Lecture 7:

Molecular Dynamics Simulations

Simulations of many-body systems

- **General problem:** description of motion of many interacting objects (particles). The type of particles and their interaction can vary
  e.g. gases/liquids (interacting molecules), plasma (electrons and ions), planetary systems and galaxies (planets, stars), etc.

- The laws governing the motion of individual particles are known (classical, quantum, relativistic), and the nature of their interaction is also typically known

- How many is **"many"**? *E.g. interacting classical particles:*

$$m\frac{d^2\vec{r}_i}{dt^2} = -\frac{\partial}{\partial\vec{r}_i}\sum_{\substack{j=1\\j\neq i}}^{M} V(|\vec{r}_i - \vec{r}_j|)$$

Two particles: can solve analytically

Three particles: … need a computer!

Few-body vs many-body

- Can sub-divide many-body problems into two categories

1) Start with exact (known) initial condition (*e.g. positions and velocities for classical particles*).
   Aim: obtain exact trajectories
   e.g. model the Solar system

- This category of problems often requires accurate integration

  e.g. for planning various space missions we need to be able to model the Solar system with a good precision

- Can handle computationally reasonably small numbers of particles ($N \sim 100$), various special techniques exist for such problems.

  Such problems are often called "few-body" problems

Simulations of many-body systems

2) "Statistical mechanics".
    Aim: extract information about global/macroscopic properties of the system from local/microscopic rules of interaction
    e.g. obtain the state equation for a non-ideal gas

- Typical systems have HUGE numbers of particles *(remember that one mole of a liquid/gas has ~10^{23} molecules)* => Can only model approximately, by studying a small sub-system and carefully applying adequate boundary conditions in attempt to mimic a much larger system

- Initial condition is not known, but it does not matter too much either, as we are interested in statistical properties

- Likewise, a great accuracy in integration is not really required...

In what follows we will mainly focus on such many-body problems

Many-body problems for statistical mechanics: phase space

- **Many-particle system:** 6N-dimensional phase space
  (that's x, y, z and $\dot{x}, \dot{y}, \dot{z}$ for each particle)

- A particular combination of coordinates and velocities of each particle defines a <u>state of the system: a point Γ</u> in the 6N-dimensional phase space

- Each particular combination (state) corresponds to a specific set of macroscopic properties of the entire system, such as temperature, pressure, volume, energy…

E.g. average kinetic energy of N classical particles $U \; = < \displaystyle\sum_i \frac{p_i^2}{2m} >$

gives the temperature via $U \; = (3/2) k_B T$

- Evolution of the system corresponds to a trajectory in phase space $\Gamma(t)$

- Specific constrains *(such as fixed energy/volume/pressure)* define which part of the phase space Ω is accessible by the system
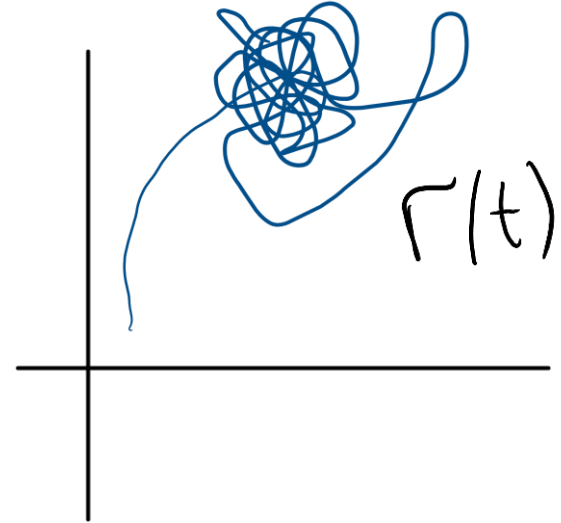
Many-body problems for statistical mechanics: phase space

- As the system evolves, you will observe that some parts of the phase space are visited more frequently than other
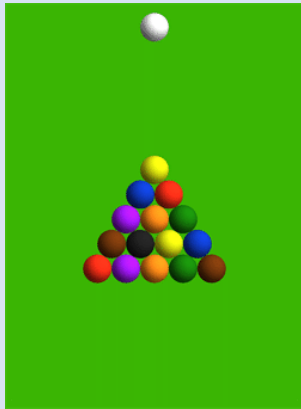
E.g. for billiard balls the initial state is very unlikely to be reproduced again spontaneously...

$\Gamma(t)$

- We can define a probability of the system to be in state Γ: $\boldsymbol{p(\Gamma)}$

- If we measure a macroscopic property at different times (or in different "copies" of the system), the corresponding average (*"ensemble average"*):

Here $A(\Gamma)$ is a macroscopic property (e.g. temperature), which depends on the particular microscopic state

Many-body problems for statistical mechanics: phase space

$$< A > = \int_\Omega A(\Gamma)p(\Gamma)d\Omega$$

- The main task of many-body simulations in statistical mechanics would be in evaluating such integrals.

- This is just a volume integral… could we simply calculate it by using e.g. the trapezium rule?

  - Probability function $p(\Gamma)$ is often unknown, or difficult to normalize *(i.e. known up to a normalization pre-factor)*

  - The integral is in multi-dimensional space! *E.g. for N classical particles the dimensionality of Ω is 6N – that's a lot!*

- Generally, there are two approaches in computational stat mechanics:

1) Replace ensemble average with time average and use "deterministic" methods to generate a trajectory $\Gamma(t)$ **– Molecular Dynamics simulations**

2) "Probe" the phase space by generating random points **– Monte Carlo methods**

Molecular Dynamics

- Developed in 1950s, initially as a statistical mechanics computational tool.

    In 1957 Berni Adler performed simulations of a dense fluid of hard spheres, and formulated the main ingredients of the MD simulations.

- MD method became a powerful technique in many branches of science including chemistry, biophysics, material science, astrophysics

- **Main idea:** model time evolution of many particles (e.g. molecules) using known equations of motion (Newton's laws). **The method is generic**, does not rely on a specific nature of interactions or type of particles

- Allows to **analyse explicit dynamics** of individual molecules

- There are few technical difficulties:

    - Finite-size effects (boundary conditions)

    - Calculation of the interaction forces

    - Numerical implementation of the time step

Berni Adler
1925-2020
https://en.wikipedia.org/wiki/Berni_Alder

Boundary conditions

- A small volume of gas/water/solid might contain ~10^{23} particles
  => This is far too many for a simulation!
  (a typical MD simulation will have N between 10^2 and 10^6)

- Often can only simulate a small part of the system. **Finite size effects?**

- Take e.g. a $10 \times 10 \times 10$ cluster of particles in a box $(N = 1000)$. How many of those will be affected by the boundaries?

A simple estimation:

- In 1D 2 out of 10 would be at the boundaries

- In 2D 100-64=36 would be at the boundaries

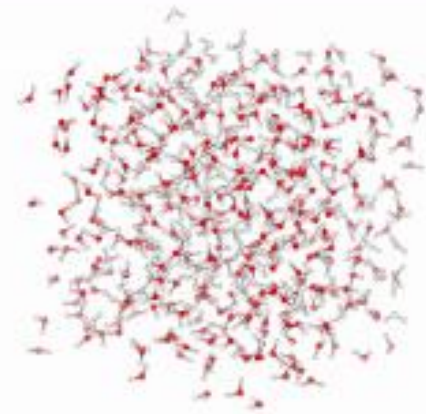- In 3D 1000-8*8*8=488 would be at the boundaries

I.e. nearly a half of all molecules would be affected by the surface!

- To mimic a larger system, **periodic boundary conditions** is the best option

Periodic Boundary Conditions

- You are performing a simulation in a small volume

- Assume the volume is surrounded by identical copies of the simulated volume

MD simulation of water (100ps evolution)
https://en.wikipedia.org/wiki/Molecular_dynamics

- A particle that leaves the volume on one side is replaced by an image particle that enters from the other side

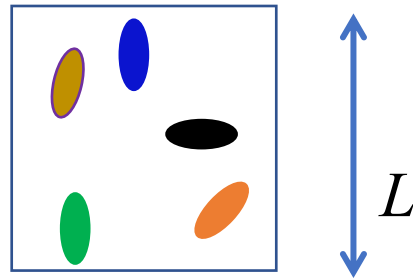- There are no walls, and no surface particles – a model of an infinite system

Note: some systems inherently contain a boundary *(e.g. a droplet of water).* Periodic Boundary Conditions is not a good option in such situations.
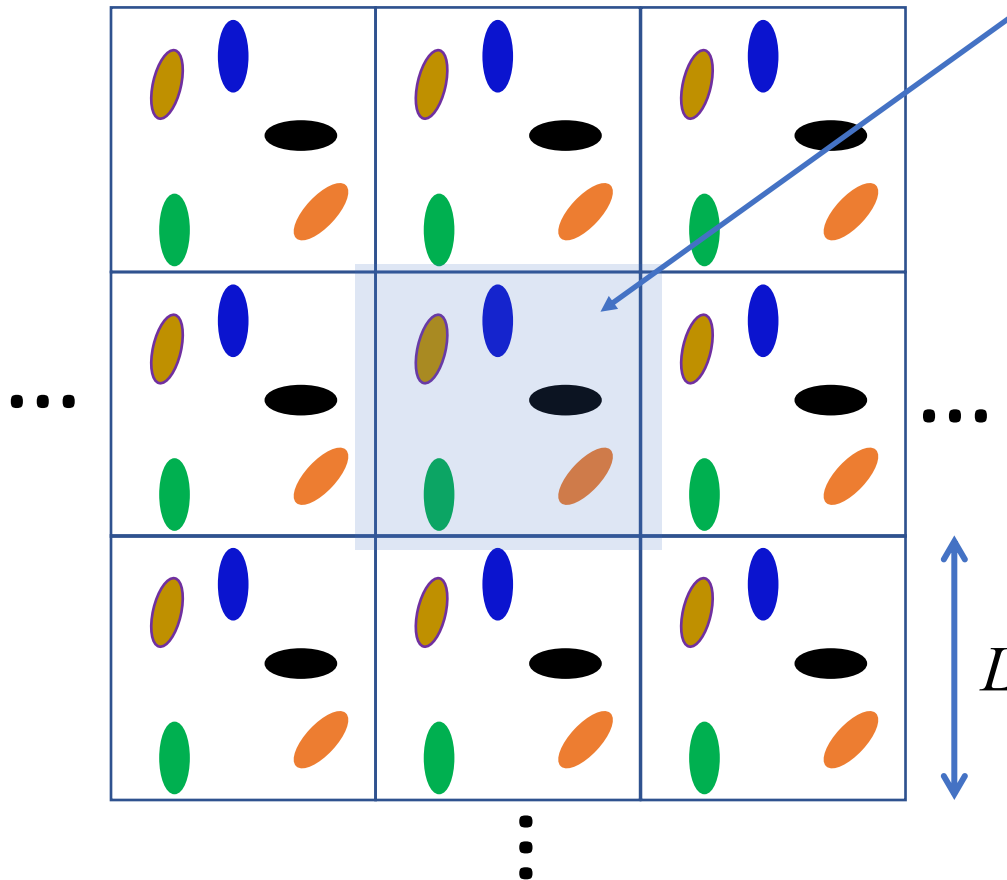
Periodic Boundary Conditions (PBC)

Single box:

$N = 5$

L

We are actually modelling one box

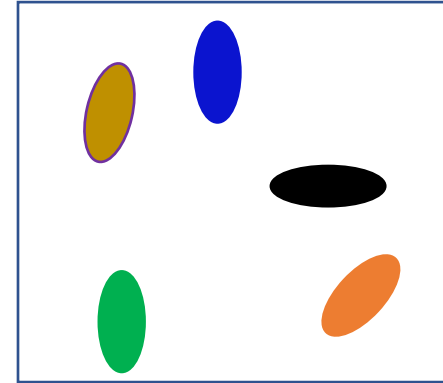Same system, periodic boundary conditions:

L

Particles move around, and may change box, but always in each box $N = 5$

PBC is still an approximation, but it is better than just a single box. *(If you expect to observe e.g. a clustering of particles, you need to have an adequate size of the box)*

Interaction

$$m_i \frac{d^2 \vec{r}_i}{dt^2} = - \frac{\partial}{\partial \vec{r}_i} \sum_{\substack{j=1 \\ j \neq i}}^{N} V(|\vec{r}_i - \vec{r}_j|)$$

- **Potential function** $V\left(\left|\vec{r}_i - \vec{r}_j\right|\right)$ defines the pair-wise interaction

- At each step we need to calculate forces acting on each particle. **This is the most time consuming part of the simulation**

- Some useful assumptions:

  - Split in pairs. In each pair action=-reaction (3rd Newton's law)

  - The force of interaction depends only on the distance between the particles

  - (for identical molecules) interaction potential is the same for each pair

Interaction: Truncating the potential

- With periodic boundary conditions, each atom interacts with every other atom + infinite number of its periodic images => impossible to compute!

