$v_1 = 2u_1 - 1$$

$$r^2 = v_1^2 + u_2^2 \quad \begin{cases} r^2 < 1: & \begin{array}{c} \sin = 2v_1 u_2 / r^2 \\ \cos = (v_1^2 - u_2^2)/r^2 \end{array} \\ r^2 > 1: & \text{draw new u's} \end{cases}$$

Gaussian distribution

- draw u1 and u2

$$\left.\begin{aligned} v_1 &= \sin 2\pi u_1 \sqrt{-2\ln u_2} \\ v_2 &= \cos 2\pi u_1 \sqrt{-2\ln u_2} \end{aligned}\right\} \text{ independent, mean 0 variance 1}$$

$$v_3 = \mu + \sigma v_i \quad \text{mean } \mu \text{ variance } \sigma$$

- Numerical recipes 3: #include deviates.h, needs ran.h and gamma.h. NormalDev x(mean,sigma,seed); // seed preferably a negative integer
- more distributions (chi^2, Poisson, Gamma, Binomial, Student's t etc.) see particle data group summary.

Rejection method

- p(x)dx computable, but inverse not known.

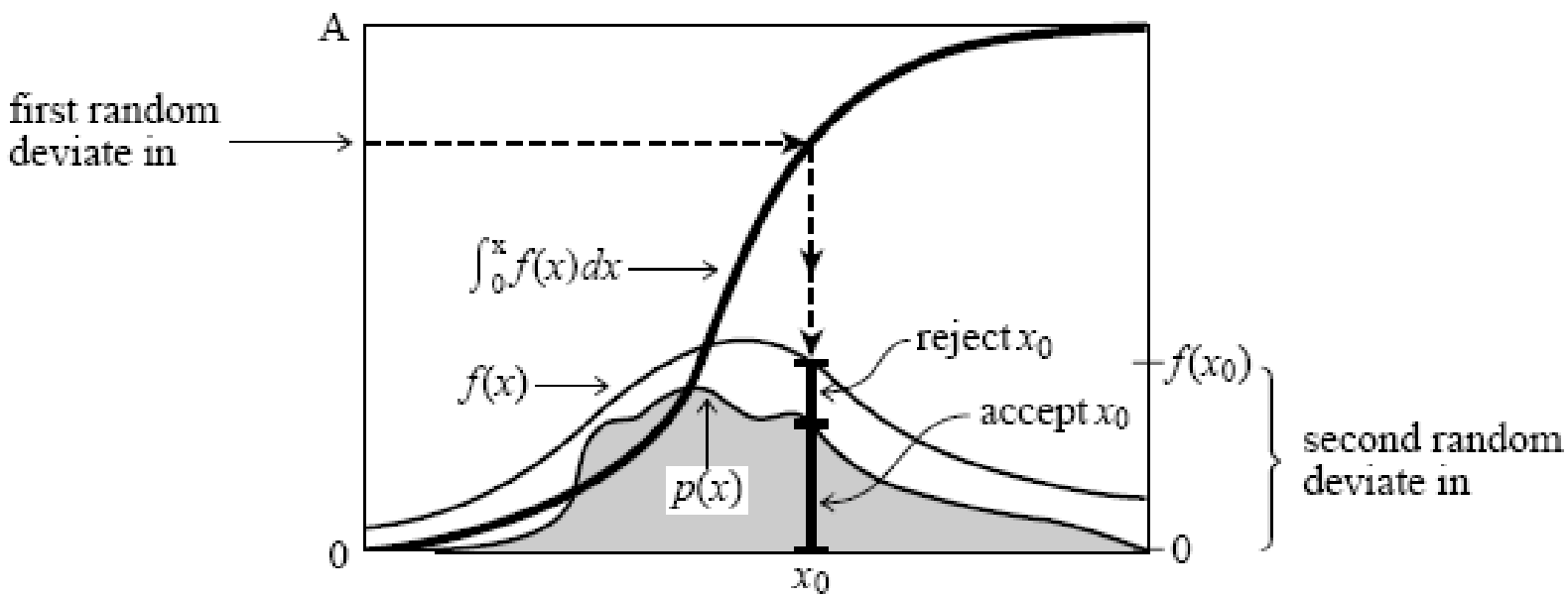


Figure 7.3.1. Rejection method for generating a random deviate x from a known probability distribution $p(x)$ that is everywhere less than some other function $f(x)$. The transformation method is first used to generate a random deviate x of the distribution f (compare Figure 7.2.1). A second uniform deviate is used to decide whether to accept or reject that x. If it is rejected, a new deviate of f is found; and so on. The ratio of accepted to rejected points is the ratio of the area under p to the area between p and f.

