Computational Physics

Lecture 7

Organisation of the course - 2 versions

3 ECTS course

6 weeks - 1 project:

- Molecular Dynamics simulation of different phases of matter

6 ECTS course

12 weeks - 3 projects:

- Molecular Dynamics simulation of different phases of matter (6 weeks)
- Monte-Carlo simulation of the the Ising model (3 weeks)
- Individual projects (3 weeks)

Organisation of the course - Program

Mar 22 – Introduction to Monte-Carlo and Ising model

Mar 29 – Good Friday Bank Holiday → No class

Apr 05 – Choice of units, energy of the system

Apr 12 – Individual discussions

Second project deadline: Thursday April 18 (midnight)

Groups

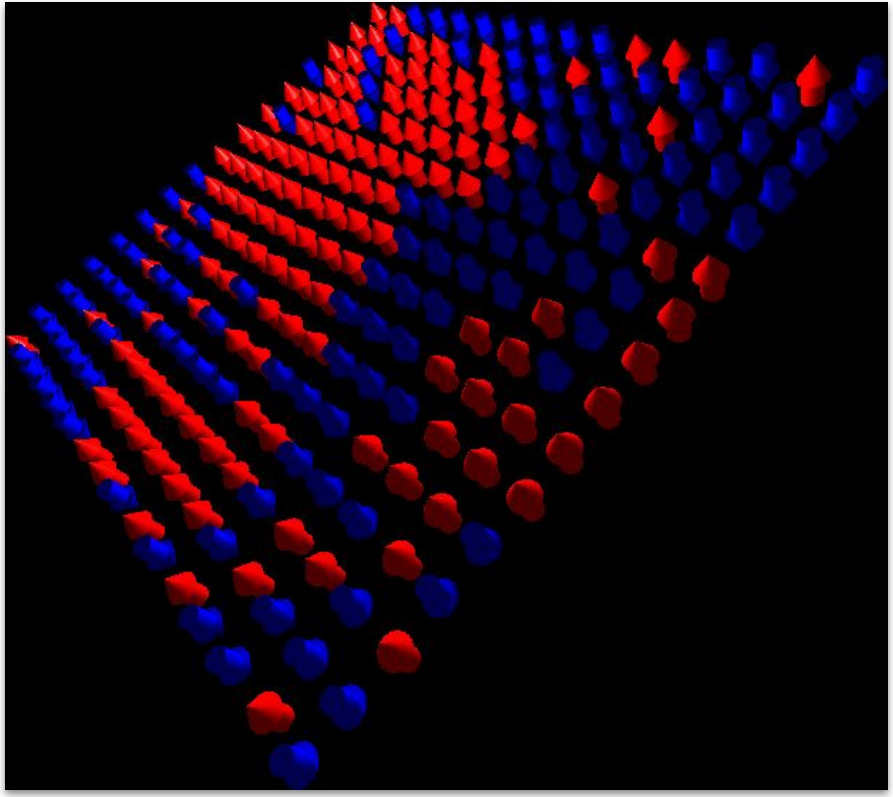
- Please register your groups on *Brightspace*.

- There is a new group category for the 2nd project.

Questions?

Project 2: Monte-Carlo simulation of the Ising model

Goal: Ferromagnetic behaviour in a 2D material



Goal: Ferromagnetic behaviour in a 2D material

We consider a square lattice of $N \times N$ spins that can take two values: up (+1) and down (-1).

The Hamiltonian for this system is:

$$\mathcal{H} = -J \sum_{\langle i,j \rangle} s_i s_j - H \sum_i s_i$$

Where the first sum runs over pairs of neighbouring spins. We take $J = 1$, $H = 0$. $H > 0$ corresponds to a system with an external magnetic field.

Ising Model - Goals

Study the magnetization as a function of the equilibrium temperature of the system.

At temperature below the critical temperature T_c the system should be spontaneously magnetic. For a 2D model, we know (analytically):

$$\frac{k_B T_c}{J} = \frac{2}{\ln\left(1+\sqrt{2}\right)} \approx 2.269.$$

Try to see that behaviour by testing T=$1.0 \rightarrow 4.0$ in steps of 0.2.