- Interaction forces decay with distance, hence we can "cut it off" when the distance is too large

$V(x)$

x

$r_1\, r_c$
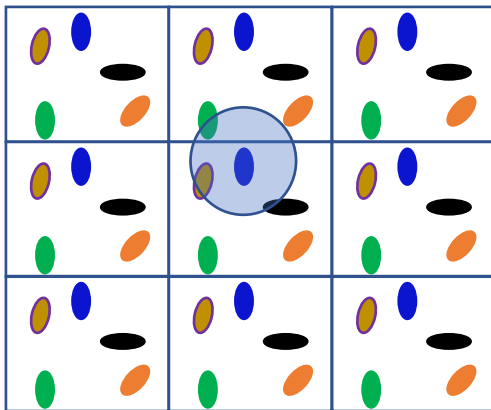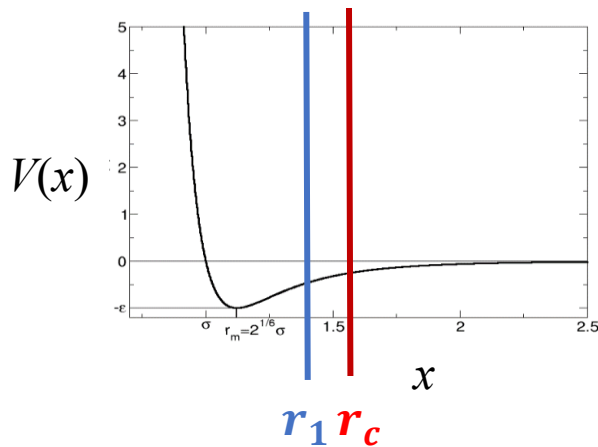
- We can introduce a cut-off distance $r = r_c$:

$$V^{(cut)}(r) = V(r),\, \mathrm{r} \le r_c$$
$$V^{(cut)}(r) = 0,\, \mathrm{r} > r_c$$

- **BUT** a discontinuity in the potential would create an infinite force at the boundary ($\vec{F} = -dV/d\vec{r}$). => It is better to "switch off" the potential smoothly:

$$V^{(cut)}(r) = V(r),\, \mathrm{r} \le r_1$$
$$V^{(cut)}(r) =$$
$$V^{(cut)}(r) =$$

where $S(r)$ is a "tapering" function of your choice, $S(r_1) = 1, S(r_c) = 0$. *(e.g. could simply use linear tapering)*

Interaction: Minimum Image Convention

- It is common to set the cut-off radius to be not larger than half of the size of the unit cell: $r_c \leq L/2$

- With such cut-off **each molecule interacts with only the nearest image of the other $N - 1$ molecules**

- This approach is known in literature as the "Minimum Image Convention"
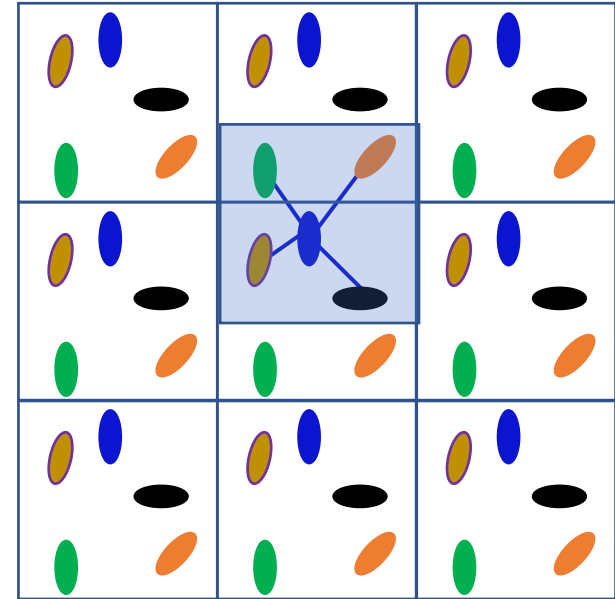
  - Requires some "book keeping"

- By setting the box size L large enough, $L \geq 2r_c$, and hence using the Minimum Image Convention, we are minimizing the impact of the artificial periodicity of the system

Initial conditions

- …are usually not known – need to setup the system yourself!

- Knowledge of the background physics is helpful (as usual), in particular:

  - Typical distributions of particles in a volume *(e.g. crystal structure for a solid, or distribution of masses and velocities in a "typical" galaxy)*

  - Distributions of particle velocities/energies *(e.g. Boltzmann distribution at a fixed temperature)*

- Often can start with an approximate distribution and let the system "settle" dynamically *(that would require an extra computational time)*

  - E.g. for a liquid it is common to start with a solid crystal structure and let the structure "melt" (by setting appropriate velocities corresponding to the liquid phase temperature!)

- Sometimes can use results of another simulation to assist you with a proper initial condition

  - E.g. to setup a collision of two galaxies, you could try to generate a stable distribution of masses and velocities for a single galaxy first by performing a separate simulation

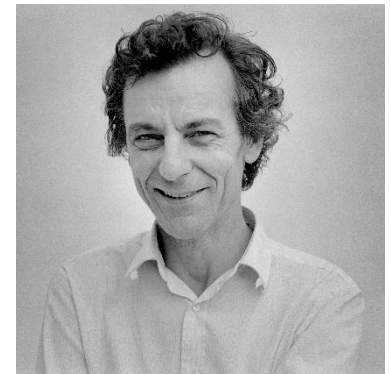# Numerical Integration: Verlet methods

- The most common numerical method in classical MD simulations are different variants of the Verlet method.

- Developed specifically for 2nd law of Newton-type equations where the right-hand side (force) does not depend explicitly on time or velocities.

Loup Verlet
1931-2019
https://www.houches-school-physics.com/news/loup-verlet-510698.kjsp

- Simple to implement

- Compared to other generic IVP methods for ODEs (such as Runge-Kutta methods), offer similar accuracy at lower computational effort

  E.g. the Velocity Verlet method offers $O(\Delta t^4)$ accuracy and requires 4 calculations per step.

  RK4 method gives similar accuracy, but requires 8 calculations per step (4 for positions and 4 for velocities). This is twice slower.

Numerical Integration: Velocity Verlet method

1) Starting with $r_i^{(j)}$ and $v_i^{(j)}$ calculate forces $b_i^{(j)}$ and proceed to calculate updated positions:

$$y_i^{(j+1)} = y_i^{(j)} + v_i^{(j)}\Delta t + \frac{1}{2}b_i^{(j)}\Delta t^2$$

(this is the famous SUVAT formula assuming forces are constant during time interval Δt)

2) Calculate velocities at *half-step*:

$$v_i^{(j+1/2)} = v_i^{(j)} + \frac{1}{2}b_i^{(j)}\Delta t$$

(that's not quite like SUVAT!)

3) Update forces from knowing new positions $y_i^{(j+1)}$

4) Make *another half-step* with the updated accelerations and obtain $v_i^{(j+1)}$

$$v_i^{(j+1)} = v_i^{(j+1/2)} + \frac{1}{2}b_i^{(j+1)}\Delta t$$

5) Repeat steps 1-4 in iterations

How this topic will be assessed:

- Typical questions on exam:

  - *Discuss what type of problems Molecular Dynamics simulations are designed for;*

  - *Explain how periodic boundary conditions work, and why they are necessary in MD simulations;*

  - *For a given size of the computational domain, select the appropriate truncation radius of the interaction potential. Explain what is Minimal Image Convention, and why is it important;*

  - *Explain why Verlet methods are more advantageous than Runge-Kutta methods in molecular dynamics simulations. Why can't (or can?) we use Verlet methods for other types of problems.*

Note: you do not need to memorize the Velocity Verlet algorithm!

Lecture 8:
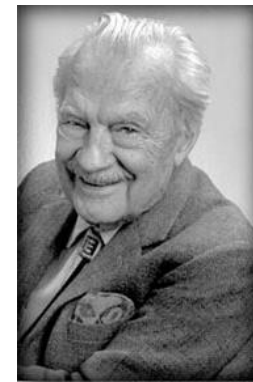
Monte Carlo methods, part 1

Playing a dice game…

Stanislaw Ulam
1909-1984

John von Neumann
1903-1957

Nicholas Metropolis
1915-1999

- *"What are the chances that a Canfield solitaire laid out with 52 cards will come out successfully?"*

- *"After spending a lot of time trying to estimate them by pure combinatorial calculations, I wondered whether a more practical method than "abstract thinking" might not be to* **lay it out say one hundred times and simply observe and count** *the number of successful plays. "*

S. Ulam

Monte Carlo Methods

- Developed in late 1940s in LANL, initially applied to problems related to the development of nuclear weapons (neutron diffusion in the core)

- As this was a classified research, it required a code name. Nicholas Metropolis suggested the name "Monte Carlo" – in reference to the famous casino

- The method turned out to be very useful in many applications... Essentially it can be applied to any problem having a probabilistic interpretation

    - Statistical mechanics
    - Random trajectories (e.g. Langevin equation)
    - Optimisation
    - Data science...