Monte Carlo integration

- N random points, uniformly distributed in volume V.

$$\int f dV \approx V < f > \pm V \sqrt{\frac{< f^2 > - < f >^2}{N}}$$

$$< f > \equiv \frac{1}{N} \sum_{i=0}^{N-1} f(x_i) \quad ; \quad < f^2 > \equiv \frac{1}{N} \sum_{i=0}^{N-1} f^2(x_i)$$

- uncertainty: only as indication
- Note: f constant: no uncertainty (but in that case, you did the integral analytically)

Torus

- complicated boundary may be done via Monte Carlo

$$z^2 + \left(\sqrt{x^2 + y^2} - 3\right)^2 \le 1$$

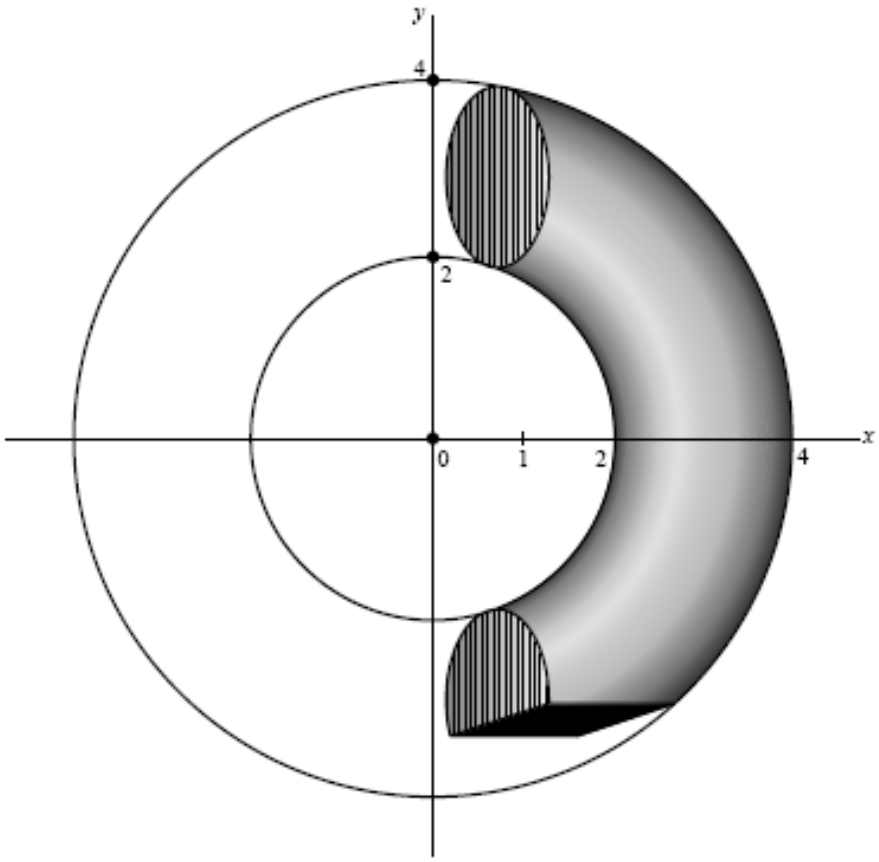
$$x \ge 1$$

$$y \ge -3$$

density: $\int \rho dxdydz;$

Inertial momenta: $\int x\rho dxdydz;$

$\int y\rho dxdydz;$ $\int z\rho dxdydz$



Integrand: change of variables

- if the integrand varies rapidly, you need many points at the small part of the volume that dominates the integral. Change of variables are important
- often: exponential sampling.

