Random Number Generators

Algorithms typically produce numbers within (0,1] or similar
Be careful with exactly zero or one

- ‘Pseudo-Random’ Numbers: Deterministic Algorithm
- Many Different Algorithms: linear congruential, Mersenne Twister, ...
- Built-in generators in Fortran, Python, ...
- Many others available: Gnu Scientific Library, Random123, ...
- Typically well-tested, but errors do creep in
- Be careful with initialization

Using Random Number Generators: Serial Codes

1. Give option to initialize random number generator
2. Store state of random number generator at end of run
3. For next run initialize with stored state
4. Print out state to be able to reproduce runs

Examples in Fortran and Python

- Fortran
  - call random_number(u) produces a random number u from 0 to 1
  - *u* can be an array
  - call random_seed(size,put,get) sets or gets information about seed
  - *size* is an integer describing number of integers to describe state
  - *put* is an integer array used to set the state
  - *get* is an integer array used to get the state
  - example ' call random_seed ' gives random initialization
  - example ' call random_seed(size=k) '
  - example ' call random_seed(put=seed(1:k) '

Examples in Fortran and Python

- Python
  - 'import random' sets up random number routines
  - 'randomm.random(u)' gets random numbers from [0,1) (always < 1)
  - u can be an array (list)
  - 'random.seed(i)' initializes the generator with seed i
  - 'random.getstate()' gets the state of the random number generator

Example code for Python

```
import random, os, pickle

if os.path.exists('state.dat'):
    print 'Found state.dat, initializing random'
    with open('state.dat','rb') as f:
        state=pickle.load(f)
    random.setstate(state)

else:
    # Use a well-known start state
    print 'No state.dat, seeding with 1 '
    random.seed(1)

....

with open('state.dat','wb') as f:
    pickle.dump(random.getstate(),f)
```

Measurements: Integrals with Monte Carlo

Monte Carlo allows us to approximate an integral:

$$\begin{aligned} I &= \int dx\ F(x) \\ &= \int dx\ W(x)\ \frac{F(x)}{W(x)} \end{aligned}$$

We sample points x_i from $W(x)$ and evaluate $F(x_i)/W(x_i)$ for each point.
In the limit of a large number of samples, the average

$$\bar{X} = \frac{1}{N} \sum_{i=1,N} F(x_i)/W(x_i) \rightarrow I$$

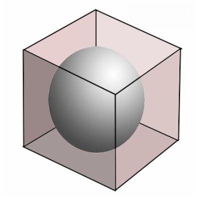
converges to the integral I. The error typically goes like $1/\sqrt{N}$, and depends critically upon the choice of $W(x)$.

Sampling Distributions

Simple Algorithms Exist for Smapling Other Distributions:

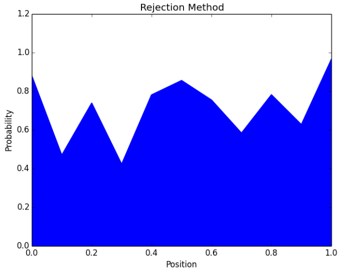
- Different range (eg. 0 to 2π): $r \;=\; 2\,\pi\,\xi$
- Exponential Decay: $r \;=\; t_0\,(\,-\,ln\,\xi)$
- Linear between 0 and 1: $r = \max\,(\xi_1, \xi_2\,)$
- Quadratic between 0 and 1: $r = \max\,(\xi_1, \xi_2, \xi_3\,)$
- Gaussian (Box-Muller Algorithm):
  1. Generates Pairs of Gaussian distributed numbers
  2. $\Theta \;=\; 2\,\pi\,\xi_1$
  3. $R \;=\; \sqrt{-2\;\ln\,(\xi_2)}$
  4. $r_1 \;=\; R\;\cos(\Theta)$
  5. $r_2 \;=\; R\;\sin(\Theta)$

Sampling Other Distributions



- Sampling within a sphere (Rejection) :
  1. $r_1 = 2 * (\xi_1 - 0.5)$
  2. $r_2 = 2 * (\xi_2 - 0.5)$
  3. $r_3 = 2 * (\xi_3 - 0.5)$
  4. If $r_1^2 + r_2^2 + r_3^2 \ > \ 1$; go to (1) and repeat
- Sampling on a sphere:
  1. Sample within a sphere
  2. Scale x, y, and z components to be on a sphere
- Sample 3 (right-handed) orthogonal axes randomly
  1. Sample on a sphere (set as first axis): v_1
  2. Sample a second point within a sphere: v_a
  3. Set second axis: $v_2 \ = \ v_a \times v_1 / |v_a \times v_1|$
  4. Third axis $v_3 \ = \ v_1 \ \times \ v_2$

Sampling By Rejection



If function you want to sample has a maximum value P_{max}:

1. Sample x randomly in the volume
2. Compute the ratio of the probability $R = P(x)/P_{max}$
3. Select a random number, Reject if $\xi > R$, return to step 1
4. If $\xi < R$ you are done, keep x as the sample.

