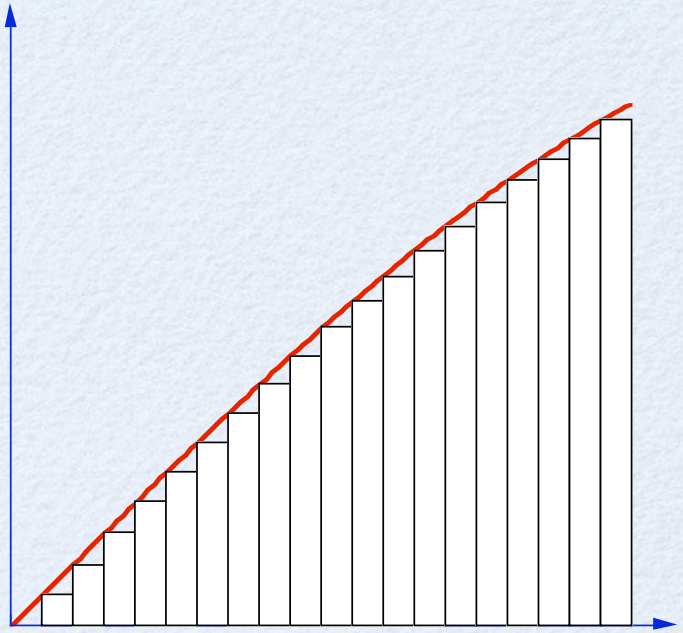
HPCSE I

Monte Carlo integration

Integrating a function

- Convert the integral to a discrete sum

$$\int_a^b f(x)dx = \frac{b-a}{N}\sum_{i=1}^{N} f\left(a + i\frac{b-a}{N}\right) + O(1/N)$$



- Higher order integrators:
  - Trapezoidal rule:

$$\int_a^b f(x)dx = \frac{b-a}{N}\left(\frac{1}{2}f(a) + \sum_{i=1}^{N-1} f\left(a + i\frac{b-a}{N}\right) + \frac{1}{2}f(b)\right) + O(1/N^2)$$

  - Simpson's rule:

$$\int_a^b f(x)dx = \frac{b-a}{3N}\left(f(a) + \sum_{i=1}^{N-1}(3-(-1)^i) f\left(a + i\frac{b-a}{N}\right) + f(b)\right) + O(1/N^4)$$