Goal: Ferromagnetic behaviour in a 2D material

This system **cannot be attacked** by a technique like Molecular Dynamics.

Question for you: Why?

Goal: Ferromagnetic behaviour in a 2D material

This system **cannot be attacked** by a technique like Molecular Dynamics.

Question for you: Why?

We are interested in the **behaviour at thermal equilibrium**.
Statistical physics (again!) can hence help us.

Time for some integrals…

Statistical physics integrals

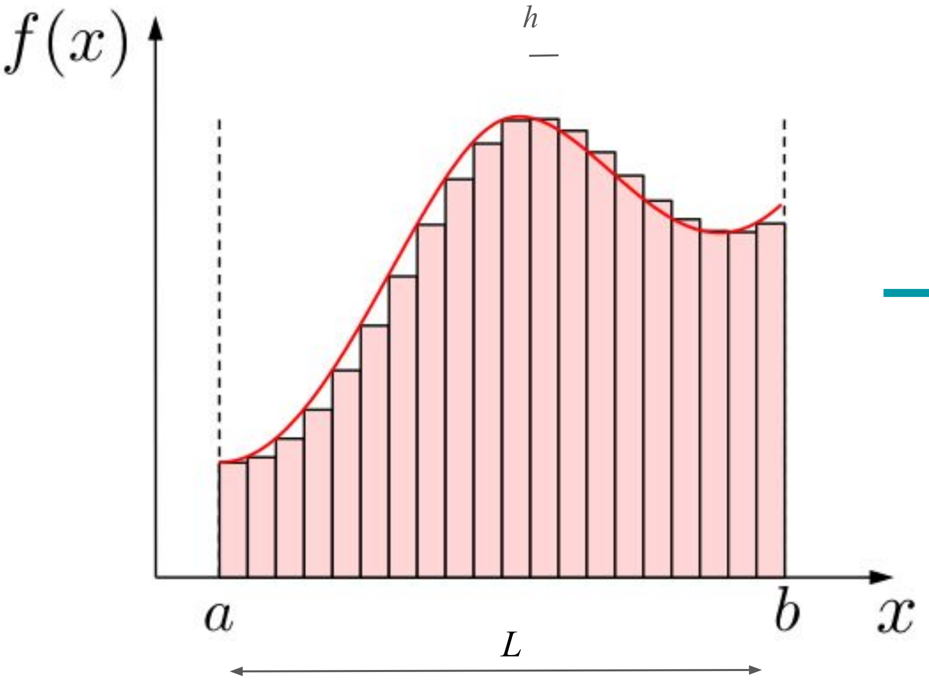
Most integrals will look like: (as always $\beta = 1/k_B T$)

$$\langle A \rangle = \frac{1}{Z} \int A\left(x\right) e^{-\beta H\left(x\right)} dx$$

With the **integral running over phase-space**. For instance all the positions, momenta and spin of all the particles in a system.

For 1000 particles in 3D with 2 spin states, that is a *12000-dimensional* integral!

Numerical integration (in 1D) - The “classic” way



$$\int_a^b f(x)\,dx \approx h \sum_{i=0}^{N-1} f_i,$$

Numerical integration - Error analysis in 1D

1) Assume the maximal slope of f is M.

2) Then the maximal error in a bin is at most $M\,h\,/\,2$.

3) The total error is thus $h\,N\,M\,h\,/\,2 = (b - a)\,M\,h\,/\,2 = L\,M\,h\,/2$.

4) As $h \propto 1/N$, we get $L\,M\,/\,2\,N$:
   the **total error is thus proportional to $1/N$.**

Numerical integration - Error analysis in dD

1) Assume the maximal slope of f is M.

2) Then the maximal error in a bin is at most $M\,h^{k} / 2$. (k ~ 1)

3) The total error is thus $h^{d}\,N\,M\,h^{k} / 2 = L^{d}\,M\,h^{k} / 2$.

