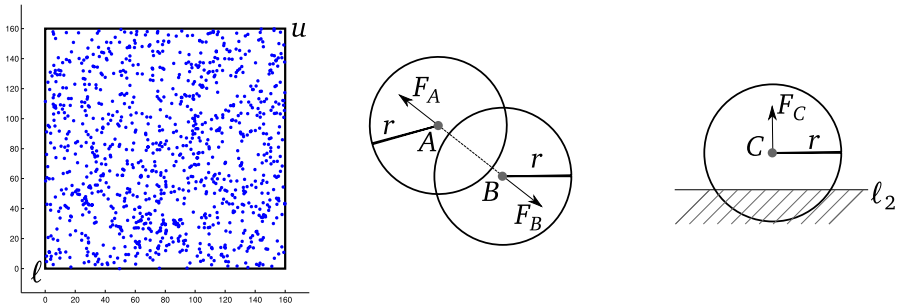
Many-particle system in the plane (Jan Sieber)

This project performs computer experiments with a large number of particles in a box, mimicking a simple gas. We treat the particles as small elastic balls that follow Newton's Laws of motion, travelling freely with until they collide with each other or the wall. Quantities such as density, temperature and pressure at any location can be measured as averages over particles near this location. Figure 1 shows a typical situation and the two types of collisions



(a) (b) (c)

Figure 1: Sketches illustrating box and forces. (a): a typical situation with 1024 particles in a box, (b) illustration of forces F_A and F_B when two particles collide, (c) illustration of force F_C when particle collides with bottom wall

the particles can encounter.

Challenge Implement and test a time-stepping simulation that takes a set of $2 \times N$ initial positions and $2 \times N$ initial velocities and updates them according to Newton's Laws of motion in small time steps h using the Verlet updating formula (an alternativ to Runge-Kutta methods, specifically suited for second-order systems).

Once implemented and tested, use the simulation to perform a sequence of experiments:

1. Observe the law of large numbers by measuring how temperature fluctuations depend on the number of particles.
2. Find the typical distribution of particle speeds in steady state.
3. See how pressure and temperature change as you compress the box by moving a wall inside and compare to theoretical predictions.
4. Check how gravity affects vertical temperature and pressure distribution.

Threshold

- **Correct implementation of the simulation** All forces are taken into account on all particles and the simulation of few particles follows Newton's Laws of motion.
- **Experiments** Simulations with several particle numbers have been demonstrated, measuring temperature, density, pressure.
- **Presentation of results** Time series, and distributions as found ion the computations are presented in appropriate form and discrepancies to expectations from theory (if present) are explained.

Higher

- For a large number N of particles, it becomes computationally inefficient to compute the collision force for all pairs of particles, because there are $N(N-1)/2$ pairs and most of them are far apart. For more efficency (and to enable larger particle numbers) one may put a grid of smaller boxes over the entire box and only checking for collisions between particles that are in the same box or in boxes neighbouring each other. Higher-level marking threshold is achieved if the simulation avoids computations that require computational effort of order N^2.
- Statistical quantities such as temperature, density and pressure have to be averaged over space (and possibly time), and their distributions have to be fitted with appropriate methods to appropriate models.
- For each result a comparison to theoretically expected results is made, with a good explanation of possible differences in observations, all integrated well into the report.

Parameter	Value	Meaning
N	your choice	number of particles (use a power of 4: $N = 4^p$);
K	250	spring constant for particles `ball.spring`;
r	0.2	radius in which particle exerts force `ball.radius`;
h	0.01	time step size in updating formula;
ℓ	`[0;0]`	lower-left corner of box containing particles, first column of input `box` in `SimulationStep`;
u	`[10;10]`$\sqrt{N}$	initial upper-right corner of box (u_2 is changing in part (3)), second column of input `box` in `SimulationStep`;
g	0 or 0.05	gravity (zero for parts (2) and (3), positive for part (4));
x_{ini}	...	initial positions of particles at start of experiment: `x=[l(1)+rand(1,N)*(u(1)-l(1));...` `l(2)+rand(1,N)*(u(2)-l(2))];`
v_{ini}	2.5	approximate initial speed of all particles at start of experiment: `v=2*(rand(2,N)-0.5)*vini;`.