density: $\int \rho dxdydz;$

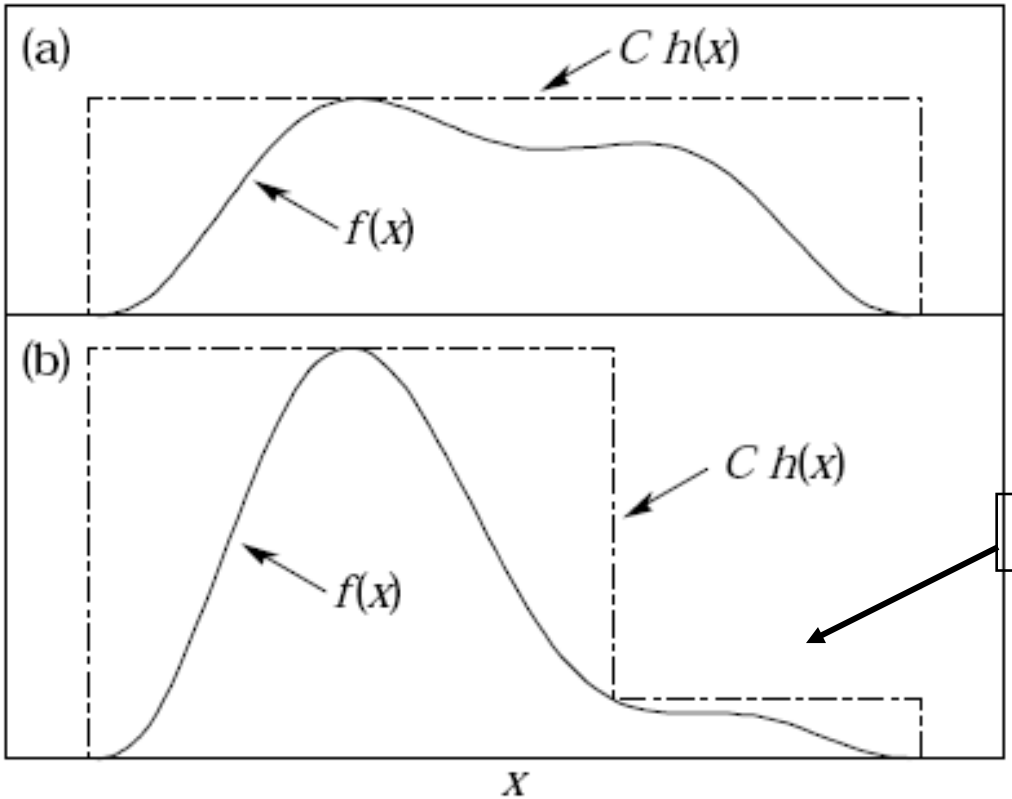
$$\rho = e^{5z}$$

$$ds = e^{5z}dz; \quad s = \frac{1}{5}e^{5z}; \quad z = \frac{1}{5}\ln(5s)$$

$$-1 < z < 1 \rightarrow .00135 < s < 29.682$$

Rejection method

- p(x)dx computable, but inverse not known.



importance sampling

Stratified sampling

- Divide volume in more parts. Variance of total integral will be less than the variance of the sum of the parts, if the number of sample points is chosen optimally.
- If the average value of the integrands does not change notably in the volumes A and B, no gain is made

Stratified sampling

$$<<f>> \equiv \frac{1}{V}\int f dV \quad ; \quad <f> \equiv \frac{1}{N}\sum_{i=0}^{N-1} f(x_i)$$

$$Var(<f>) = Var(f)/N$$

$$<f>' \equiv \frac{1}{2}(<f>_a + <f>_b)$$

$$Var(<f>') = \frac{1}{4}\left(Var(<f>_a) + Var(<f>_b)\right) =$$

$$= \frac{1}{4}\left(\frac{Var_a(<f>)}{N/2} + \frac{Var_b(<f>)}{N/2}\right) = \frac{1}{2N}\left(Var_a(<f>) + Var_b(<f>)\right)$$

$$Var(f) = \frac{1}{2}\left[Var_a(f) + Var_b(f)\right] + \frac{1}{4}\left(<<f>>_a - <<f>>_b\right)^2$$

different number of sample points:

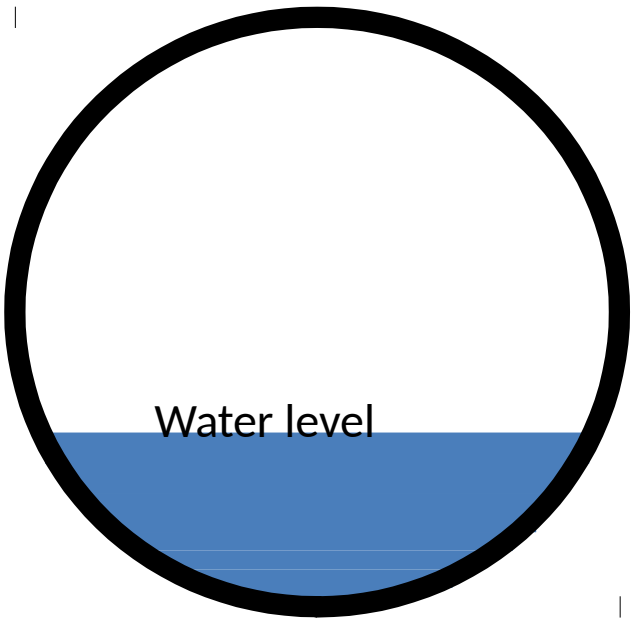
$$Var(<f'>) = \frac{1}{4}\left(\frac{Var_a(<f>)}{N_a} + \frac{Var_b(<f>)}{N_b}\right)$$