This can be extremely inefficient in large spaces with a wide dynamic range, but it can works well for low dimensions.

Sampling a general 1-D positive distribution

A general normalizable positive function D can be sampled either analytically (if closed integrals exist) or numerically.

Here we assume a positive probability D from zero to infinity:

- Let $F(X) = \int_0^X \ dY \ D(Y)$
- Note: $F(0) \ = 0$ and $F(\infty) \ = \ 1$
- Set $r = F^{-1}(\xi)$

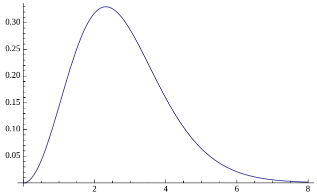
If integrals or inverse is not simple, this can be done numerically.
Example: $D(x) = (1/\gamma) \ \exp(-x/\gamma), x \ > \ 0$
$F(X) \ = \ 1 - \exp(-X)$
$F^{-1}(\xi) \ = \ -\gamma \ \ln(1 - \xi) = -\gamma \ \ln(\xi')$

Example: sampling a general distribution

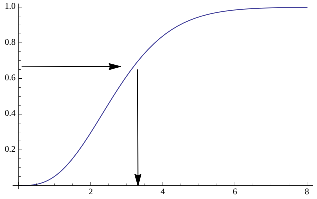
Suppose we want to sample a momentum p from:

$$P(p)\mathcal{N}\ p^2\ \exp[-\sqrt{(\hbar c p)^2 + m^2}\ \Delta\tau]$$

with $m = 935$ MeV, $\Delta\tau = 0.01$ MeV-1



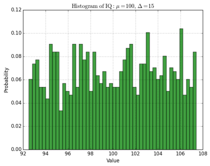
P(p)

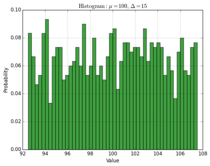


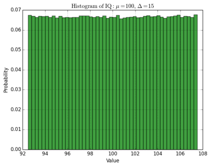
Integrated P(p)

Central Limit Theorm

Sampling a distribution from: $x_i = 100 + 15 * (\xi - 0.5)$
ξ is uniformly distributed from 0 - 1.







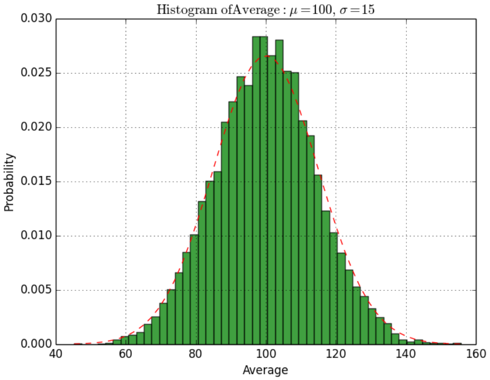
100 samples 1000 samples 1,000,000 samples

(see python code histogram0.py)

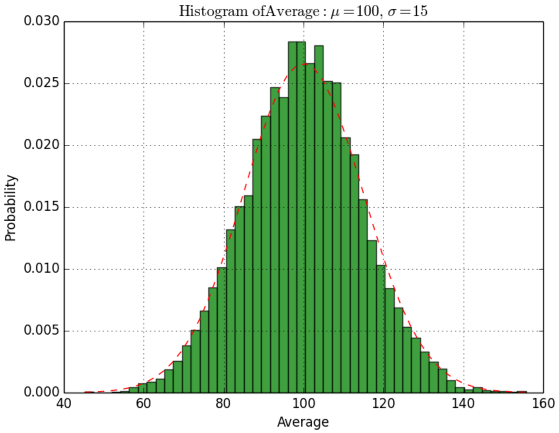
Central Limit Theorem

- Assume independent samples
- Compute averages by 'blocks':
- $\bar{X}_i = \sum_{j,\ldots,j+nb} X_i / nb$
- Distribution of averages over blocks converges to a Gaussian (for large enough nb)

Histogram of 100,000 blocks, each the average over 100 samples:



Central Limit Theorem



Statistical Errors from Gaussian Distributions

- Assume independent samples with gaussian distribution of 'measurements'
- Compute average $\bar{X} \;=\; \frac{1}{nb} \sum X_i$
- Compute standard error $\sigma \;=\; \left((\sum (X_i - \bar{X})^2 / nb) / nb \right)^{1/2}$

Probabilities exact average x of being more than 1,2,3 σ from exact integral

- $\Pr\,(\bar{X} - \;\; \sigma \leq x \leq \bar{X} + \;\; \sigma\;) \;\approx 0.6827$
- $\Pr\,(\bar{X} - 2\;\sigma \leq x \leq \bar{X} + 2\;\sigma\;) \;\approx 0.9545$
- $\Pr\,(\bar{X} - 3\;\sigma \leq x \leq \bar{X} + 3\;\sigma\;) \;\approx 0.9973$

Many other methods are available to try to deal with non-gaussian statistics.

Bayesian methods are valuable, particularly in difficult cases.

Classical Monte Carlo

Suppose we have N particles in a finite volume, with a Hamiltonian:

$$H = \sum_i \mathbf{p}_i^2/(2m) + \sum_i V^1(\mathbf{x}_i) + \sum_{i<j} V^2(\mathbf{x}_i, \mathbf{x}_j)$$

We want to determine the system's properties at a finite termperature T.
The partition function is defined as the sum over all states:

$$\mathcal{Z} = \sum_i \exp\left[-E_i/(k_B T)\right]$$

From the partition function we can calculate all observables of the system in statistical equilibrium.
The states of a classical system are defined by the positions $\mathbf{x}_i$ and the momenta $\mathbf{p}_i$ of the particles. Each state has a definite energy E_i.

Classical Monte Carlo (cont'd)

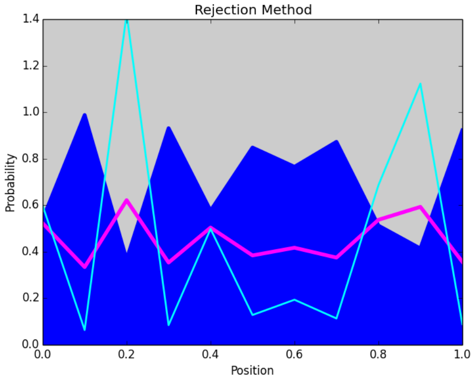
We can sample states from the partition function using MC methods. The expectation value of an observable is

$$\langle O \rangle = \sum_i \exp\left[-E_i/(k_B T)\right] O_i$$

This can be calculated by Monte Carlo up to fairly large systems.

- Sample the states $(\mathbf{x_i}, \mathbf{p_i})$ with energy E_i from $\mathcal{Z}$.
- Note that for a classical system positions and momenta can be sampled indepedently: for each $\mathbf{x_i}$ the momenta can be sampled from a gaussian.
- Compute observables by averaging over these states.

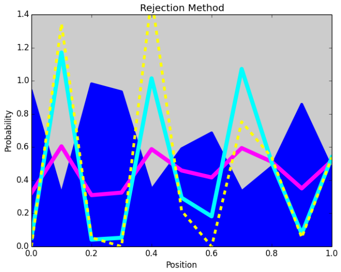
Example of Classical Monte Carlo

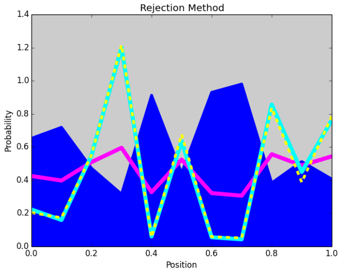


- Create random potential (blue) as before
- Exact distribution at $\beta = 1/(k_B T) = 1$ is magenta line
- Exact distribution at $\beta = 1/(k_B T) = 5$ is cyan line

Example of Classical Monte Carlo

Sample by rejection:



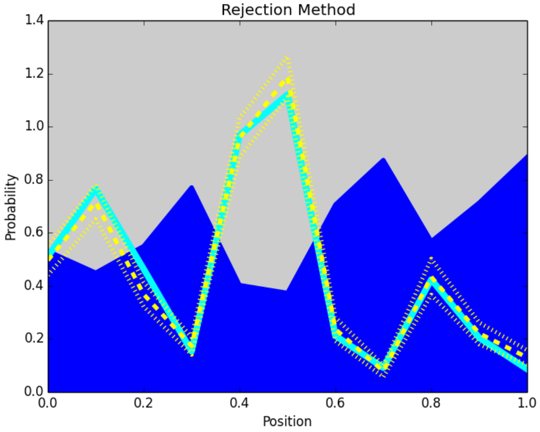


100 samples per bin 10000 samples

Sample by rejection for $\beta = 5$ using max $exp[-\beta E_i] < 1$

Example of Classical Monte Carlo

Sample by rejection (with errors):