Table 1: Common parameters to be used for experiments

Time stepping and geometry Consider N particles in a rectangle-shaped box, as shown in Figure 1(a). The coordinates of the lower-left corner ℓ of the box are (ℓ_1, ℓ_2), the upper-right corner u has coordinates (u_1, u_2). Each particle's position has a horizontal and a vertical coordinate, stored in an $2 \times N$ array `x`. We also keep track of the velocity of each particle in an array `v` of the same size. All particles have the same mass $m = 1$ and the same spring constant K (which determines how elastic they are). All particles have a radius r, which determines the range at which forces act on the particles (see below).

The simulation runs in small time steps of size h from t_{ini} to t_{end}, say, the times are stored in an array `t=tini:h:tend`. At each time step t_k we calculate the sum of the current forces

on each particle, and store this sum in a $2 \times N$ array `Forces`. Then we update position `x` and velocity `v` of each particle according to the formula

```
xnew=x+h*v+h^2*Forces; % mass is equal 1 !
vnew=(xnew-x)/h;
```

Then `xnew` is the position for the next time t_{k+1}. This is the Verlet updating formula or Störmer method (`vnew` is the approximate velocity of the particle at $t_k + h/2$, not at t_{k+1}).

Forces For each particle one has to sum up the forces below, which act on the particle in each time step (often this sum is zero).[1]

Particle collisions (See Fig. 1(b).) If two particles have distance d less than $2r$, say, particle 1 is at position A and particle B is at position B, then the forces F_A on particle 1, and F_B on particle 2 are

$$F_A = \begin{cases} K(2r-d)\begin{bmatrix}\cos\alpha\\ \sin\alpha\end{bmatrix} & \text{if } 0 < d < 2r, \\ 0 & \text{else,} \end{cases} \qquad \text{and } F_B = -F_A \tag{1}$$

where

$$\begin{aligned} d &= \sqrt{(A_1-B_1)^2+(A_2-B_2)^2} && \text{(distance between points),} \\ \alpha &= \texttt{atan2}(A_2-B_2, A_1-B_1) && \text{(angle between points).} \end{aligned}$$

Since $A-B$ has two components, F_A and F_B also have two components. If $d = 0$, we set $F_A = F_B = 0$. The pre-factor of $K(2r-d)$ in (1) is the amount of force and α is the direction.

Wall collisions (See Fig. 1(c).) If a particle at position C has distance less than r from a wall then a force F_C pushes the particle away from the wall. For the example of the lower wall, shown in Fig. 1(c), the force F_C is pointing upward and the size of this force is $K(r+\ell_2-C_2)$ (ℓ_2 is the vertical cooordinate of the lower wall position, C_2 is the vertical coordinate of the particle). For all 4 walls combined the force on a particle at position C is

$$F_C = K\begin{bmatrix} f_{\text{left}} - f_{\text{right}} \\ f_{\text{bot}} - f_{\text{up}} \end{bmatrix}, \text{ where} \tag{2}$$

$$\begin{aligned} f_{\text{left}} &= \max\{0, r+\ell_1-C_1\}, \quad f_{\text{right}} = \max\{0, r+C_1-u_1\}, \\ f_{\text{bot}} &= \max\{0, r+\ell_2-C_2\}, \quad f_{\text{up}} = \max\{0, r+C_2-u_2\}. \end{aligned} \tag{3}$$

Most of the terms will be zero for most particles. Note that in part (3) the upper wall moves, so u_2 will be different for each time step.

Gravity For part (4) we include an additional gravitational force

$$F_G = \begin{bmatrix} 0 \\ -g \end{bmatrix} \tag{4}$$

[1] If you find typos in the force formulas or something that doesn't make sense, please contact me to clarify.

The number g is different from the usual gravitational acceleration, see Table 1 and part (4) for details.

For each particle i one sums up the forces in (1), (2), (4) and puts this sum (a 2×1 column) into the array `Forces` used in the updating formula. One particle can have a collision with several other particles at the same time (or it could have no collision at some time), it could collide with a wall and a particle simultaneously, or it could collide with two walls at the same time (in a corner).

