contact info: Astrid de Wijn <dewijn@fysik.su.se>
office: C4:3035

General Practical Information:
- give me your email addresses
- course website: http://www.syonax.net/physics/FK7029-2014
- book: Understanding Molecular Simulation by (Frenkel & Smit) available electronically from library
- prerequisite: some statistical physics
- course consists of:
  - lectures + small number of lab hours
  - programming projects } ⟶ grade 50/50
  - written exam
  - hand-in exercises for bonus points
- There is a parallel course in Stat Phys taught by Supriya Krishnamurthy, which I recommend
- I will try to make hand-outs or just copy my notes or something

About the programming
- recommended language: C/C++ with unix-like OS and GNU tools
- start on time. you will get stuck
- lab sessions are to help you along. Do the rest at home.
- deadlines 24 Feb for first project 17 March for second.

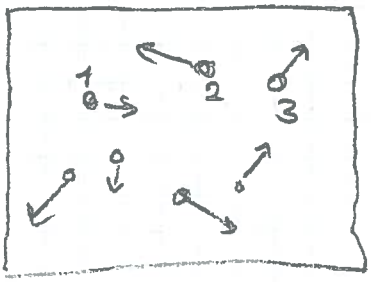
About the hand-in exercises
- deadline 1 week after posted
- highly recommended. (similar to exam questions)

First lecture: basic intro. give you some idea of what and why
example: Lorentz gas.

What's the problem?

statistical physics ≡ many particles.

$\vec{r}_1, \vec{v}_1, \vec{r}_2, \vec{v}_2, \vec{r}_3, \vec{v}_3 \ldots \ldots \vec{r}_N, \vec{v}_N$



example gas in a box or lecture room
$N \sim N_A \approx 6.10^{23}$

in general, particles interact: pot. energy
- $V = \sum_i V_0(\vec{r}_i - \vec{r}_j)$
- or more complicated $V(\vec{r}_1, \ldots \vec{r}_N)$

Equations of motion (classical)

$$\dot{\vec{r}}_1 = \vec{v}_1 \qquad \dot{\vec{r}}_2 = \vec{v}_2 \qquad \dot{\vec{r}}_3 = v_3$$

$$\dot{v}_1 = -\frac{1}{m_1} \frac{\partial V(\vec{r}_1 \ldots \vec{r}_N)}{\partial \vec{r}_1}$$

$$\dot{v}_2 = -\frac{1}{m_2} \frac{\partial V(\ldots)}{\partial \vec{r}_2}$$

6N coupled, nonlinear differential equations

⇒ too difficult / too much work to solve with pen and paper.

not interested in individual particles but in <u>global averages</u>

(pressure, density profile, current, etc...)

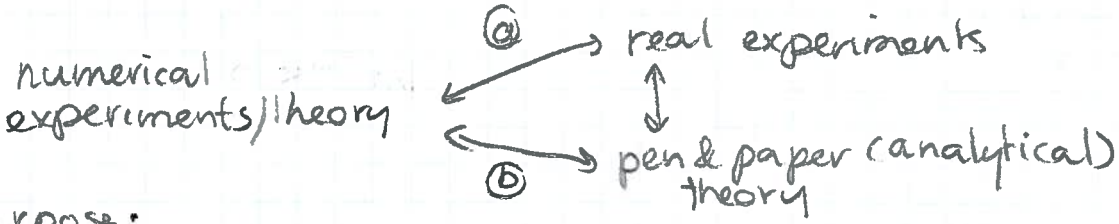
Equilibrium Statistical Physics is a powerful general theory for N large.

- complex interactions make it hard to use
- does not apply in systems out of equilibrium
  ⇒ some ad-hoc approaches still exist, but cumbersome and very specific (f.e. kinetic theory of gases)

air in this room: not in equilibrium (we move and breathe, pressure outside changes, temperature, humidity)

What to do when pen & paper won't do?
[not enough time (too lazy), or too stupid]

=> simulate it (computer does the work!)