- **Here, we will only focus on the "basics" of Monte Carlo and its applications in statistical mechanics**

Many-body problems in statistical mechanics
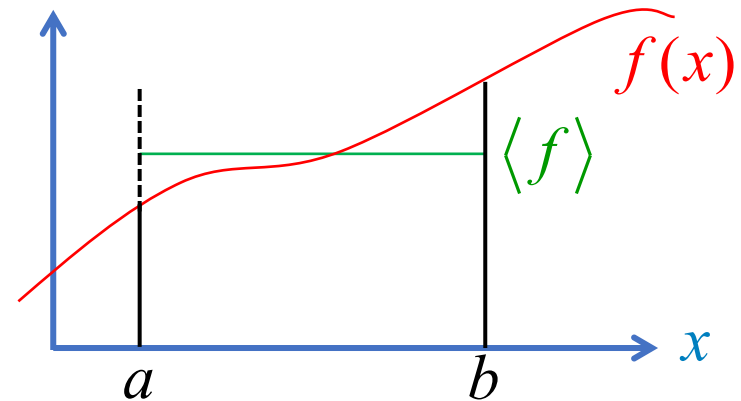
- Need to calculate integrals like

$$< A > = \int_\Omega A(\Gamma) p(\Gamma) d\Omega$$

  in multi-dimensional phase space *(d = 6N for N classical particles)*

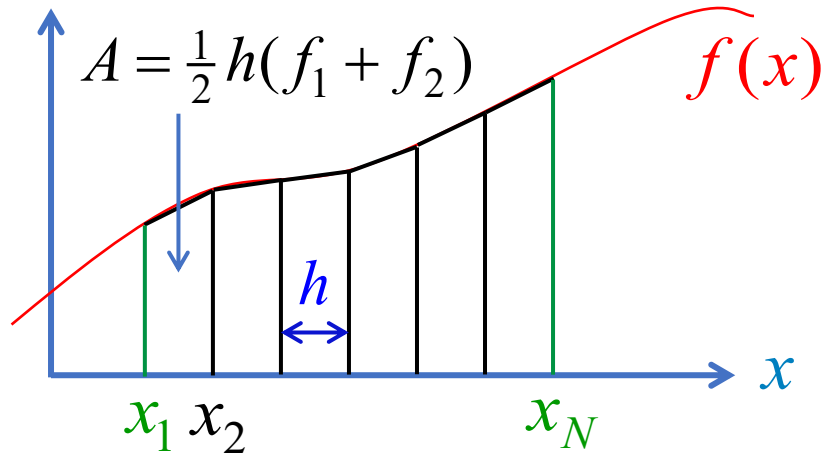- To understand the idea of MC method, we will consider a similar integral in 1D:

This is nothing else but the calculation of the mean value of $f(x)$ in the interval $x \in [a, b]$

Integrate by composite trapezium rule (CTR)

- Need to calculate area under the curve

- Pick a regular grid

$$x_1 = a; \quad x_N = b$$

$$x_{i+1} = x_i + h; \quad f(x_i) = f_i$$

$$A = \tfrac{1}{2} h(f_1 + f_2)$$

$f(x)$

$x_1 \; x_2$ $\quad x_N$ $\quad x$

h

- Area of each section as a trapezium:

- Sum up: $\displaystyle \int_a^b f(x)\,dx \approx h\left\{ \frac{1}{2} f_1 + f_2 + \cdots + f_{N-1} + \frac{1}{2} f_N \right\}$

- In other words: $\displaystyle <f> = \frac{1}{b-a} \int_a^b f(x)\,dx \approx \frac{h}{Nh} \left\{ \frac{1}{2} f_1 + f_2 + \cdots + f_{N-1} + \frac{1}{2} f_N \right\}$

That is simply an average! *(if you ignore ½ coeffs for the very first and the very last points)*

Monte Carlo integration

- Composite Trapezium method gives:

$$< f > = \frac{1}{b-a} \int_a^b f(x)dx \approx F_{CTR} = \frac{1}{N}\left\{\frac{1}{2}f_1 + f_2 + \cdots + f_{N-1} + \frac{1}{2}f_N\right\}$$

- But we could calculate the average in a different way:

$$< f > = \frac{1}{b-a} \int_a^b f(x)dx \approx F_{MC} =$$

where x_j is a set of N random coordinates inside the interval $x \in [a, b]$

- Which of the two methods gives a better estimate?

$$\epsilon_{CTR} = O(h^2) \sim \frac{1}{N^2}$$ - Error of the composite trapezium method

$$\epsilon_{MC} \sim \frac{\sigma}{\sqrt{N}}$$ - error of MC method (standard error of the mean), where

$$\sigma^2 = \frac{1}{N-1}\sum_{j=1}^{N}(f(x_j) - < f_j >)^2$$

Monte Carlo vs Trapezium

composite trapezium: $\epsilon_{CTR} \sim \dfrac{1}{N^2}$ Monte Carlo: $\epsilon_{MC} \sim \dfrac{\sigma}{\sqrt{N}}$

- I.e. if we e.g. increase number of points $N \rightarrow 2N$, the CTR method gives $\epsilon_{CTR} \rightarrow \epsilon_{CTR}/4$, while MC only gives $\epsilon_{MC} \rightarrow \epsilon_{MC}/\sqrt{2}$

> => MC integration is a bad idea in 1D…

- BUT the situation becomes different for d-dimensional volume integrals

$$< A > = \int_{\Omega} A(\Gamma)p(\Gamma)d\Omega$$

- If you take **N points in total**, for CTR method this would mean $N^{1/d}$ points per dimension. =>

- But the error of MC does not depend on the number of dimensions!

-

(in stat mechanics e.g. for 10 particles the phase space is d=60-dimensional)

Monte Carlo integration: Importance sampling

$$< f > = \frac{1}{b-a}\int_a^b f(x)dx \approx \frac{1}{N}\sum_{j=1}^{N} f(x_j) = < f_j >$$
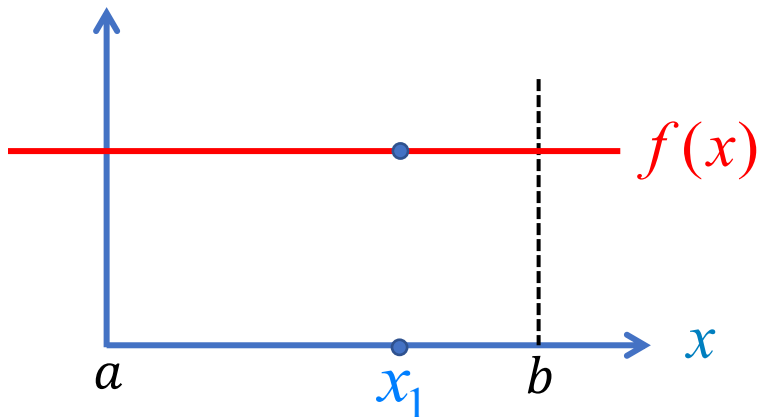
The error of Monte Carlo integration is: $\epsilon_{MC} \sim \dfrac{\sigma}{\sqrt{N}}$

- What can we do to improve σ?

$$\sigma = \left\{ \frac{1}{N-1}\sum_{i=1}^{N}(f(x_i) - \langle f_i \rangle)^2 \right\}^{1/2} \qquad \text{(standard deviation)}$$

Thus, if we can make all the deviations $f(x_i) - < f_i >$ small, then σ and ϵ_{MC} will be small.
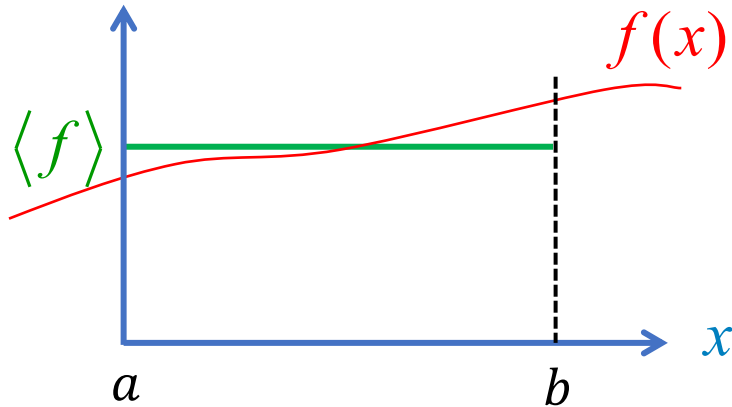
E.g. *(ideal case)*

$$f(x_1) = \langle f \rangle = I$$

Even with just 1 point,

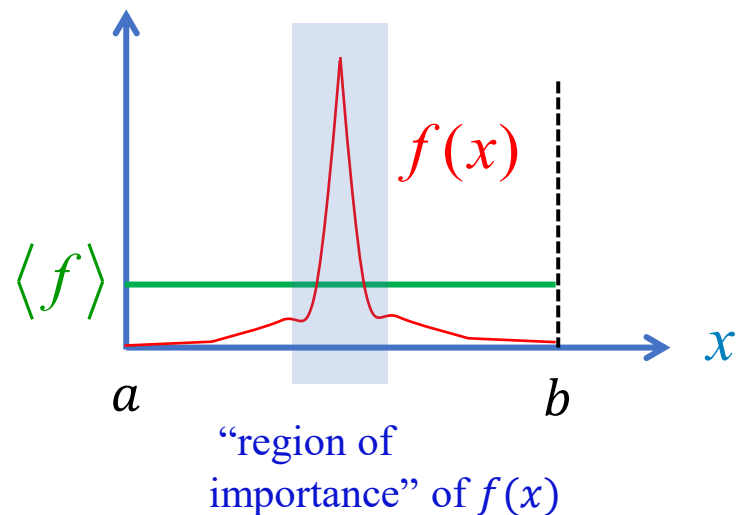$$\sigma = 0 \rightarrow \varepsilon_{MC} = 0.$$

Case 2 *(close to ideal case)*

$f(x)$ slowly varying:

Every $f(x_i) - <f_i>$ is small,
=> σ and ϵ_{MC} are all small

Case 3 *(the difficult one)*

"region of importance" of $f(x)$

$f(x)$ has a strong peak:

If x_i are <u>distributed uniformly</u> in the interval (0,1), nearly every $f(x_i) - <f_i>$ will be large

=> σ and ϵ_{MC} **are all large**
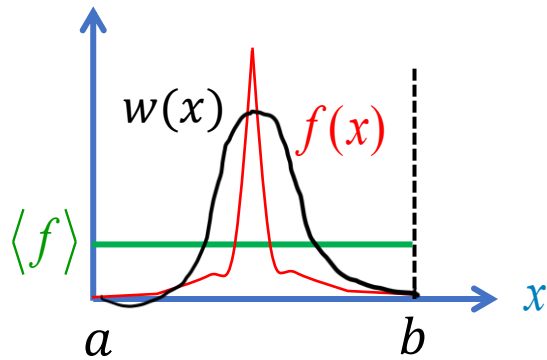
- We could improve the accuracy of MC integration, if we know how $f(x)$ behaves.