4) As $h \propto 1/N^{d}$, we get $L^{d}\,M / 2\,N^{k/d}$:
the **total error is thus proportional to $1/N^{1/d}$.**

Monte-Carlo integrals

Let’s start with a 1D integral over a finite interval:

Here we take a uniform prob. distribution p(x) = 1 / (b-a).

$$\int_a^b f\left(x\right) dx = \left(b-a\right) \int_a^b \frac{1}{b-a} f\left(x\right) dx$$

$$= \left(b-a\right) \boxed{\int_a^b p(x) f\left(x\right) dx}$$

“Expectation value of f for the probability distribution p(x).”

The x_i are distributed according to p(x).

Monte Carlo integrals

In summary:

“I can compute the integral of f over an interval by taking the average of a randomly chosen series of (uniformly distributed) values of f in this interval.”

Monte Carlo integrals

How good is it? Well...

Let's check the variance:

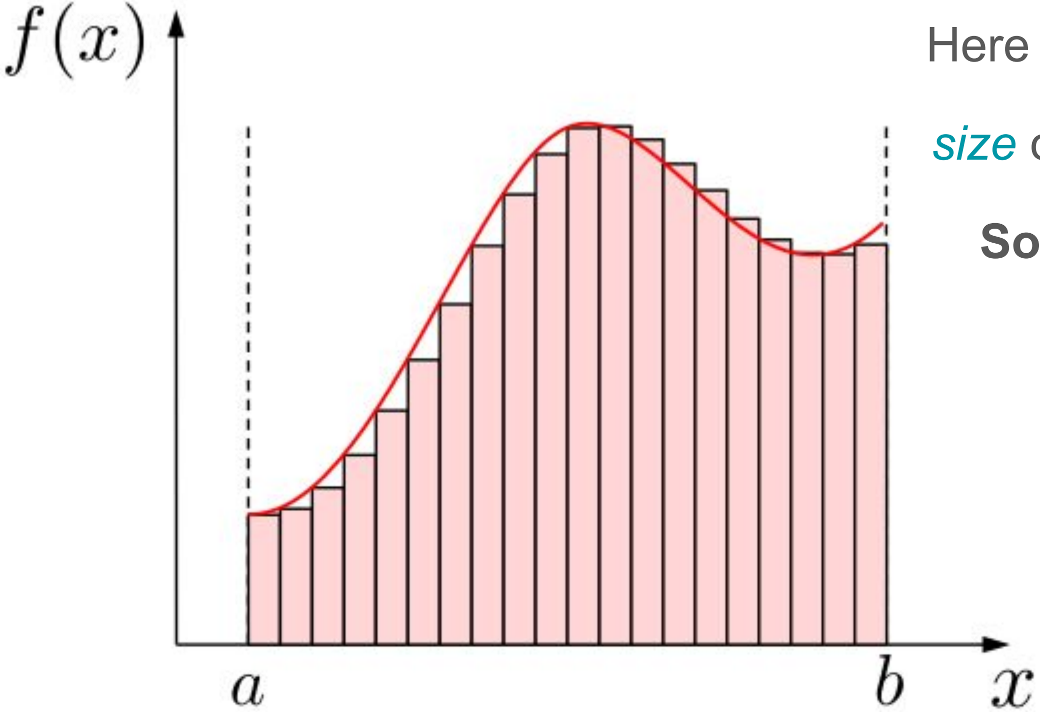
$$\sigma^2 = \left\langle \left(\frac{b-a}{N} \sum_{i=1}^{N} f_i \right)^2 \right\rangle - \left(\left\langle \frac{b-a}{N} \sum_{i=1}^{N} f_i \right\rangle \right)^2$$

$$\sigma^2 = \frac{(b-a)^2}{N} \left(\overline{f^2} - \overline{f}^2 \right)$$

→ **Error goes as $1/\sqrt{N}$**

Monte Carlo integrals

An approach where we regularly space N rectangles would do much better!



Here the error is proportional to the

size of each of the N sub-intervals.