Purpose:
@ quantitative, realistic simulations, compare to experiments
b simple model systems just beyond theoretical understanding to support development of analytical theory

in simulations you can switch stuff on & off and see everything

---

Types of simulations, most are:
- Molecular Dynamics (MD) or
- Monte Carlo (MC)
(one programming project each)

MD: solve the classical time-dependent, nonlinear equations of motion explicitely (of molecules or something)

example gas in box

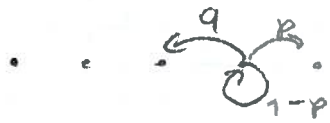
$\gamma = (r_{1,} r_{2,} r_{3} \ldots r_{N,} v_{1,} v_{2} v_{3} \ldots v_{N})$ phase space variable } → dynamical system

eq. o. m $\dot{\gamma} = f(\gamma)$

MC: some kind of random sampling

→ to replace part of dynamics (usually based on stat. phys theory)

example random walk on a line



What does the probability distribution look like after a while

- take one walker, draw random steps, let it run around a
- repeat many times
- not exact solution but reasonable approximation

Truth: line between MD & MC is vague

FK7029
2014
(4)

Structure of a simulation

Step 1 initialise

- declare variables double *r double *v
- allocate memory malloc(3N*sizeof(double))
- set initial conditions
  $r_1 = \ldots$, $r_2 = \ldots$
  $v_1 = \ldots$, $v_2 = \ldots$

Step 2 Run integrate eq. of motion or MC algorithm

Step 3 Analyse (on the fly or at the end) calculate interesting quantities

Step 4 Output (on the fly or at the end)

...

+ additional fiddling of course

There are some big codes that take care of a lot of the details for you, such as LAMMPS, Gromacs

---

Practical stuff about programming

- bugs (most time spent on finding them)
- available resources (cpu & memory)

tips for bugs and related hassle

- think before you type
- keep it simple programmer time vs computer time
- use subroutines & functions
- use descriptive variable names
- write comments
- indent the code
- use a smart editor (folds, syntax highlighting)
- use debugging tools such as gdb, valgrind, cachegrind.
- **KEEP THE OVERVIEW**

show folds in vim & indentation in dipole code

Scientific considerations

- model ≠ real system
- accuracy ⟷ resources

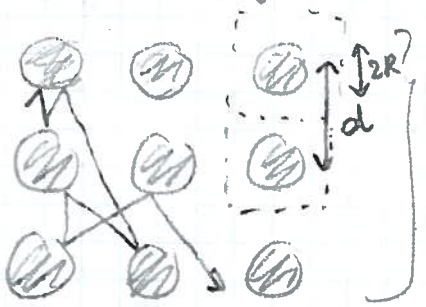
⇒ model considerations
- basic interactions
- what quantities to keep constant (P, V, N, T, E)
- external forces
- boundary conditions
- integration algorithms
- target quantities
- particle number
- initial conditions
- time scales.

Simple example of some of these issues

Particle bouncing between other particles?
(i.e asymmetric gas mixture, classical picture of electron in solid)

Lorentz gas: point particle + fixed array of spherical scatterers

regular L-gas.



= model (in 2d here)

pinball!

Let's say we do MD.

Initialise (step 1)
variables: $\vec{r}$ $\vec{v}$ t?
parameters v_0, R, d.
initial conditions?

→ which ones should you avoid?

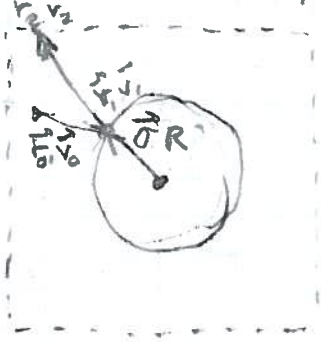
gives scatterer edge positions (do not want to store ∞ scatterers)

step 2 run (MD) Sketch integration algorithm

time increments? $t \to t+\Delta t$ $(x,y) \to (x+v_x\Delta t,\ y+v_y\Delta t)$ (1)

Not good

⇒ calculate intersections to scatterer or boundary or area around scatterer



Sinai billiard
- single circular scatterer
- periodic boundaries

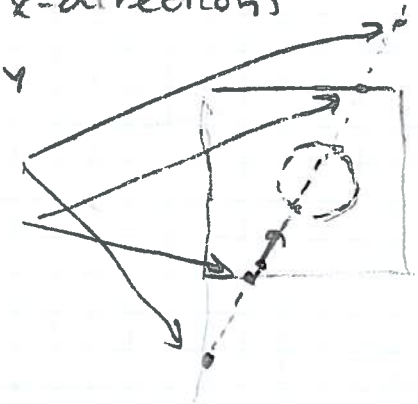
(assume scatterer is at (0,0)

boundaries: $x + v_x \Delta t = \pm \frac{d}{2}$ (x-direction)