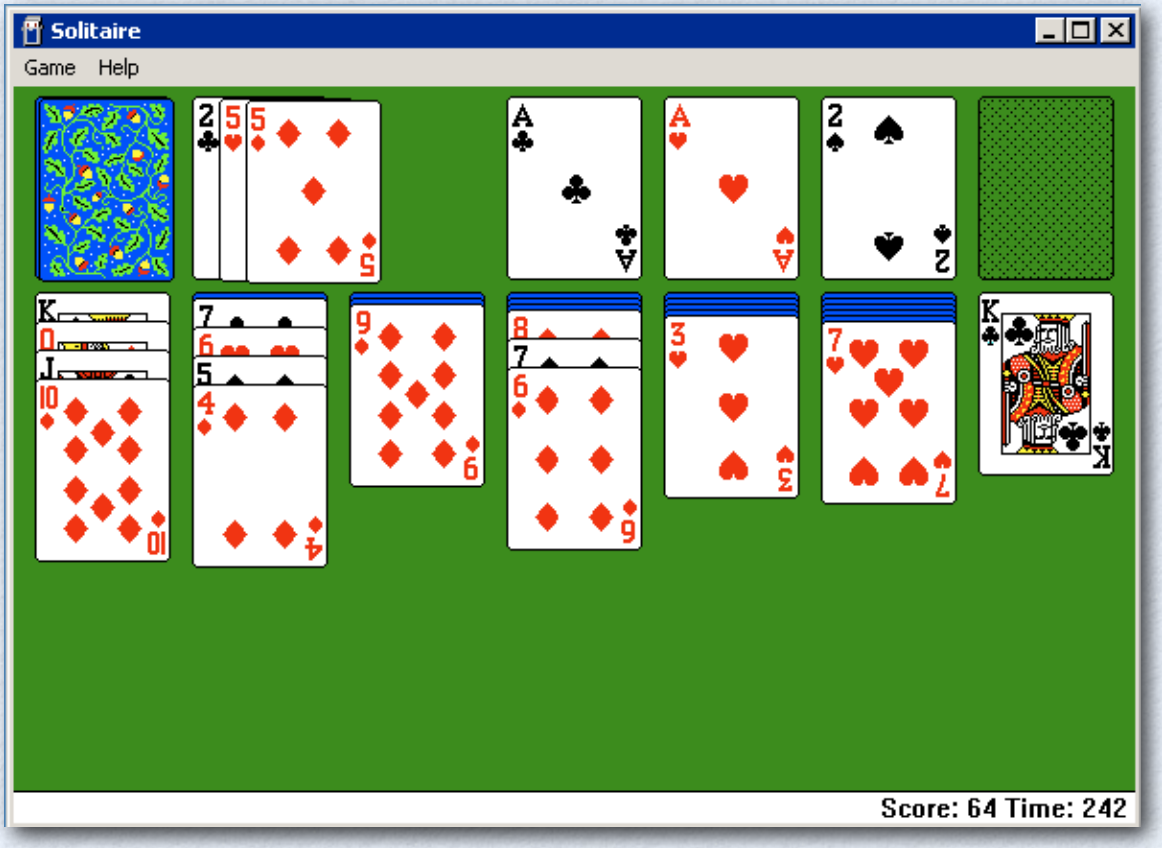
High dimensional integrals

- Simpson rule with *M* points per dimension
  - one dimension the error is $O(M^{-4})$
  - *d* dimensions we need $N = M^d$ points
    the error is order $O(M^{-4}) = O(N^{-4/d})$

- An order-*n* scheme in 1 dimension is order-*n/d* in *d* dimensions!
- In a statistical mechanics model with *N* particles we have 6*N*-dimensional integrals (3*N* positions and 3*N* momenta).
- Integration becomes extremely inefficient!

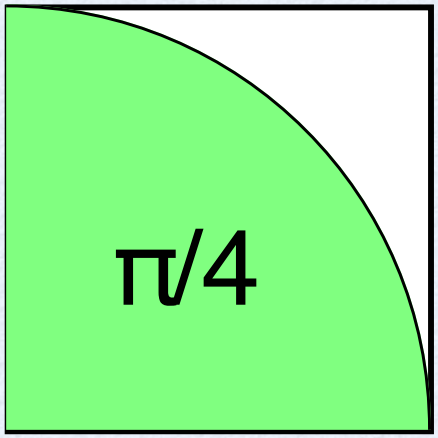
Ulam: the Monte Carlo Method

- What is the probability to win in Solitaire?
  - Ulam's answer: play it 100 times, count the number of wins and you have a pretty good estimate



Throwing stones into a pond

- How can we calculate π by throwing stones?
- Take a square surrounding the area we want to measure:



- Choose *M* pairs of random numbers (*x, y*) and count how many points (*x, y*) lie in the interesting area

Monte Carlo integration

- Consider an integral $$\langle f \rangle = \int_\Omega f(\vec{x}) d\vec{x} \Big/ \int_\Omega d\vec{x}$$
- Instead of evaluating it at equally spaced points evaluate it at M points x_i chosen randomly in Ω:

$$\langle f \rangle \approx \frac{1}{M} \sum_{i=1}^{M} f(\vec{x}_i)$$

- The error is statistical:

$$\Delta = \sqrt{\frac{\mathrm{Var}\, f}{M}} \propto M^{-1/2}$$

$$\mathrm{Var}\, f = \langle f^2 \rangle - \langle f \rangle^2$$

- In d>8 dimensions Monte Carlo scales better than Simpson!