So, overall, it goes as 1/N !!!

Why Monte-Carlo integrals?

What happens in higher dimension?

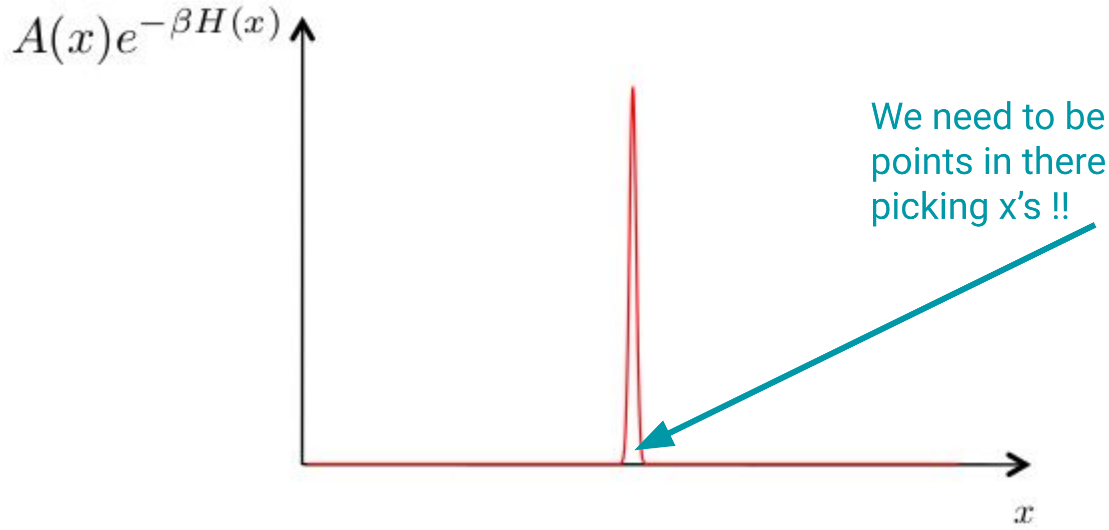
The Monte-Carlo method does not care about dimensionality!

Error always goes as $1/\sqrt{N}$ irrespective of dimension

Questions?

Statistical physics integrals

Monte-Carlo integral would work well *in principle*. However, randomly sampling the interval would work badly because of the shape of the function:



We need to be very lucky to get points in there when randomly picking x’s !!

Importance sampling

We want to compute $\langle A \rangle = \frac{1}{Z} \int A(x)\, e^{-\beta H(x)} dx$

If, instead of using a uniform sampling, we consider the **Boltzmann weights as our probability distribution**, we can write:

$$\int A(x)\, p(x)\, dx \approx \frac{1}{N} \sum_{i=1}^{N} A(x_i)$$

With the x_i's sampled according to the weights!

The x_i are distributed according to $e^{-\beta H}$.

Questions?

Metropolis algorithm and Markov chains

We want to be able to randomly “walk around” the part of phase space where the integrand (here the term $e^{-\beta H}$) is non-zero.

This is what **Markov chains** are for.

In other words, we want an algorithm that helps us draw the random points where the integral is sampled *effectively*.

Or: Given the current point, what point should I pick next?

Markov chains

Basic definitions:

- We define x_i as a state of the system (e.g. the position and velocity of all particles. Or the spin configuration). That's what the integral runs over.

- We assume that the "next” state x_{i+1} in a chain only depends on the current state x_i, not on whatever happened before. (*The system has no memory*)

- We define $T(x \rightarrow x')$ the probability to go from state x to state x'. (e.g. the probability to flip 1 spin in our lattice).

Markov chain master equations

We can then write for the *probability of the system to be in a given state* at the next step:

$$p(x, i+1) = p(x, i) - \underbrace{\sum_{x'} p(x, i)\, T(x \rightarrow x')}_{\text{Probability to move to } \textbf{\textit{any}} \text{ other state } x'} + \underbrace{\sum_{x'} p(x', i)\, T(x' \rightarrow x)}_{\text{Probability that } \textbf{\textit{any}} \text{ other state } x' \text{ transitions to } x}$$

Probability to be in state x at current step i

Probability to be in state x at the next “time” $(i+1)$

Markov chain master equations - Simplifications

If we are at equilibrium then the probability distribution should be constant.