$y + v_y \Delta t = \pm \frac{d}{2}$ y

$\Rightarrow \Delta t = \frac{1}{v_x}\left(\pm \frac{d}{2} - x\right)$

$\Rightarrow \Delta t = \frac{1}{v_y}\left(\pm \frac{d}{2} - y\right)$.



scatterer collision.

$$(x + v_x \Delta t)^2 + (y + v_y \Delta t)^2 = R^2$$

$$\Rightarrow \Delta t^2 v^2 + \underbrace{2(x v_x + y v_y)}_{b} \Delta t + x^2 + y^2 = R^2$$

$\Rightarrow$ collision if $b^2 - 4v^2(x^2+y^2) > 0$

$$\Delta t = \frac{1}{2v^2}\left[-b \pm \sqrt{b^2 - 4v^2(x^2+y^2)}\right]$$

So, max 6 possible Δt.

earliest one that is in the future is correct.

update now with ①, and if collision, reflect $\vec{v}$ according to

$(v_x, v_y) \rightarrow (1 - 2\sigma)(v_x, v_y)$ $\quad \vec{\sigma} = \frac{1}{R}(x, y)$

then put back in $[-d/2, d/2]\,[-\frac{d}{2}, \frac{d}{2}]$

produces sequence of boundary crossings and collisions

Question: how best to implement this (minimising cpu time)

Every thing you do cost cpu effort!

Function	time needed
drand48()	1
*, +	0 (1/10)
sin	5
cos	5
sincos	5
log	5
exp	4
sqrt	2

beware of these
lots of +, * etc to calculate these
(though well-optimised, still expensive)

use this if you need sin & cos.

initialisation happens once
update algorithm gets used over and over.

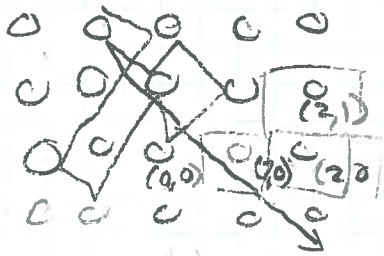
make sure only do calculate sqrt in scatterer collision time when needed and only once!

use cachegrind to see what cpu spends much time on

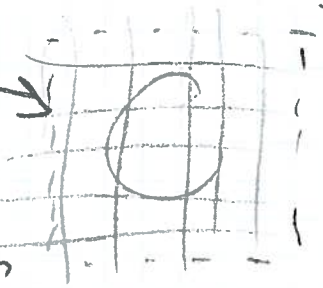
Lorentz gas step 3: analyse.

Where does the particle go?

① does it stay in the same neighbourhood?
② where in the square does it go?



for
① store sequence of squares (update along with x,y)



② probability distribution: binning
how much time spent in each bin?
(also requires extra calculation)

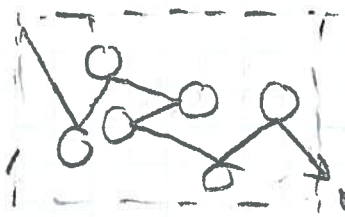
Show demonstration sim. — with some initial condition
— with bad initial condition

result of analysing: ① goes everywhere
② goes everywhere

unless weird initial conditions! (never-colliding trajectories)

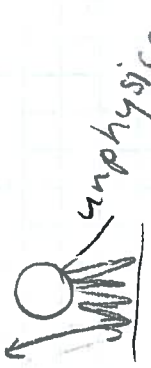
(lecture 1 end)

Slightly more complicated: Random Lorentz gas



N randomly placed scatterers
↑ large, but can't be infinite
$N_A \approx 6.10^{23}$ >> what we can store

What to do when it leaves the stuff you know? not reflecting boundary



← periodic with big box!

pro: consistent, deterministic
con: still possibly funny periodic trajectories (not so bad if big enough box)

OR generate on the fly and forget afterwards randomly

⇒ Monte Carlo

pro: long trajectories with fully random scatterers
con: may not be consistent