Estimating the error

- The expectation value of random variables *X* and *Y* is linear:

$$E[aX + bY] = aE[X] + E[Y]$$

- The expectation value of the mean

$$\bar{X} = \frac{1}{N}\sum_{i=1}^{N} X_i$$

$$E\left[\bar{X}\right] = E\left[\frac{1}{N}\sum_{i=1}^{N} X_i\right] = \frac{1}{N}\sum_{i=1}^{N} E\left[X_i\right] = \frac{1}{N}\sum_{i=1}^{N} E\left[X\right] = E[X]$$

- => the mean is a “true estimator”

Estimating the error

- The sampling error is the rms (root mean square) deviation

$$(\Delta X)^2 = E\left[\left(\bar{X} - E[X]\right)^2\right] = E\left[\left(\frac{1}{N}\sum_{i=1}^{N}(X_i - E[X])\right)^2\right]$$

$$= E\left[\frac{1}{N^2}\sum_{i,j=1}^{N}(X_i - E[X])(X_j - E[X])\right]$$

$$= \frac{1}{N^2}\sum_{i,j=1}^{N}\left(E[X_i X_j] - E[X]^2\right)$$

$$= \frac{1}{N^2}\sum_{i=1}^{N}\left(E[X_i^2] - E[X]^2\right) = \frac{1}{N}\left(E[X^2] - E[X]^2\right) = \frac{\mathrm{Var}\,X}{N}$$

- We used that samples are uncorrelated:

$$E[X_i X_j] = E[X_i]E[X_j] \text{ for } i \neq j$$

Monte Carlo data analysis

- Recipe for Monte Carlo integration and data analysis of uncorrelated data:
  - Draw random points *x* and evaluate the function, to get random variables X=f(x).
  - Store the number, sums, and sum of squares

$$N, \sum_{i=1}^{N} X_i, \sum_{i=1}^{N} X_i^2$$

  - Calculate the mean as an estimate of the expectation value:

$$E[X] \approx \bar{X} = \frac{1}{N} \sum_{i=1}^{N} X_i$$

  - Estimate the variance from the mean of squares, and from it the error

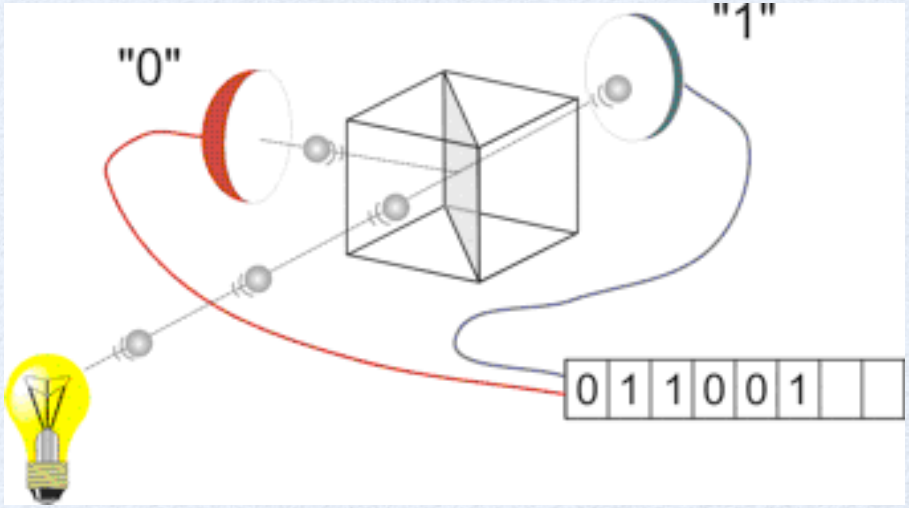
$$\mathrm{Var}\, X = E[X^2] - E[X]^2 \approx \frac{N}{N-1}\left(\overline{X^2} - \bar{X}^2\right) = \frac{N}{N-1}\left(\frac{1}{N}\sum_{i=1}^{N} X_i^2 - \bar{X}^2\right)$$

$$\Rightarrow \Delta X \approx \sqrt{\frac{1}{N-1}\left(\overline{X^2} - \bar{X}^2\right)}$$

Generating random numbers

- Real random numbers are hard to obtain
  - classical or thermal chaos (atmospheric noise)
  - quantum mechanics
- Commercial products: quantum random number generators
  - based on photons and semi-transparent mirror
  - 4 Mbit/s from a USB device, too slow for most MC simulations

http://www.idquantique.com/



Pseudo Random numbers

- Are generated by an algorithm
- Not random at all, but completely deterministic
- Look nearly random however when algorithm is not known and may be good enough for our purposes
- Never trust pseudo random numbers however!