Calculate averages and errors with 20 blocks, 50 samples per block
see code rejection.py

Example Code in Python

```
beta = 5 ; nsamp=50 ; nblk = 20
zpart = np.exp( -beta*y)
nexp = np.zeros ((11,nblk))
nr = np.zeros (11) ; ne = np.zeros (11)
# loop over blocks
for j in range (0,nblk):
    zr =  np.random.rand(11,nsamp)
#   loop over positions for sampling
    for i in range (0,11):
        nr[i] = np.sum( zr[i,:] < zpart[i])
#   noremalize and store results for this block
    nrsum=np.sum(nr)
    nr[:] = nr[:]*0.5/(nrsum*0.1)
    nexp[:,j] = nr[:]
#   compute average and error for each position
for i in range(0,11):
    nr[i] = np.sum( nexp[i,:])/nblk
    ne[i] = np.sum( (nexp[i,:]-nr[i])**2 )/nblk
    ne[i] = np.sqrt( ne[i] / nblk )
```

Metropolis Monte Carlo and Markov Chain Algorithms

Metropolis Monte Carlo is an algorithm designed to sample complicated many-variable distributions. It employs a Markov Chain Monte Carlo algorithm to achieve this. see: *Metropolis, N.,*

Rosenbluth, A., Rosenbluth, M., Teller, A., and Teller, E., "Equations of state calculations by fast computing machines", J. Chem. Phys. 2121(6) 1087 (1953).

- A Markov Chain Monte Carlo Algorithm employs random walks where each step depends only upon the present position of the system (no 'history').
- The Metropolis algorithm assumes that you want to sample from a non-negative function $W(\mathbf{R})$ which may be in many dimensions: eg. $\mathbf{R} = (\mathbf{r}_i)$ for many particles i.

Metropolis Monte Carlo

- Metropolis algorithm converges to sampling $W(\mathbf{R})$ asymptotically, but with *a priori* unkown equilibration time and unknown time between 'independent' samples.
- Detailed balance is enforced, requiring the flux from point $\mathbf{A} \rightarrow \mathbf{B}$ equals that from $\mathbf{B} \rightarrow \mathbf{A}$
- Rejection is used, we propose a move with a probability $T(\mathbf{A} \rightarrow \mathbf{B}|$ and accept that move with probability $P(\mathbf{A} \rightarrow \mathbf{B}|$.
- Detailed balance requires:
  $W(\mathbf{A})\ T(\mathbf{A} \rightarrow \mathbf{B})\ P(\mathbf{A} \rightarrow \mathbf{B}) \ = \ W(\mathbf{B})\ T(\mathbf{B} \rightarrow \mathbf{A})\ P(\mathbf{B} \rightarrow \mathbf{A})$

Note that detailed balance is more restrictive than strictly necessary.

Metropolis Monte Carlo

A simple Metropolis Algorithm:

1. Initialize all particles within the physical volume and calculate W(**R**)
2. Propose a move of all particles $\mathbf{R}'$, within a box centered on the current positions: $r'_{i,j} = r_{i,j} + b(\xi_{i,j} - 0.5)$
   where i runs over 3 dimensions and j over N particles
3. Calculate $W(\mathbf{R}')$
4. Accept this proposed move with probability P given by:
   $P \;=\; \min[W(\mathbf{R}')/W(\mathbf{R}), 1]$
5. If the move is 'accepted', set the current position to $\mathbf{R}'$, otherwise keep the current position at $\mathbf{R}$
6. Return to step 2, and propose a new move

After some number of steps the samples will be independent. Averages and Errors can be calculated using these independent samples

Problem for afternoon

Consider a system of 10 particles in one dimension ($0 < x_i < 1$) with a one-body (background) potential : $V_1(x) = 1 + Cos(20\pi x)$.

- Calculate analytically the probability density $\rho(x)$ using the partition function at $\beta = 0.2$ and $\beta = 1$.
- Use rejection to determine this probability density with averages and errors
- Use Metropolis Monte Carlo to sample this same density
- Add a two-body repulsive potential

$$\begin{aligned} V_{ij} &= V_0 \text{ if } |x_i - x_j| < 0.05 \\ &= 0 \text{ if } |x_i - x_j| \geq 0.05 \end{aligned}$$

Rejection is harder for the last case, what behavior do you expect? Try $V_0 = 1$ to start with.