- E.g. if we can find a *"weight"* function $w(x)$ which behaves in a similar way….

  then the ratio $f(x)/w(x)$ would be a much "flatter" function – better for MC accuracy

- But we cannot simply replace one integrand with another!
  Instead, use change of variables procedure:

$$< f > = \frac{1}{b-a} \int_a^b f(x)dx \quad = \boxed{}$$

Change variable: $\quad w(x)dx = \dfrac{dy}{b-a} \quad$ $\boxed{}$

 $\quad < f > = \dfrac{1}{(b-a)^2} \int_{y(a)}^{y(b)} \dfrac{f[x(y)]}{w[x(y)]} dy \quad$ - This is *almost* what we wanted

- Original MC integration:

$$< f > = \frac{1}{b-a} \int_a^b f(x)dx \approx \frac{1}{N} \sum_{i=1}^{N} f(x_i)$$

x_i - random coordinates in the interval $[a, b]$

Error: $\epsilon_{MC} = \frac{\sigma_M}{\sqrt{N}}, \quad \sigma_M = \left\{ \frac{1}{N-1} \sum_{i=1}^{N} (f(x_i) - \langle f_i \rangle)^2 \right\}^{1/2}$
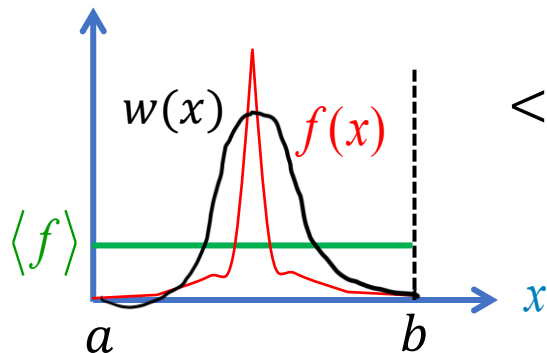
$$\frac{1}{y_2 - y_1} \int_{y_1}^{y_2} h(y) dy$$

- We are replacing it with:

$$< f > = \frac{1}{(b-a)^2} \int_{y(a)}^{y(b)} \frac{f[x(y)]}{w[x(y)]} dy = \frac{y(b)-y(a)}{(b-a)^2} \frac{1}{y(b)-y(a)} \int_{y(a)}^{y(b)} \frac{f[x(y)]}{w[x(y)]} dy$$

$\approx$

y_i - random coordinates in the interval $[y(a), y(b)]$

Error: $\epsilon_I = \frac{\sigma_I}{\sqrt{N}}, \quad \sigma_I = \left\{ \frac{1}{N-1} \sum_{i=1}^{N} \left(\frac{f[x(y_i)]}{w[x(y_i)]} - \left\langle \left(\frac{f}{w} \right)_i \right\rangle \right)^2 \right\}^{1/2}$

Expect $\sigma_I \ll \sigma_M$

$$<f> \approx \frac{y(b) - y(a)}{(b-a)^2} \frac{1}{N} \sum_{i=1}^{N} \frac{f[x(y_i)]}{w[x(y_i)]}$$

y_i - random coordinates in the interval $[y(a), y(b)]$

- **Are there any restrictions for $w(x)$?**

1) $w(x) \neq 0$ everywhere where $f(x) \neq 0$ *(otherwise f/w diverges!)*

2) We must be able to invert $\boldsymbol{y(x) = (b-a) \int_a^x w(x')dx' + C}$

N.b. this also means we need to be able to take integral $\int_a^x w(x')dx'$ *analytically.*
This, and the requirement of inversion, are very serious restrictions!

Note: y(x) maps $[a,b] \rightarrow [y(a), y(b)]$, we must ensure that the inverted function $x(y)$ maps $[y(a), y(b)] \rightarrow [a,b]$

Q: (naively) can we select $w(x) = f(x)$? This would give us $(f(x)/w(x)) \equiv 1$ and zero error!
A: This would mean you are able to take integral $\int w(x)dx = \int f(x)dx$ analytically. But then why would you need Monte Carlo to estimate this integral?

3) (Optional but useful) if we normalize $\int_a^b w(x)dx = 1$ and choose $C = a$ then

$$y = (b-a) \int_a^x w(x')dx' + a \implies y(a) = a, \qquad y(b) = b$$

I.e. we can use random coordinates y_i in the interval $[a, b]$

An Example

Evaluate $I = \int_1^5 x^2 dx$ *[the true answer is $I = \frac{124}{3} \approx 41.43$]*

1) Use Monte-Carlo:

Note: $I = 4 < f >$, $< f > = \frac{1}{5-1} \int_1^5 x^2 dx \approx F_{MC} =$

For illustration, I generated 5 sets of N = 10 random numbers in the interval (1,5):

```
Set 1:  1.6487    3.4079    2.8022    4.3033    1.4266    4.4748    2.7257    1.5443    4.4121    1.3039
Set 2:  4.1771    2.0519    1.3353    3.1534    4.8476    1.3377    4.6426    4.4772    3.4882    1.9597
Set 3:  2.2449    3.6163    1.9159    4.9845    1.0185    2.5991    1.7274    3.3188    2.4038    1.4933
Set 4:  3.1141    3.7569    4.6533    1.3127    4.0996    2.0395    2.0552    3.1994    3.0530    1.7356
Set 5:  1.6626    3.9926    1.6095    2.7707    4.2692    4.2003    1.5822    1.5798    2.6072    1.9598
```

(we will discuss on next lecture how a computer can generate random numbers)

Plug this in the formula to obtain:

$$I \approx 4F_{MC} = \frac{4}{10} \sum_{i=1}^{10} x_i^2 \approx [37.50, 46.51, 30.57, 37.98, 32.19]$$

Estimate error: $\epsilon = \frac{\sigma}{\sqrt{10}} = [9.58, 10.80, 9.02, 8.24, 8.40]$

2) Now, try to improve our result with the introduction of a weight function

$$I = \int_1^5 x^2 dx \qquad I = 4 <f>, \qquad <f> \approx \frac{1}{(b-a)}\frac{1}{N}\sum_{i=1}^{N}\frac{f[x(y_i)]}{w[x(y_i)]}$$
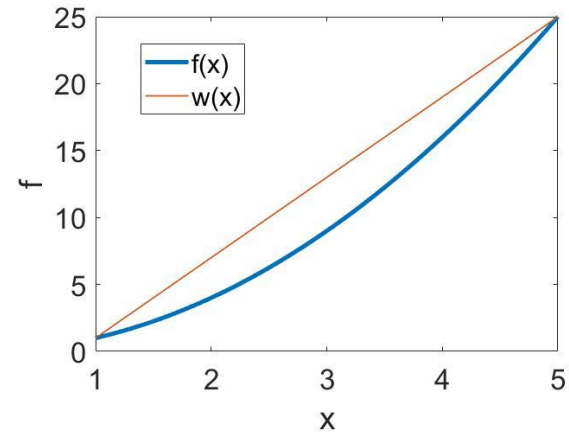
where $y(x) = (b-a)\int_a^x w(x')dx' + a$ and $\int_a^b w(x)dx = 1$

a) Find a suitable weight function $w(x)$, such that $f(x)/w(x)$ has small variations

Observe that the function $f(x)$ is monotonically increasing in this interval

Can try to introduce a linear weight function $w(x)$: e.g. $w(x) = 6x - 5$ to balance this

b) Normalize $w(x)$:

choose $\boxed{w(x) = \frac{1}{52}(6x - 5)}$

2) Now, try to improve our result with the introduction of a weight function

$$I = \int_1^5 x^2 dx \qquad I = 4 < f >, \qquad < f > \approx \frac{1}{(b-a)} \frac{1}{N} \sum_{i=1}^{N} \frac{f[x(y_i)]}{w[x(y_i)]}$$

where $\qquad y = \frac{1}{13}(3x^2 - 5x + 15) \qquad$ and $\qquad w(x) = \frac{1}{52}(6x - 5)$

d) Invert $y(x)$ to obtain $x(y)$:

$$13y = (3x^2 - 5x + 15)$$ ➡️ $$x = \frac{5 \pm \sqrt{156y - 155}}{6}$$ - which sign to choose?

Select the sign which gives $x(1) = 1$ and $x(5) = 5$

$$x(1) = \frac{5 + \sqrt{156 - 155}}{6} = 1 \quad ✔ \qquad\qquad x(5) = \frac{5 + \sqrt{156 \cdot 5 - 155}}{6} = 5 \quad ✔$$

➡️

$$f[x(y)] = x(y)^2 = \frac{\left(5 + \sqrt{156y - 155}\right)^2}{36} \qquad\qquad w[x(y)] = \frac{1}{52}(6x(y) - 5) = \frac{\sqrt{156y - 155}}{52}$$

2) Now, try to improve our result with the introduction of a weight function

$$I = \int_1^5 x^2 dx \qquad I = 4 < f >, \qquad < f > \approx \frac{1}{(b-a)}\frac{1}{N}\sum_{i=1}^{N}\frac{f[x(y_i)]}{w[x(y_i)]}$$

where $\qquad f[x(y)] = \dfrac{\left(5 + \sqrt{156y - 155}\right)^2}{36} \qquad$ and $\qquad w[x(y)] = \dfrac{\sqrt{156y - 155}}{52}$

e) Evaluate

Note: since y is in the same interval $[1,5]$ as x, we can use the same sets of random coordinates as in our first evaluation

$$I \approx 4F_{MC} = 4 \cdot \frac{1}{4} \cdot \frac{1}{10}\sum_{i=1}^{10}\frac{52\left(5 + \sqrt{156y_i - 155}\right)^2}{36 \cdot \sqrt{156y_i - 155}} \approx [39.82, 41.82, 39.13, 40.74, 38.89]$$

[note much smaller variations than in our previous attempt, and all values are closer to the true answer $I \approx 41.43$]

Estimate error: $\quad \epsilon = \dfrac{\sigma}{\sqrt{10}} = [2.57, 2.67, 1.99, 2.14, 2.24]$

[again, the error is ~5 times smaller than in previous attempt!]

How this topic will be assessed:

- Typical questions on exam:

  - *Discuss why and under which conditions the Monte-Carlo integration method can work better than the trapezium method;*

  - *Estimate the value of a 1D integral by:*
    (a) trapezium rule;
    (b) Monte-Carlo (with a provided set of random numbers);

  - *Explain how introduction of a weight function can improve the result of Monte-Carlo integration. Discuss possible limitations on the choice of a weight function;*

  - *For a given 1D integral, suggest a suitable weight function and estimate the value of the integral using the importance sampling method.*

    Note: you do not need to memorize the formula for the Monte-Carlo integration!

See further examples in worksheet 3

Lecture 09:

Random numbers

Random numbers

- The key ingredient of a Monte Carlo simulation is random numbers.
Let us first summarize some important fundamentals of random numbers.

1) Distribution function

x - real random variable with a range of values $[a, b]$

$P(x_0)$ - Distribution function. Gives the probability that x be not greater than x_0

E.g. $P(x_0) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x_0} e^{-x^2} dx$ - Gaussian (normal) distribution, $a = -\infty$, $b = \infty$