Linear congruential generators

- are of the simple form $x_{n+1}=f(x_n)$
- A reasonably good choice is the GGL generator

$$x_{n+1} = (ax_n + c) \bmod m$$

with a = 16807, c = 0, m = 2^{31}-1

- quality is sensitive to a good choice of a,c,m

- Periodicity is a problem with such 32-bit generators
  - The sequence repeats identically after 2^{31}-1 iterations
  - With 500 million numbers per second that is just 4 seconds!
  - **Should not be used anymore except to seed other generators**

Lagged Fibonacci generators (LFG)

$$x_n = x_{n-p} \otimes x_{n-q} \bmod M$$

- Good choices are (*p*,*q*) with (2281,1252), (9689,5502), (44497,23463) with $M = 2^{32}$ or $M = 2^{31}-1$ and either addition or xor as operations

- Seed blocks usually generated by linear congruential generators
- Have very long periods since large block of seeds
- A very fast generator that can be parallelized!

More advanced generators

- As well-established generators fail new tests, better and better generators get developed
  - Mersenne twister (Matsumoto & Nishimura, 1997) http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html
  - Well generator (Panneton and L'Ecuyer , 2004) http://www.iro.umontreal.ca/~panneton/WELLRNG.html
- However, they cannot be easily parallelized or vectorized

Creating independent sequences

- **Leapfrog**: elements of the sequence are given in a cyclic fashion. For *n* cores, element *i* is given to core *i* mod *n*.
- **Sequential splitting**: the cores use non-overlapping sequential blocks.
- **Independent sequences**: the lagged Fibonacci generators can produce independent streams, depending on the seed.
  - See the SPRNG library and papers by Mascagni for seeding the lagged Fibonacci generator
- **Stochastic seeding**: use a good generator to fill all *n* seed blocks and hope for the best

Independent sequences in the LFG

- With $M=2^m$, the LFG has a maximum cycle length of $(2^q-1)2^{m-1}$
- This is less than the theoretically possible cycle length of $2^{mq}-1$
- There exist $2^{(q-1)(m-1)}$ cycles of maximal length
- We assign each thread to a different cycle by choosing the seed block in a smart way. This seeding is implemented in the SPRNG library (http://sprng.cs.fsu.edu/)

- The Mersenne twister and WELL generator are smarter: they have the maximum possible cycle length. The disadvantage is that no independent sequences can be generated.

Are these numbers really random?

- No!
- Are they random enough?
  - Maybe?
- Statistical tests for distribution and correlations
- Are these tests enough?
  - No! Your calculation could depend in a subtle way on hidden correlations!
- What is the ultimate test?
  - Run your simulation with various random number generators and compare the results

Marsaglia's diehard tests

- **Birthday spacings:** Choose random points on a large interval. The spacings between the points should be asymptotically Poisson distributed. The name is based on the birthday paradox.
- **Overlapping permutations:** Analyze sequences of five consecutive random numbers. The 120 possible orderings should occur with statistically equal probability.
- **Ranks of matrices:** Select some number of bits from some number of random numbers to form a matrix over {0,1}, then determine the rank of the matrix. Count the ranks.
- **Monkey tests:** Treat sequences of some number of bits as "words". Count the overlapping words in a stream. The number of "words" that don't appear should follow a known distribution. The name is based on the infinite monkey theorem.
- **Count the 1s:** Count the 1 bits in each of either successive or chosen bytes. Convert the counts to "letters", and count the occurrences of five-letter "words".
- **Parking lot test:** Randomly place unit circles in a 100 x 100 square. If the circle overlaps an existing one, try again. After 12,000 tries, the number of successfully "parked" circles should follow a certain normal distribution.