Mathematically, $p(x, i{+}1) = p(x, i)$

Which means: $$\sum_{x'} p(x,i)\,T(x \rightarrow x') = \sum_{x'} p(x',i)\,T(x' \rightarrow x)$$

We know all the $p(x)$, let’s focus on the $T(x \rightarrow x')$.

Detailed balance

The most common way of solving the master equation is to additionally require the so-called *detailed balance*.

$$p\left(x,i\right)T\left(x\rightarrow x'\right)=p\left(x',i\right)T\left(x'\rightarrow x\right)$$

This means the system only makes small adjustments, no large reshuffling.

Last assumption

We now need to assume something for $T(x \to x')$.

We write:

$$T\left(x \to x'\right) = w_{xx'} \times A_{xx'}$$

Trial probability:

Probability to ***test*** the state x' if we are at state x.

Acceptance probability:

Probability to ***accept*** the state x' if we are at state x.

Last assumption

We will take $w_{xx'}$ to be symmetric and uniform. That means:

“From any state, we can *try* to go to any neighbouring state with equal probability.”

Our master equation thus reduces to:

$$\frac{A_{xx'}}{A_{x'x}} = \frac{p(x')}{p(x)}$$

Solution & Metropolis algorithm

One solution to

$$\frac{A_{xx'}}{A_{x'x}} = \frac{p\left(x'\right)}{p\left(x\right)}$$

is:

“If state is more favorable → Go there”

“If state is less favorable → Try your luck”

$$A_{xx'} = \begin{cases} 1 & \text{if } p\left(x'\right) > p\left(x\right) \\ p(x')/p(x) & \text{if } p\left(x'\right) < p\left(x\right) \end{cases}$$

Quick summary

- We want to perform an integral using the Monte Carlo method and Boltzmann weights.

- We want to sample a probability distribution.

- We wrote down an equation telling us how to move from the probability of being in one state to the probability of being in that state at the next step.

- With some general assumptions we simplified the equations to two terms: one telling us what state to try as our next state ($w_{xx'}$) and one whether to go to that state ($A_{xx'}$).

- We have equations for both.

Questions?

Metropolis-Hastings algorithm

Finally, can now nicely sample the probability distribution using this algorithm:

1) Start with a random state x_0 (say all spin up. Or all spin random)

2) Generate a different state x' (i.e. a different configuration of spins.)

3) - If $E(x') < E(x)$ → $A_{xx'} = 1$ → We move to that new state.
   - If $E(x') > E(x)$ → Move to the new state with probability $p(x')/p(x)$.

4) Go back to (2).

Recall:

$$p(x) \propto e^{-\beta H(x)}$$

Summary

We constructed an algorithm that will move the states towards minimal energy! Starting from *any point*, we will move towards a minimum.