$P(x)$ - Monotonically increasing function. $P(a) = 0, P(b) = 1$
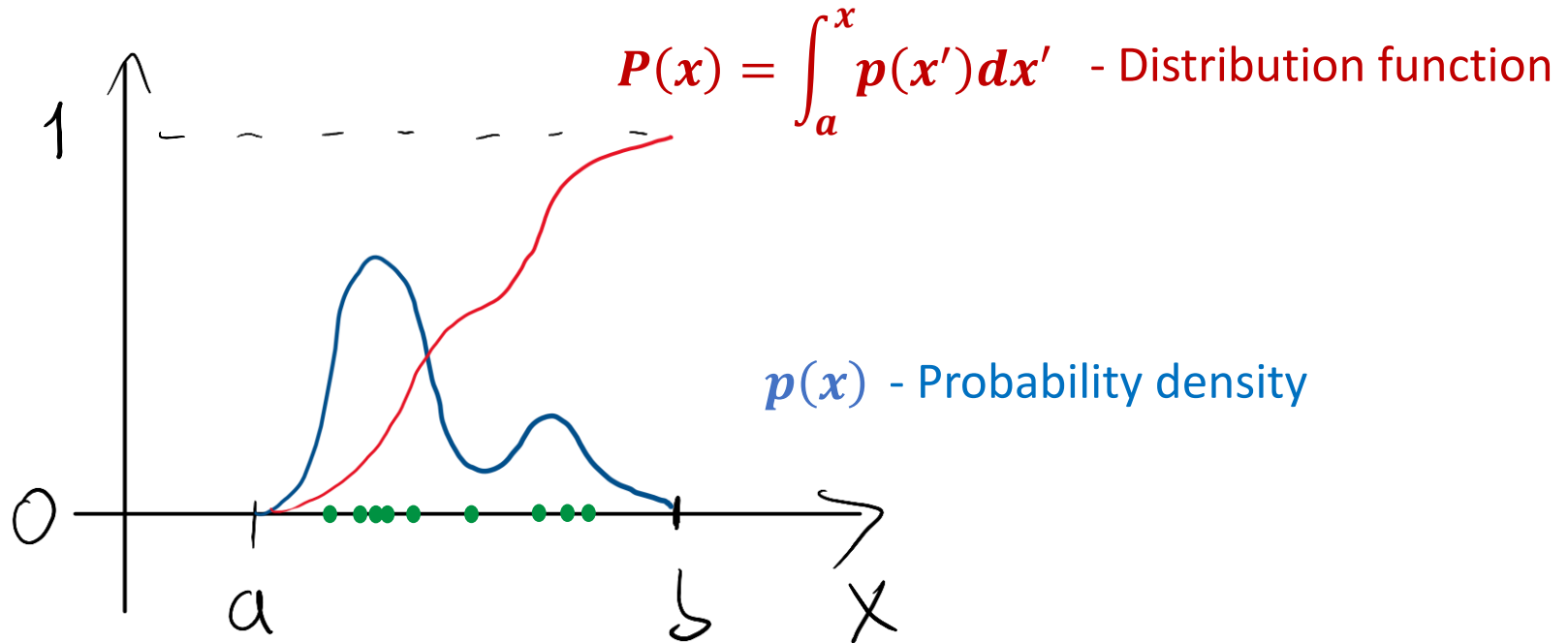
2) Probability density

$p(x_0)dx = dP(x_0)$ - Probability that x is within an infinitesimally small interval around x_0

$p(x) = \frac{dP}{dx},$

In our example, $p(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2}$

Random numbers

$$P(x) = \int_a^x p(x')\,dx'$$ - Distribution function

$p(x)$ - Probability density

- If you have a sequence of many random numbers corresponding to the probability density $p(x)$, they would fill the x −axis more densely in regions with higher $p(x)$

Note: a particular realization of a random variable is called *a variate*. And a sequence of random numbers is called *variates*. In literature, often *variates* and *deviates* are used as synonyms.
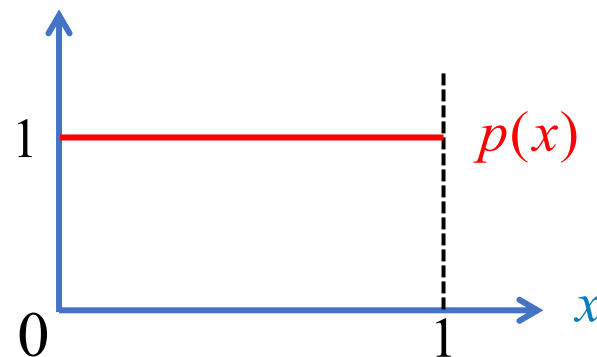
- The great challenge of MC methods is to generate a sequence of Γ points in the phase space from a given probability density $p(\Gamma)$

3) Uniform deviates

Uniform deviates (ud) are simply random numbers within a specified range *(typically between 0 and 1)*, with any value being equally probable

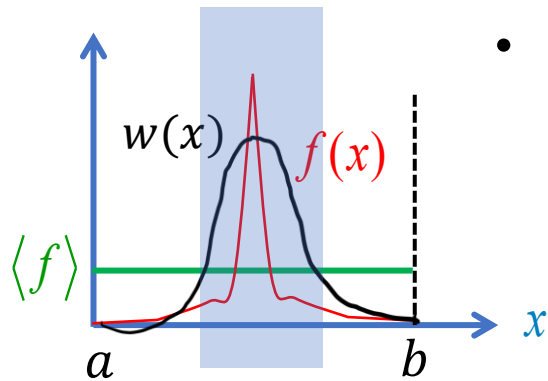$$p(x)dx = dx \qquad 0 \leq x \leq 1$$

$$\int_0^1 p(x)dx = 1$$

More generally, in the interval $[a, b]$:

$$a \leq x \leq b \qquad \int_a^b p(x)dx = 1$$

- We will be using uniform deviates for generation of more complex sequences with non-uniform probability densities $p(x)$

- **On last lecture we derived:**

$$< f > \approx \frac{1}{b-a} \frac{1}{N} \sum_{i=1}^{N} \frac{f[x(y_i)]}{w[x(y_i)]}$$

y_i - uniform deviates in the interval $[a, b]$

(with the normalization $\int_a^b w(x)dx = 1$)

- Use uniform deviates y_i => calculate new random coordinates $x(y_i)$.
  How the new coordinates will be distributed in the interval $[a, b]$?

- **Transformation law of probabilities**

  If y - real random variable with probability density $p(y)$

  $x = f(y)$ - obtain another random variable x

  Probability density of x is given by: $p(x) = p(y) \left| \dfrac{d(f^{-1}(x))}{dx} \right| =$

- In our case $p(y) = \dfrac{1}{b-a}$ - uniform deviates in the interval $[a, b]$

  earlier we defined y(x) as: $w(x)dx = \dfrac{dy}{b-a}$ $\longrightarrow$ $\left| \dfrac{dy}{dx} \right| = |w(x)|(b-a)$

$$p(x) = \frac{|w(x)|(b-a)}{b-a} = |w(x)|$$

- we are sampling the region of "importance" of $f(x)$. This method is called **Importance Sampling**

Monte Carlo and Importance Sampling: a different point of view

- If we want to calculate $I = \int_a^b f(x)dx = (\mathbf{b-a}) < \mathbf{f} >$, we can either:

  - Use the standard MC formula:

  $$I = (b-a) < f > \approx (b-a)\frac{1}{N}\sum_{i=1}^{N} f(x_i)$$

  x_i - uniform deviates in the interval $[a,b]$

  - Or apply the Importance Sampling technique:

  $$I = (b-a) < f > \approx \frac{1}{N}\sum_{i=1}^{N} \frac{f(x_i)}{w(x_i)}$$

  x_i - deviates with the probably density $|w(x)|$ in the interval $[a,b]$

  - But $(\boldsymbol{b-a}) = \frac{1}{u(x)}$, where $u(x)$ is the probability density for uniform deviates

  - And so, we can re-write our first formula as:

  $$I = (b-a) < f > \approx \frac{1}{N}\sum_{i=1}^{N} \frac{f(x_i)}{u(x_i)}$$

  x_i - deviates with the probably density $u(x)$ in the interval $[a,b]$

Monte Carlo for evaluation of the statistical integrals

- Our universal result for 1D integrals:

$$I = \int_a^b f(x)dx \approx \frac{1}{N}\sum_{i=1}^{N}\frac{f(x_i)}{w(x_i)}$$

x_i - deviates with the probably density $|w(x)|$ in the interval $[a, b]$

- In particular:

$$I = \int_a^b f(x)w(x)dx \approx \frac{1}{N}\sum_{i=1}^{N}f(x_i)$$

x_i - deviates with the probably density $|w(x)|$ in the interval $[a, b]$

- Generalizing this for the multi-dimensional case (our original problem):

Γ_i - random points in the multi-dimensional phase space Ω with the probability density function $p(\Gamma)$

- **Need to learn how to generate Γ_i for a given distribution $p(\Gamma)$**

Computer generated random numbers

- A computer cannot generate a truly random number!

  It merely follows your instructions! Anything generated by a computer is deterministic by nature… Which is the opposite of random!

- We can instruct a computer to generate sequences of numbers which will look almost like random: **pseudo-random numbers**

- Any pseudo-random number sequence is generated by an algorithm, and it is completely determined by an initial value called seed. For one and the same value of seed, you will obtain exactly the same sequence of random numbers. *Sometimes a truly random number is used as a seed.*

- Issues with pseudo-random sequences include:

  - Sequences are periodic *(and the period also depends on the seed)*

  - Lack of uniformity of distributions in large sequences

  - Correlation of successive values

Linear congruential generator

- The most common generator of sequences of non-negative integers

$$I_{i+1} = (aI_i + b) \bmod m$$

X mod Y - modulo operation: returns remainder of the division X/Y

Generates integer numbers between 0 and $m - 1$

a - Multiplier (an odd integer)

b - increment (b and m have no common factor)

m   - modulus (typically, the largest integer number)

I_0 - seed

- Sequences are periodic, the period depends on a, b, m and I_0. The largest period is m.

E.g. $a = 2, b = 0, m = 9, I_0 = 1$:   1,2,4,8,7,5,1,2,4,... *(period is 6)*
      $a = 2, b = 0, m = 9, I_0 = 3$:   3,6,3,6,3,6, ... *(period is 2)*
      $a = 4, b = 1, m = 9, I_0 = 0$:   0,1,5,3,4,8,6,7,2,0,1,5 ... *(period is 9)*

- To obtain uniform deviates in the interval $[0,1]$ use $R_i = I_i/(m - 1)$

Non-uniform distributions: transformation method

- The idea is to generate sequences with a desired probability density $p(x)$ from uniform deviates

- Transformation method is based on the transformation law of probabilities:

    y - real random variable with probability density $p(y)$

    $x = f(y)$ - generate another random variable

New probability density: $\quad p(x) = p(y)\left|\dfrac{d\left(f^{-1}(x)\right)}{dx}\right| = p(y)\left|\dfrac{dy}{dx}\right|$

- **We can transform from ud to $p(x)$ if we can solve this for $f(y)$…**

E.g. Need to generate deviates x_i with the exponential density $p(x) = e^{-x}$ using ud y_i in the interval $(0,1)$ – i.e. $p(y) = 1$.

$$e^{-x} = \left|\dfrac{dy}{dx}\right| \quad \Rightarrow$$

$\Rightarrow$ *1) Generate ud y_i in the interval $(0,1)$. Have probability density $p(y) = 1$*
2) Calculate $x_i = -\ln y_i$. Newly obtained y_i will be in the interval $(0, +\infty)$ and have probability density $p(x) = e^{-x}$

- **Rarely works!** Good for exponential, normal, Lorentzian distributions

Non-uniform distributions: rejection method

- The idea of the rejection method is to generate uniform deviates, but then either "accept" or "reject" each value with a probability dictated by the local probability density.