Marsaglia's diehard tests (cont.)

- **Minimum distance test:** Randomly place 8,000 points in a 10,000 x 10,000 square, then find the minimum distance between the pairs. The square of this distance should be exponentially distributed with a certain mean.
- **Random spheres test:** Randomly choose 4,000 points in a cube of edge 1,000. Center a sphere on each point, whose radius is the minimum distance to another point. The smallest sphere's volume should be exponentially distributed with a certain mean.
- **The squeeze test:** Multiply 231 by random floats on [0,1) until you reach 1. Repeat this 100,000 times. The number of floats needed to reach 1 should follow a certain distribution.
- **Overlapping sums test:** Generate a long sequence of random floats on [0,1). Add sequences of 100 consecutive floats. The sums should be normally distributed with characteristic mean and sigma.
- **Runs test:** Generate a long sequence of random floats on [0,1). Count ascending and descending runs. The counts should follow a certain distribution.
- **The craps test:** Play 200,000 games of craps, counting the wins and the number of throws per game. Each count should follow a certain distribution.

Non-uniform random numbers

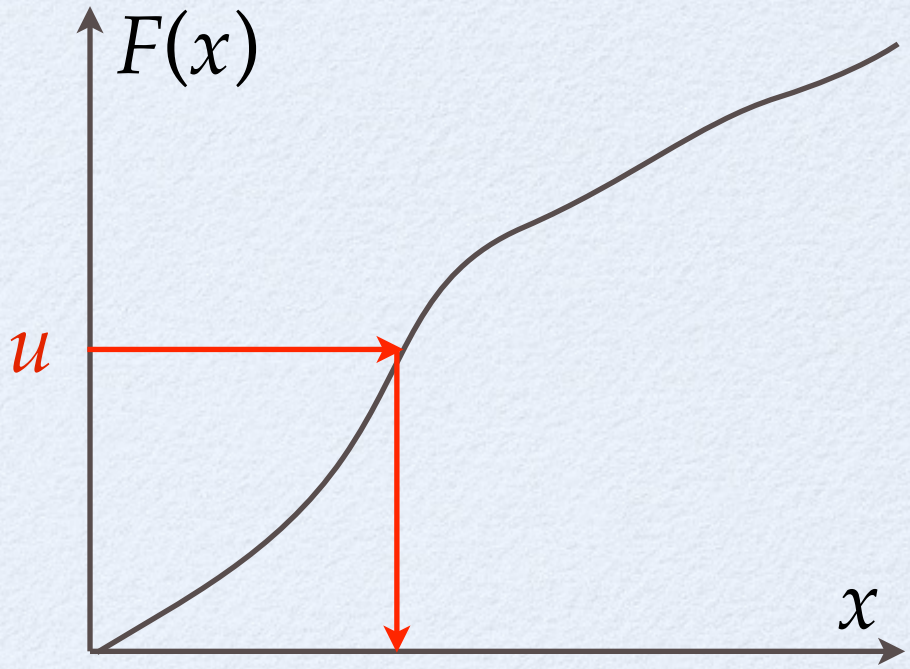
- we found ways to generate pseudo random numbers u in the interval [0,1[
- How do we get other uniform distributions?
  - uniform x in $[a,b[$: $x = a+(b-a)\,u$
- Other distributions:
  - Inversion of integrated distribution
  - Rejection method

Non-uniform distributions

- How can we get a random number *x* distributed with *f(x)* in the interval [*a*,*b*[from a uniform random number *u*?
- Look at probabilities:

$$P[x < y] = \int_a^y f(x)dx =: F(y) \equiv P[u < F(y)]$$

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



- This method is feasible if the integral can be inverted easily
  - exponential distribution $f(x)=\lambda \exp(-\lambda x)$
  - can be obtained from uniform by $x=-1/\lambda \ln(1-u)$