Statistical quantities

Temperature/kinetic energy is half the square of the average speed of the particles. A particle with velocity $v = (v_1, v_2)$ has speed $\sqrt{v_1^2 + v_2^2}$, so

$$E_{\text{kin}} = (v_1^2 + v_2^2)/2.$$

We can measure the (average) temperature in the entire box or in a part of the box. The temperature may be different at the top and at the bottom, for example, if we have gravity.

Density is the average number d of particles per unit square. So, the average density of the entire box is $d = N/(u_1 - \ell_1)/(u_2 - \ell_2)$. We can measure the (average) density in the entire box or in a part of the box. For example, the density can be different at top and bottom if gravity is non-zero.

Pressure The sum P of the forces in (3) over all particles that hit a piece of wall, divided by the length of the piece of wall. Pressure needs to be averaged over several time steps, because at some time steps no particle hits the wall.

Mean free path For each particle its mean free path is the distance it has travelled during the simulation divided by $1 + n_{\text{cp}} + n_{\text{cw}}$ where n_{cp} is the number of collisions with other particles the particle had, and n_{cw} is the number of collisions of the particle with a wall. The overall mean free path is the average of all mean free paths over all particles.

Tasks

T1 Write a function `SimulationStep` that performs a single time step. Its first line is of the form

```
function [xnew,vnew,...]=SimulationStep(h,x,v,ball,box,g)
```

Inputs

- `h`: time step size (called h above)
- `x`: $2 \times N$ array of current particle positions (first row are horizontal coordinates, second row are vertical coordinates),
- `v`: $2 \times N$ array of current approximate particle velocities (first row are the horizontal coordinates, second row are vertical coordinates),
- `ball`: a structure storing the properties of the ball. Its fields are `ball.radius` (the radius r of all particles) and `ball.spring` (the spring constant K of all particles).

- `box`: 2×2 array containing the corners of the box: $\begin{bmatrix} \ell_1 & u_1 \\ \ell_2 & u_2 \end{bmatrix}$.
- `g`: single number, the amount g of gravity (pointing downward, see equation (4)).

Outputs

- `xnew`: $2 \times N$ array of updated particle positions (first row are horizontal coordinates, second row are vertical coordinates);
- `vnew`: $2 \times N$ array of updated approximate particle velocities (first row are horizontal coordinates, second row are vertical coordinates);
- Note that your function will require more outputs than `xnew` and `vnew`. For example, to measure pressure on some piece of wall, you may have to output all forces from wall collisions (and their locations). The choice of these outputs is left to you. Preferably collect the additional outputs into structures. I will test this function only with the two outputs `xnew` and `vnew`.

A simulation with large N becomes feasible only if one avoids cross-checking each pair of particles for collisions to determine collision forces in (1), because the computational effort will grow proportional to N^2.

A higher threshold is achieved if your function only checks for collisions for nearby particles. This is achieved by putting a grid of smaller boxes over the entire box and only checking for collisions between particles that are in the same box or in boxes neighbouring each other. The additional 20 points get awarded only if your simulation avoids computations that require computational effort of order N^2.

Hints

- When you subdivide the box into a grid of smaller boxes, beware that some particles may be located slightly outside of the big box between ℓ and u (when colliding with the wall with high speed).
- I recommend to split up the function `SimulationStep` into smaller functions that you can test spearately, for example, a function computing the collision force for particle collisions, a function for computing the collision force for wall collisions, a function for finding which particles can collide at the current time step, etc. The choice how to split up is left to you, but you have to upload all functions that you call in your scripts and in `SimulationStep`.

You should submit in addition to your report:

- function file `SimulationStep.m` and all function files called by it electronically;
- script files for each task, and all files called by them electronically;
- printouts of the outputs of the script files, containing plots and the printouts of numbers.

Experimental tasks and table of parameters Once you have tested function `SimulationStep` with a small number of particles to see if it implements the Störmer update rule with the forces as described, conduct the following computational experiments using the parameters in Table 1.