- Consider first a trivial example: need to generate ud in the interval $(0.25, 0.75)$ from ud in the interval $(0,1)$
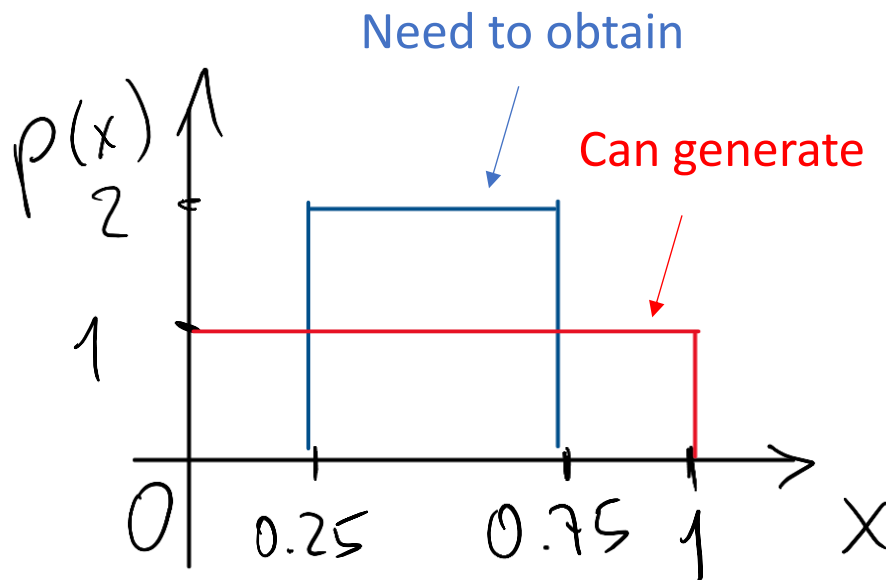
Could obtain this by trivial rescaling, but let's pretend we do not know how to scale

Need to obtain

Can generate

Could do:

1) Generate a random x_i from available ud in the $(0,1)$ interval

2) If it happens to be outside the desired interval $(0.25, 0.75)$, **reject** and ask to produce another number

- Will end up throwing away ~half of all numbers, but will obtain the desired deviates

Non-uniform distributions: rejection method

- Let's modify the task: need to obtain deviates in the interval (a, b) with the probability density $p(x)$
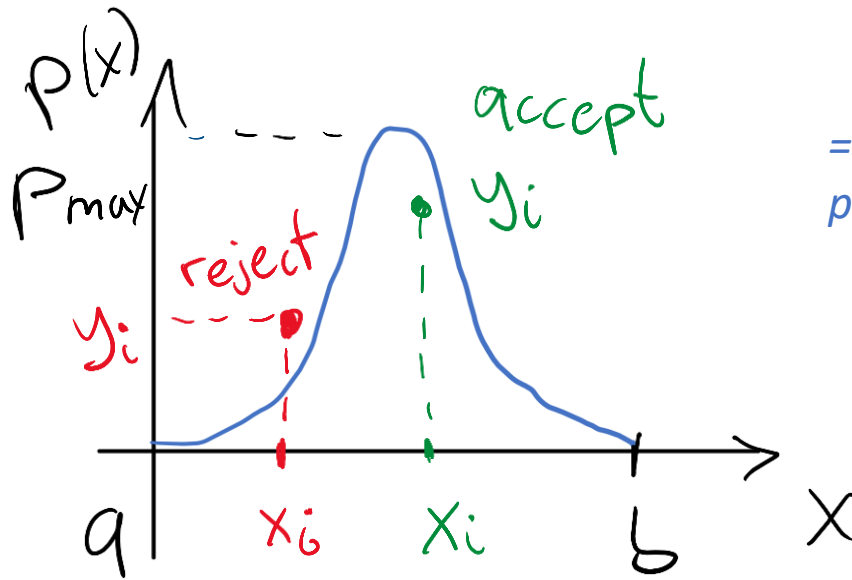
1) Generate a ud x_i in the (a, b) interval

=> Want to accept it with the probability $p(x_i)$. For this:

2) Generate another ud y_i in the $(0, p_{max})$ interval

3) If $y_i \leq p(x_i)$ then accept x_i. Otherwise, reject x_i and return to step 1

Non-uniform distributions: rejection method

- A simple geometrical interpretation

Each time we generate two random numbers:

x_i - in the interval (a, b)

y_i - in the interval $(0, p_{max})$

$\Rightarrow$ We are generating **random points inside the rectangle** *(uniformly distributed)*

If $y_i \leq p(x_i)$ then accept x_i

If the point is **inside the area under the curve** of the desired distribution, we accept it.

Otherwise – reject.

Non-uniform distributions: **rejection method**
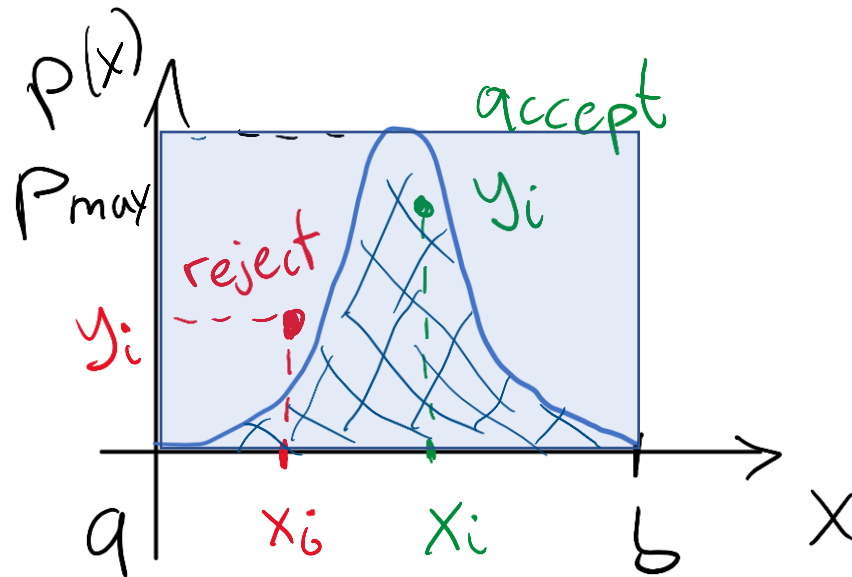
- Starting with a uniform random number generator (such as linear congruential) **can generate random numbers with any desired probability densities**

- If $p(x)$ has a narrow peak (or several), will end up "throwing away" lots of attempts

- Often need to generate numbers on an infinite interval (i.e. $a = -\infty, b = +\infty$) => linear congruential method would not work!

Non-uniform distributions: rejection method

- A better option would be to start with x_i from another non-uniform distribution, which can be obtained by the transformation method

=> Modified algorithm:
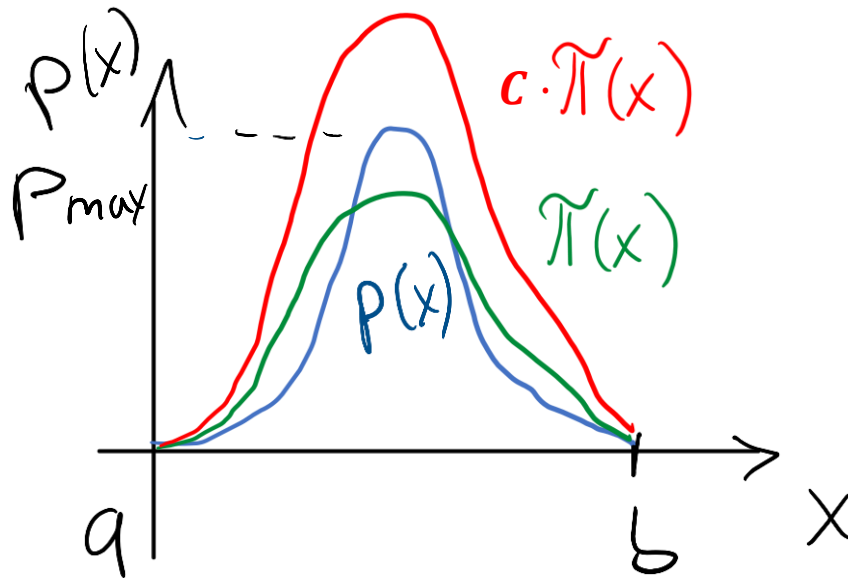
1) Use ud a_i in the $(0,1)$ interval

2) Use a transformation method to map a_i **onto the desired interval** $(\boldsymbol{a}, \boldsymbol{b})$ with the probability density function $\pi(x)$ which is as similar as possible to the desired $p(x)$

$$x_i = f(a_i)$$

At this step deviates x_i will have distribution $\pi(x)$

3) Find the minimal scaling constant C such that $\boxed{C \cdot \pi(x) \geq p(x) \; \forall \; x}$

Note: because of the normalization $\int_a^b p(x)dx = \int_a^b \pi(x)dx = 1$, the scaling constant will have to be larger than one.

4) Generate another ud y_i in the $(0,1)$ interval

5) If $y_i \leq p(x_i)/[C\pi(x_i)]$ then accept x_i. Otherwise, reject x_i and return to step 1

Non-uniform distributions: rejection method
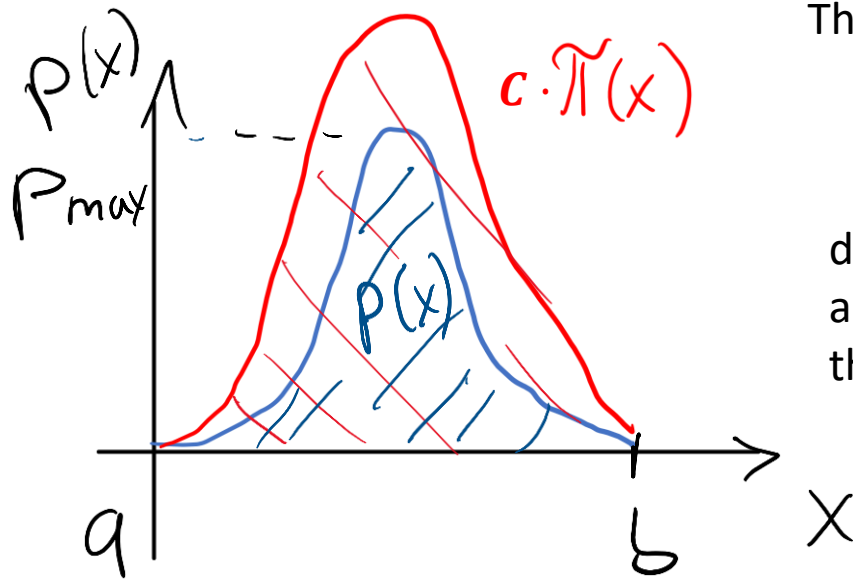
The combination of two random numbers:

$$[x_i, y_i \cdot C\pi(x_i)]$$

distributed in accordance to $\pi(x)$ in the interval (a, b)

ud in the interval $(0,1)$

generates random points under the curve $C\pi(x)$

If $y_i \leq p(x_i)/[C\pi(x_i)]$ then accept x_i

If the point is **inside the area under the curve** of the desired distribution, we accept it.
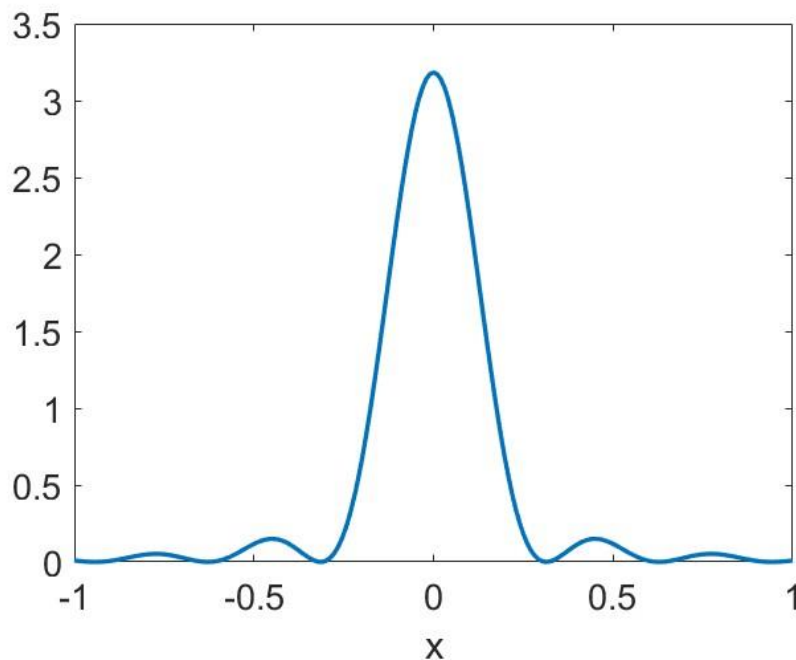
Otherwise – reject.