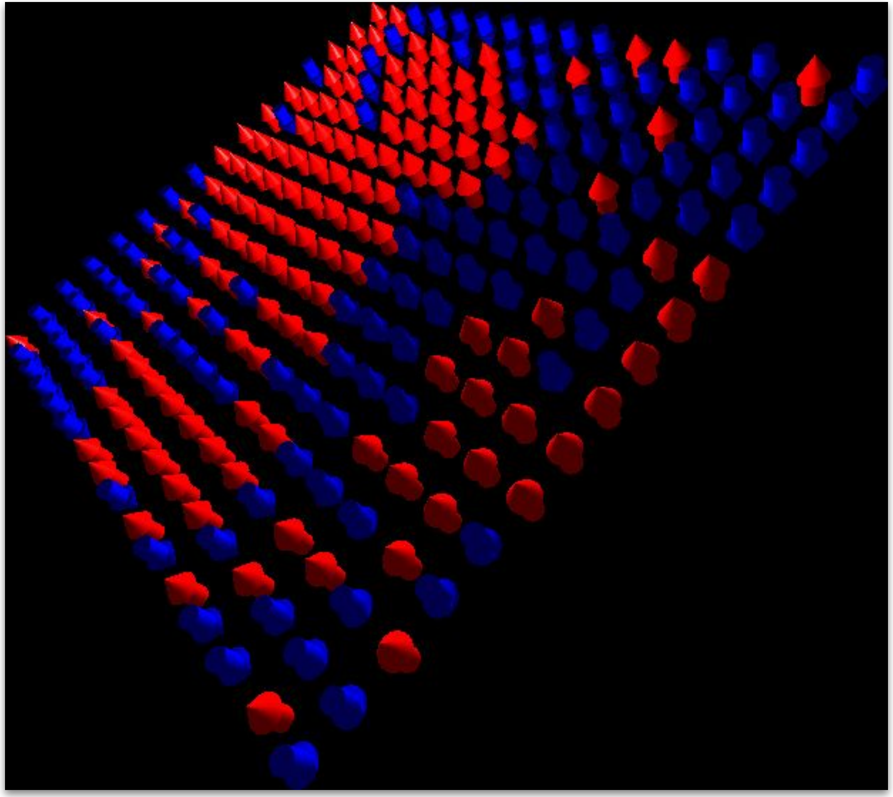
At every step, we can either go to a state with lower energy, or have a small, temperature-dependant, chance of going to higher energy.

Once we have reached a state close to the minimum (i.e. equilibrium for this temperature), we can keep going to sample the relevant part of the integral.

We can use this to compute the total spin or magnetisation of the system at a given temperature T.

Questions?

The Ising model



The Ising model - practicalities

An array of $N \times N$ spins. Let's take N=50.

Each spin can be in state +1 or -1.

How do we move from one state to the next? We know $A_{xx'}$ already.
To keep it simple we choose:

$$\omega_{xx'} = \begin{cases} 1/N^2 & \text{if } x \text{ and } x'\text{differ by one spin} \\ 0 & \text{otherwise.} \end{cases}$$

Trial state: Create a new state which differ from the current one by ***exactly one*** spin flip.

The Ising model - Algorithm

1) Start with a random state x_0 (say all spin up. Or all spin random)

2) ***Try*** a different state x'
(i.e. a different configuration by flipping just one spin!)

3) - If $E(x') < E(x)$ → Keep that new state.
- If $E(x') > E(x)$ → Keep that state with probability $p(x')/p(x)$. Otherwise stay.

4) Go back to (2).

Recall, the energy is:

$$\mathcal{H} = -J \sum_{\langle i,j \rangle} s_i s_j - H \sum_i s_i$$

Ising Model - Goals

At every step **measure the total magnetisation** M (sum of all spins).

Take into account **periodic boundary conditions** to mimic an infinite lattice.

Plot the magnetisation per spin $m = M / N^2$ as function of time.
We can use a full “sweep” of the mesh as our time unit.
That means give every spin a chance to flip per “unit of time”.

Ising Model - Goals

Study the magnetization as a function of the equilibrium temperature of the system.

At temperature below the critical temperature T_c the system should be spontaneously magnetic. For a 2D model, we know (analytically):

$$\frac{k_B T_c}{J} = \frac{2}{\ln\left(1+\sqrt{2}\right)} \approx 2.269.$$

Try to see that behaviour by testing T=$1.0 \rightarrow 4.0$ in steps of 0.2.

Coding aspects

A few things to think about before typing:

What is the best way to represent the array of spins in the code?

How can you decompose the problem into functions?

What are the constants of the problem and how to store them?

How do you best pick random spins to flip and compute the flip probability?

First Milestone

- Code up the system. Try N=10, N=20 and N=50.
- Try the system at different equilibrium temperatures from T=1 to T=4.
- For each temperature it is good to try different initial states to see how they converge to the magnetisation value. Plot the magnetisation as a function of time.

Happy coding!