T2 **Fluctuations** Run the simulation for a sequence of increasing numbers of particles N: $N = 4^p$ for $p = 1, 2, 3, \ldots p_{\max}$ ($p_{\max}$ as large as you can afford to wait) and for n_t time steps.

a) Record for each N the temperature T_k ($k = 1, \ldots, n_t$) in the box over time. Find how much the temperature fluctuates over the last n_{rec} time steps by taking the standard deviation $\sigma =$`std(T(end-nrec+1:end))`. You will get a different σ for each N. Store the results σ for all N in a vector of length $p_{\max}$, and plot N versus σ in a `loglog` plot.

b) For the largest N ($N = 4^{p_{\max}}$) plot the distribution of particle speeds s (s is an array of length N: $s =$`sqrt(sum(v.^2,1))`) at the final time n_t. The plotting function `histogram` is useful for plotting distributions.

c) For the largest N print out the overall mean free path. Beware that a collision of a particle with a wall or another particle typically lasts more than a single time step.

One has to discard a number of initial time steps (such that $n_{\text{rec}} < n_t$) because it takes a while until the distribution of speeds and the mean free paths settle from the random initial condition.

Create a script that performs the experiments for part (2) and generates the plots for point 2a and 2b, and the prints out the number for point 2c.

T3 **Compression** Choose a number of particles N as large as you can afford to wait. Run the simulation for some initial time τ_1 until the temperature and the speed distribution has settled (you may use values saved from part (2)). Now shrink the box to half of its original size by gradually varying the upper corner of the box during time period $[\tau_1, \tau_2]$

$$u_2(t) = 10\sqrt{N} - a(t - \tau_1) \quad \text{for } t \in [\tau_1, \tau_2], \text{ where } \tau_2 = \tau_1 + 5\sqrt{N}/a.$$

Then wait for another two time periods (call the next times τ_3 and τ_4, where $\tau_2 < \tau_3 < \tau_4$), and measure the average temperature in the shrunk half-size box and the average pressure on all walls pressure between τ_3 and τ_4.

Repeat the experiment for different wall-moving speeds a and plot how final pressure and temperature in the box depend on a. Compare your results to the fomulas on `wikipedia`'s page for adiabatic compression of an ideal gas (`en.wikipedia.org/wiki/Adiabatic_process#Ideal_gas_.28reversible_process.29`).

The choice of speeds a and switching and stopping times τ_1, τ_2, τ_3 and τ_4 is left to you. Testing low speeds a takes more computation time. The switching and stopping times should be chosen such that the gas has "settled" after τ_1 and again after τ_3 because the motion of the wall between τ_1 and τ_2 may disturb the distribution of particles and velocities in the box.

Create a script that performs the experiments for part (3) and generates the plots for temperature and pressure depending on wall speed a.

T4 **Gravity** Set the gravity constant g to the value in Table 1. Run the simulation for some time τ_1, until the distribution of particles and velocities in the box has settled. Continue the simulation up to some larger time τ_2 and measure how the following quantities depend on the vertical coordinate y during the time $[\tau_1, \tau_2]$:

a) horizontal density $d(y)$: average number of particles per unit length along the x-axis at vertical level y,
b) average temperature $E_{\text{kin}}(y)$ at y,
c) pressure on the vertical walls $P(y)$.

For each of the quantities, guess what type of dependence it has (e.g., independent of y, $1/(a+by)$ or ...), and estimate the coefficients. For example, if you believe that the temperature has the form $1/(a+by^2)$ estimate the numbers a and b.

Create a script that performs the experiments for part (4), generates the plots for density, temperature and pressure depending on the vertical coordinate y, and prints out the function coefficients for density, temperature and pressure.

Hints

- In part (2), I found $n_t = 1000$, $n_{\text{rec}} = 500$ sufficient (with $p_{\max} = 8$, which gives $N = 65\,536$).
- In part (4) one faces an obstacle when trying to measure, for example, the density at vertical coordinate y. For example, say, $N = 1000$. The 1000 particles have 1000 different vertical components y_i ($i = 1, \ldots, 1000$). One needs a method to measure the density at, for example, $y = 10$ (because likely no particle will ever be exactly at $y = 10$). There are various ways to find an approximation for the density at a given y. Temperature $E_{\text{kin}}(y)$ and pressure $P(y)$ have the same problem.