- Since x_i will already have a close to the desired distribution, you end up "throwing away" not as many attempts as with the previous version

An Example: generate random numbers with the probability density function:

$$p(x) = \frac{10}{\pi} \text{sinc}^2(10x)$$

Normalization factor, such that $\int_{-\infty}^{+\infty} p(x)dx = 1$

$$\text{sinc}(x) = \begin{cases} \dfrac{\sin(x)}{x}, & x \neq 0 \\ 1, & x = 0 \end{cases}$$

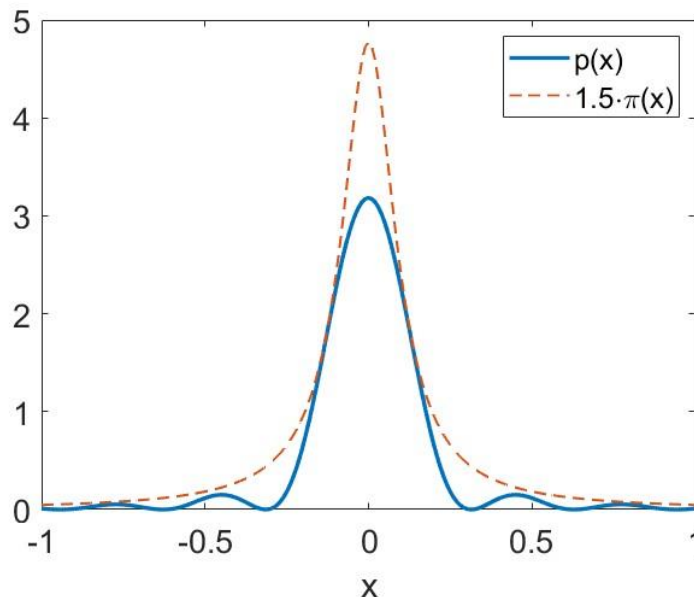Note: the interval is infinite in this case: $-\infty < x < +\infty$

$$p(x) = \frac{10}{\pi} \, \text{sinc}^2(10x)$$

- Can use transformation method:

$$x_i = \frac{1}{10} \tan\left(\pi\left(a_i - \frac{1}{2}\right)\right)$$

to convert a ud a_i in the interval $(0,1)$ to the random variable x_i in the interval $(-\infty, +\infty)$ with the distribution function

$$\pi(x) = \frac{10}{\pi} \cdot \frac{1}{1 + (10x)^2}$$

Note: check Q3 in Worksheet 3

- Observe that

$$\mathbf{1.5} \cdot \pi(x) \geq p(x) \quad \forall \, x$$

$$p(x) = \frac{10}{\pi} \operatorname{sinc}^2(10x)$$

1) Use ud a_i in the (0,1) interval
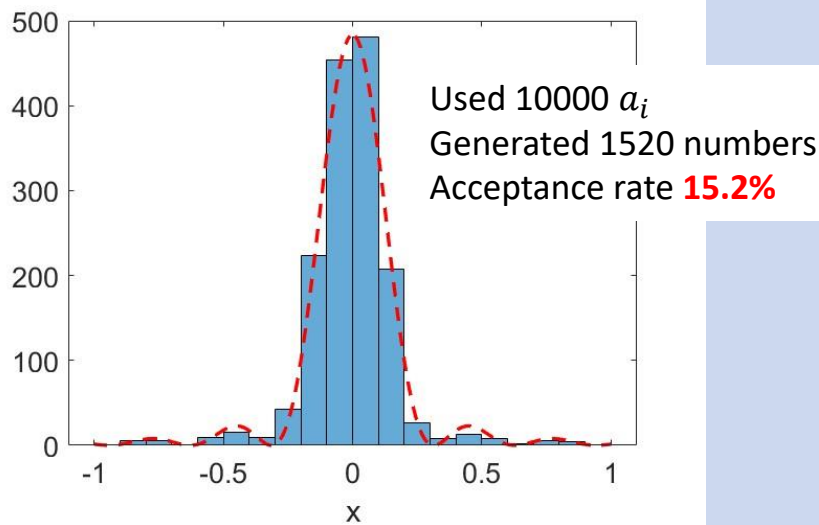
$$\pi(x) = \frac{10}{\pi} \frac{1}{1 + (10x)^2}$$

2) Scale a_i to obtain ux in $(-1,1)$ interval:

$$x_i = 2a_i - 1$$

i.e. we are ignoring anything beyond this interval due to small $p(x)$

2) transform:

$$x_i = \frac{1}{10} \tan\left(\pi\left(a_i - \frac{1}{2}\right)\right)$$

Will be generating numbers on the entire real axis, nothing is ignored!

3) Generate a random ud y_i in the interval (0,1) *["toss a coin" to accept/reject]*

4) If $\frac{10}{\pi} \cdot y_i \leq p(x_i)$ then accept x_i

4) If $1.5\pi(x) \cdot y_i \leq p(x_i)$ then accept x_i

Used 10000 a_i
Generated 1520 numbers
Acceptance rate **15.2%**

Used 10000 a_i
Generated 6667 numbers
Acceptance rate **66.7%**
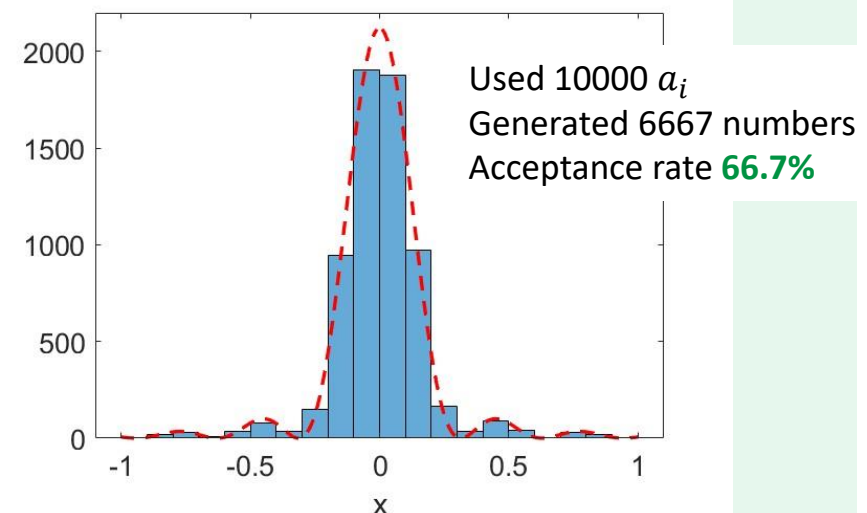
How this topic will be assessed:

- Typical questions on exam:

    - *Explain what are pseudo-random numbers, and what are the issues with using them (instead of truly random numbers) in simulations;*

    - *Suggest a method of generating random numbers with a given distribution function, explain in detail how this method works.*

    See further examples in worksheet 3

Lecture 10:

Monte Carlo methods, part 2

Monte Carlo integration for many-body problems

- Need to calculate integrals like

$$< A > = \int_\Omega A(\Gamma)p(\Gamma)d\Omega$$

in multi-dimensional phase space *(d = 6N for N classical particles)*

Here:
  Γ - a point in multi-dimensional phase space representing a particular state of the system
  $p(\Gamma)$ - the probability of the system to be observed in such state
  $A(\Gamma)$ - a macroscopic characteristic of the system *(e.g. pressure, volume, temperature)* when it is in state Γ
  $< A >$ - the "expectation value" of the measurement (the mean of many measurements)

- If we can generate random points Γ_i with probability density $p(\Gamma)$, we can estimate such integrals using Monte-Carlo integration:

$$< A > = \int_\Omega A(\Gamma)p(\Gamma)d\Omega \approx \frac{1}{N}\sum_{i=1}^{N} A(\Gamma_i)$$

Monte Carlo integration: challenges

$$< A > = \int_{\Omega} A(\Gamma) p(\Gamma) d\Omega \approx \frac{1}{N} \sum_{i=1}^{N} A(\Gamma_i)$$

- Transformation method can only generate few "standard" distributions $p(\Gamma)$, e.g. normal or Lorenzian distribution. Usually you need something else.

- Usually only tiny portions of the multi-D phase space have significant $p(\Gamma)$ => the rejection method becomes very inefficient (too many rejections)

- Often know $p(\Gamma)$ up to a normalization factor, e.g.

$$p(\Gamma) \sim \exp\left(-\frac{E(\Gamma)}{k_B T}\right)$$ *- Boltzman distribution*

Q: Why cannot we normalize this function?
A: The normalization factor is

$$C = \left[\int_{\Omega} p(\Gamma) d\Omega\right]^{-1}$$

We need to find that integral in the multi-D space – **often impossible to do analytically!**

Markov chain Monte Carlo

- Target is a sequence of $\Gamma_0, \Gamma_1, \Gamma_2$ from a known (up to a normalization factor) probability density $p(\Gamma)$

- The idea is to generate the sequence via a Markov chain – a stochastic process in which the probability of each event depends only on the state attained in the previous event.
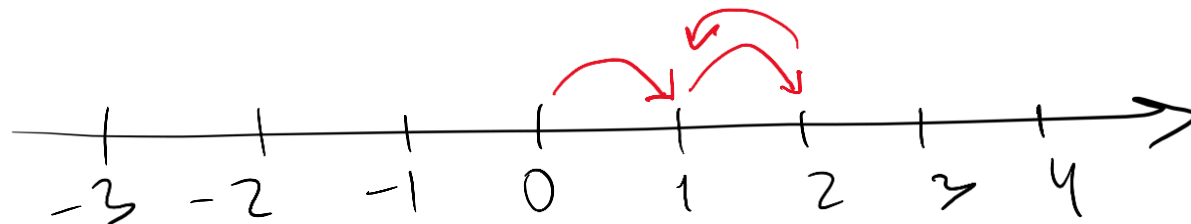
An example: a 1D random walk ("drunkman's walk")

A walk on the number line, at each step the position changes by either +1 or -1 with equal probabilities (i.e. make a step to the left, or to the right)

Will generate a sequence like:
> 0,1,2,1,2,3,2,1,2 , …. .