Normally distributed numbers

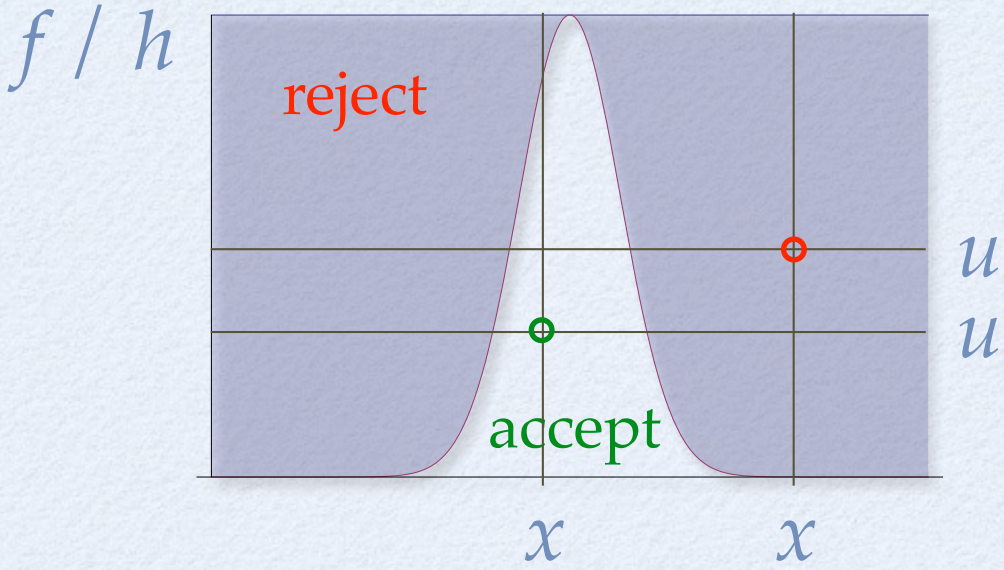
- The normal distribution $f(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-x^2\right)$

- cannot easily be integrated in one dimension but can be easily integrated in 2 dimensions!
- We can obtain two normally distributed numbers from two uniform ones (Box-Muller method)

$$n_1 = \sqrt{-2\ln(1-u_1)}\sin(2\pi u_2)$$

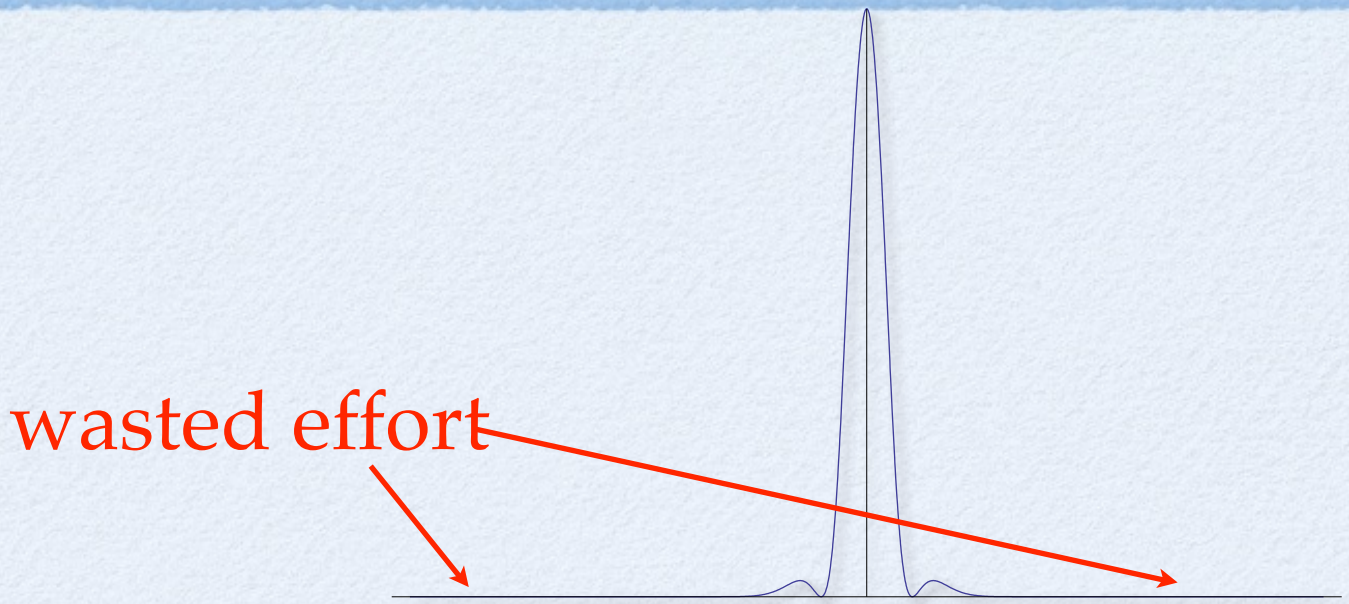
$$n_1 = \sqrt{-2\ln(1-u_1)}\cos(2\pi u_2)$$