$$\frac{N_a}{N} = \frac{\sigma_a}{\sigma_a + \sigma_b} \quad with \quad \sigma_a = \sqrt{Var_a(<f>)}$$

$$Var(<f'>) = \frac{(\sigma_a{}^2 + \sigma_b{}^2)}{4N}$$

Monte Carlo example

- Integrate the center of mass of a hollow Al sphere partially filled with liquid.
- Try drawing from dr^{3}dcostheta, from dr and dtheta, and from dx,dy,dz



$$r_{in} = 0.98$$
$$r_{out} = 1.00$$
$$\rho_{shell} = 2.7$$
$$z_{level} = -0.6$$
$$\rho_{liquid} = 1$$

$$mass = 4\pi \int_{r_{\min}}^{r_{\max}} \rho_{shell} r^2 dr + \pi \int_{-r\min}^{z_{level}} \rho_{liquid} (r_{\min}^2 - z^2) dz$$

$$z_{COM} = \frac{\pi}{mass} \int_{-r\min}^{z_{level}} \rho_{liquid} z (r_{\min}^2 - z^2) dz$$

- Code on web, sphere.cpp output
- Multiply with volume, e.g 4/3pi R^3 when drawing from d(r^3), d(cos(theta)) d(phi), and 4R pi^2 when drawing from dr,dtheta,dphi

Random walk example

- Random walk: place 2 persons in a 10x10 square grid. Let them walk randomly around until they reach the same square. How long would that take on average? Take average start positions.
- Output: Ntrials: 1000000

  average number of steps : 165.74 RMS : 169.059

- /home/henkjan/WINDOWS/CompMeth/CompMeth2014/MCexample/randomwalk.cpp

Monte Carlo, summary

- Random numbers from linear congruential series: not really random
  - Many bugged versions, use a decent one like the numerical recipes one
  - Don't use even/odd random integer: many random number algorithms produce series that alternate in the last bit (so the last bit is not random at all)
- Choose right variables: try to make the integrand flat (e.g. draw from $dy=dr^3$)
- Normalization: the result is multiplied with the full detection volume, see previous lecture
- If function is strongly peaked, try to draw more numbers in the dominating region -> priority sampling, or try to get part of the behavior of the function in the integrand (e.g. draw from $dy = de^z$)
- Deviates in deviates.h, random numbers in ran.h
- Optimal case: uncertainty ~ square root of random events drawn.
- Typically: draw parameters from a pdf, e.g. the Boltzmann distribution for velocity of molecules in gas: Maxwell-Boltzmann distribution
- http://en.wikipedia.org/wiki/Maxwel%E2%80%93Boltzmann_distribution

Example 3d velocity gas

```
#include "nr3.h"
#include "ran.h"
#include "gamma.h"
#include "incgammabeta.h"

int main() {
        Chisqdist chi(3);
        Ran ran(1234567);
        for (int i=0; i<1000000; i++) {
                double vsqr=chi.invcdf(ran.doub());
                vsqr=sqrt(vsqr);
                cout << vsqr << endl;
        }
}
```

```
root [0] TH1D hist("hist","",1000,0,5);
root [1] ifstream infil("x2.txt")
root [2] for (int i=0; i<1000000; i++) { double tmp; infil>>tmp; hist.Fill(tmp);}
root [3] hist.Draw()
root [4] TF1 fit("fit","[0]*x*x*exp(-x*x/2)")
root [5] hist.Fit("fit")

 NO.  NAME     VALUE          ERROR          SIZE      DERIVATIVE
  1  p0        3.98572e+03   3.98765e+00   6.01993e-02  -4.24914e-11
(Int_t)(0)
```

Generated a million events according to the chi-squared distribution with 3 d.o.f. Stored the results in a histogram. Compared to a fit of $x^2\exp(-x^2/2)$ (the fitted line). Excellent agreement up to highest x.
Ofcourse, v must be normalized to kT and the hydrogen atomic mass