Each position is a state of the system

Reversible Markov chains

- Any Markov process is characterized by probabilities P_{ij} of switching between states i and j

(i.e. how often Γ_j appears immediately after Γ_i in the sequence)

In particular, for the "drunkman's walk" example we have $P_{ij} = 0.5$ if $j = i \pm 1$, and else $P_{ij} = 0$

- **Reversible Markov chain:**

If there exist a probability distribution p_i such that for any i and j states

$$\boxed{p_i P_{ij} = p_j P_{ji}}\ \text{i.e. } P_{ij} = P_{ji} \cdot (p_j/p_i)$$

- <u>Theorem:</u> if Markov chain is reversible, then each state Γ_i will be visited, in the course of a sufficiently long chain, with the relative frequency p_i

This is the central theorem of Markov chain Monte Carlo methods

The Metropolis Algorithm: generate a Markov sequence of Γ_i

1) Start with a random state Γ_0 *(in d-dimensional phase space)*

2) Pick a "trial" next state Γ_t and calculate $r = p(\Gamma_t)/p(\Gamma_i)$, where Γ_i is the current state *($\Gamma_i = \Gamma_0$ if you just started)*

3) If $r \geq 1$ then "accept" this trial state, i.e. $\Gamma_{i+1} = \Gamma_t$
       (i.e. always move if the trial point has a larger probability)

4) If $r < 1$ then generate a uniform deviate x from the range $(0,1)$.

    If $x \leq r$ then "accept" the trial state, i.e. $\Gamma_{i+1} = \Gamma_t$

    If $x > r$ then "reject" the trial state, i.e. do not generate Γ_{i+1} at this stage

    (i.e. move to the trial point with a probability r)

5) Repeat 2-4 for as many times as needed to generate the sequence.
 **Then calculate** $< A > \approx (1/N) \sum_j A(\Gamma_j)$

Note: the procedure of acceptance/rejection in steps 2-4 ensures that the generated sequence is a **reversible Markov chain:** for any $p_i > p_j$ we have $P_{ji} = 1$, $P_{ij} = p_j/p_i$
$$\text{i.e.} \quad p_i P_{ij} = p_j P_{ji}$$

=> For a sufficiently long sequence we will visit each Γ_i with the relative probability $p(\Gamma_i)$
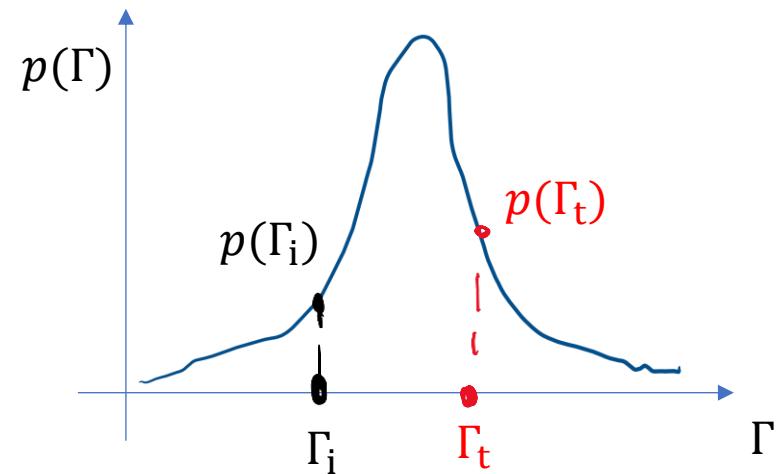
Metropolis vs Rejection method

- Both methods generate a random sequence of states Γ with a desired probability density function $p(\Gamma)$, at each step accepting or rejecting a "trial" next state
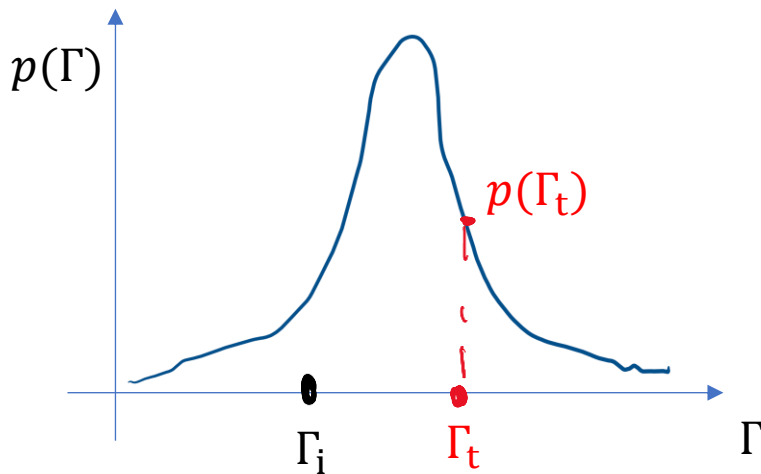
Accept Γ_{i+1} with the probability $p(\Gamma_t)$

- If $p(\Gamma_t) > p(\Gamma_i)$ - accept Γ_t
- Else accept Γ_t with the probability $r = p(\Gamma_t)/p(\Gamma)$

- For probability functions with strong peaks, the Metropolis method will have a much greater acceptance rate (i.e. much more efficient)

- Metropolis algorithm only requires relative probabilities $p(\Gamma_t)/p(\Gamma_i)$ => no issues with not knowing the normalization factor for the probability density $p(\Gamma)$

An Example of MC simulation: 2D Ising model

- A simple model of a phase transition between a low temperature ordered phase (ferromagnet) and high temperature disordered phase (paramagnet)
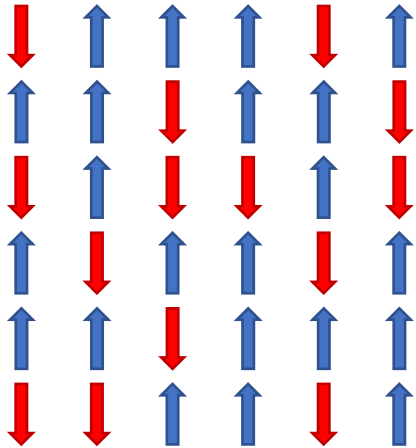
- $N \times N$ 2D lattice of spins

- Energy of a particular configuration of spins:

$$E(\Gamma) = -J \sum_{[ij]} \vec{S}_i \cdot \vec{S}_j$$

Sum over nearest neighbours

- $J > 0$ – coupling constant. Gives low energy if all spins are aligned

- Phase space Ω is discrete. Probability of state Γ is (Boltzman distribution):

$$p(\Gamma) = \frac{\exp[-E(\Gamma)/k_B T]}{\sum_\Gamma \exp[-E(\Gamma)/k_B T]}$$

Normalization

An Example of MC simulation: 2D Ising model

- Can compute e.g. mean magnetisation for a given temperature T:

$$\langle m \rangle = \sum_{\Gamma} p(\Gamma) m(\Gamma), \quad \text{where} \quad m(\Gamma) = n_\uparrow - n_\downarrow$$

- Computing this directly is practically impossible: e.g. for a 6×6 lattice there are $\sim 7 \cdot 10^{10}$ states Γ *(different combinations of spins up and down)*

- Use MC simulations instead:

  - Start with a random configuration

  - Generate a random sequence of configurations, e.g. by using Metropolis algorithm

  - Let the system "settle", then start averaging over configurations

Note: *what "settle" means here and why is it needed?*

If you start with a low-probable state, in a relatively short sequence of random states, this state will obscure the statistics. You want to reach a statistically significant region of phase space, before you start averaging. The Metropolis algorithm will naturally "drive you there"…
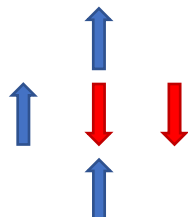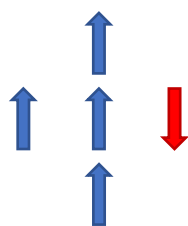
But things may become more complicated if you have several isolated statistically significant regions!

An Example of MC simulation: 2D Ising model

- MC simulation with Metropolis algorithm

  1) Start with a random configuration (Γ_0)

  2) To generate a new trial configuration flip one spin:

  Γ_i: $E(\Gamma_i) = -J(-1 + 1 - 1 - 1) = +2J$

  Γ_t: $E(\Gamma_t) = -J(+1 - 1 + 1 + 1) = -2J$

  3) Calculate $r = \dfrac{p(\Gamma_t)}{p(\Gamma_i)} = \exp\left[-\dfrac{E(\Gamma_t) - E(\Gamma_i)}{k_B T}\right]$

  4) In this case, $r > 1$ *=> accept the flip*

  5) Carry on flipping…

An Example of MC simulation: 2D Ising model

- MC simulation with Metropolis algorithm

2) To generate a new trial configuration flip one spin:

$$\Gamma_i: \qquad E(\Gamma_i) = -J(+1-1+1+1) = -2J$$

$$\Gamma_t: \qquad E(\Gamma_t) = -J(+1-1-1+1) = 0$$

3) Calculate $r = \frac{p(\Gamma_t)}{p(\Gamma_i)} = \exp\left[-\frac{E(\Gamma_t)-E(\Gamma_i)}{k_B T}\right]$

4) In this case, $r < 1$

     => generate a random number x in the interval (0,1)
   *if $x \leq r$ – accept the new state*
       else reject it

5) Carry on flipping…

An Example of MC simulation: 2D Ising model

- Sweep test spin through entire lattice -> one "MC cycle"
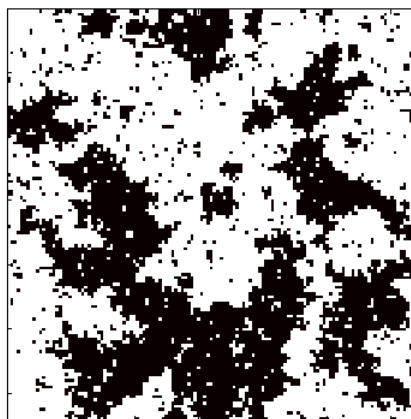
- Run enough cycles for the system to "settle"

Note: this is not a dynamical simulation. By "settling" I mean reaching statistically important regions of the phase space

- Then start averaging over Γ to obtain $m(\Gamma)$

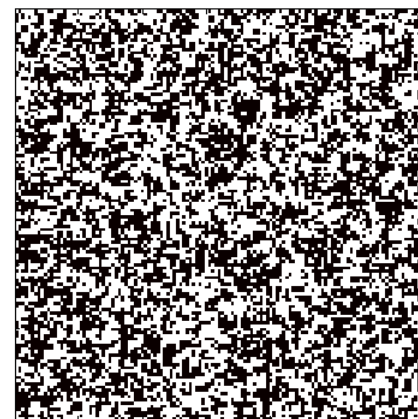- *Example snapshots for 128x128 grid:*
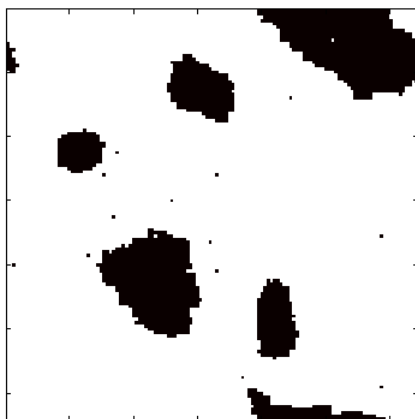
T = 1.14, M = 0.694, E = -3.84

$$T = \frac{1}{2}T_c$$

T = 2.22, M = 0.188, E = -2.79

$$T \approx T_c$$

T = 4.54, M = 0.009, E = -0.89

$$T = 2T_c$$

Note 2: this is not a dynamical simulation! The pictures are merely snapshots of "highly probable" *(read "typical")* states at the given temperature

How this topic will be assessed:

- Typical questions on exam:

  - *Explain why it is difficult to normalise a probability distribution function in multi-dimensional phase spaces. Why is this an issue for Monte-Carlo simulations?;*

  - *Name a method which does not require normalisation of the probability distribution function to generate a sequence of random numbers. Explain how it works.*