Rejection method (von Neumann)



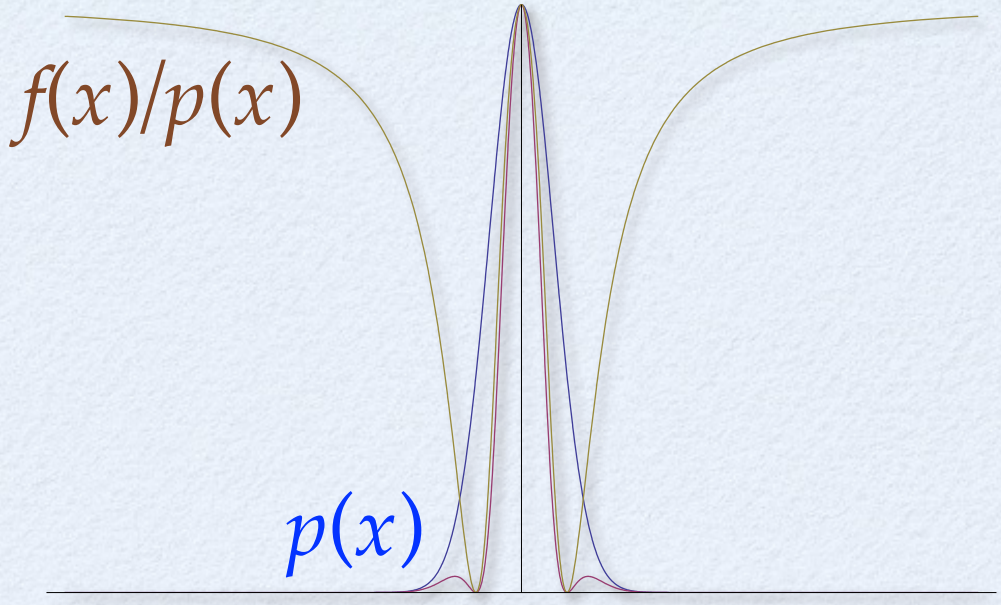
- Look for a simple distribution *h* that bounds *f*: $f(x) < \lambda h(x)$
  - Choose an *h*-distributed number *x*
  - Choose a uniform random number $0 <= u < 1$
  - Accept *x* if $u < f(x) / \lambda h(x)$,
    otherwise reject *x* and get a new pair *(x,u)*
- Needs a good guess *h* to be efficient

Sharply peaked functions



- In many cases a function is large only in a tiny region
- Lots of time wasted in regions where the function is small
- The sampling error is large since the variance is large

Importance sampling



- Choose points not uniformly but with probability $p(x)$:

$$\langle f \rangle = \left\langle \frac{f}{p} \right\rangle_p := \int_\Omega \frac{f(\vec{x})}{p(\vec{x})} p(\vec{x}) d\vec{x} \Big/ \int_\Omega d\vec{x}$$

- The error is now determined by Var f/p
- Find p similar to f and such that p-distributed random numbers are easily available