NTNU
Institutt for Fysikk

Exercise 1, 2021
TFY4235/FY8904
Computational Physics

The answer to this exercise should take the form of a report, in pdf format. You will also be required to submit the source code for your simulation program (and it is very nice, but not a requirement, if the code is commented and generally readable). When the home exam starts (at a date to be determined), we will also announce one of the three exercises, which should be handed in along with the exam. Until then you do not need to submit anything from this exercise.

1 Introduction

The system you will study is a two dimensional square box, with boundaries $x = 0$, $x = 1$, $y = 0$, $y = 1$, which is filled with a gas made up of hard disks (also called particles). See Fig. 1 for an illustration. Particle i will be characterised by six variables: position ($\mathbf{x}_i = [x_i, y_i]$), velocity ($\mathbf{v}_i = [v_{xi}, v_{yi}]$), radius ($r_i$) and mass m_i. Particles always move in a straight line with constant velocity, except when they collide either with a wall, or another particle. Collisions are treated as instantaneous.

You will use an approach known as "event driven simulation" (see Ref [1] for a nice detailed introduction). The steps in the simulation will be approximately the following:

- Initialisation:
  - For each particle, calculate if and when it will collide with all other particles, and with the walls, and store all the collision times.
  - Identify the earliest collision.
- Loop:
  - Move all particles forward in time (straight lines, constant velocity) until the earliest collision.
  - For the particle(s) involved in the collision, calculate new velocities.
  - For the particle(s) involved in the collision, calculate if and when they will next collide with all other particles, and with the walls, and store the collision times.
  - Having resolved the collision, identify the new earliest collision, and check if it is still valid (if the particle(s) involved have collided with something since the collision was calculated, it is no longer valid). If the collision is

invalid, discard it and move to the next earliest. Once a valid collision is found, repeat the steps in the loop.

The reason this approach is called "event driven" is that time is always moved forward until the next "event", instead of using a fixed time step. In order to make this procedure efficient, you will use a data structure known as a "priority queue". This will allow you to maintain a list of all future collisions, in such a way that it is easy to identify the next collision.

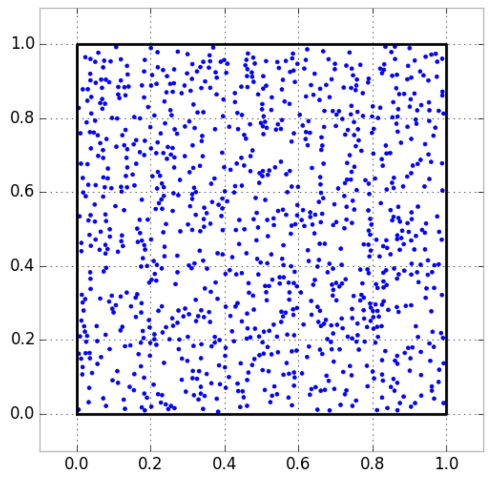


Figure 1: A gas of 1000 hard disks.

2 Collisions

There are two types of collisions: collision between a particle and a wall (see Fig. 2) and collision between two particles (see Fig. 3). We ignore the possibility of simultaneous collisions between more than two objects. The two types of collisions must each be resolved in different ways.

2.1 Collision between a particle and a wall

First, we need to calculate at what time a particle will collide with a wall. This is relatively straightforward. Since particles move in a straight line, except when they

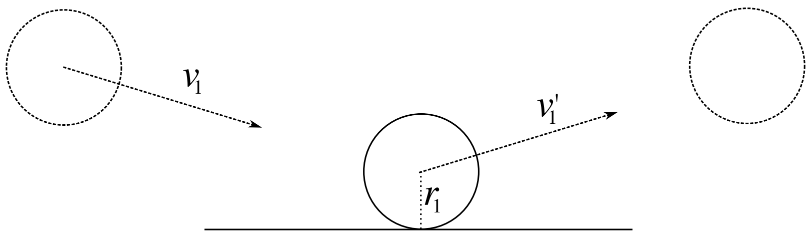


Figure 2: Collision between a particle and a wall.

collide, any particles with velocity different from zero will eventually collide with a wall (unless it collides with something else first). The time is found by treating x and y separately, and in each case calculating the time until the center has travelled a distance equal to the distance from the center to the wall, minus the radius of the particle. The time until particle i collides with a vertical wall is given by

$$\Delta t = \begin{cases} (1 - r_i - x_i)/v_{xi} & \text{if} \quad v_{xi} > 0 \\ (r_i - x_i)/v_{xi} & \text{if} \quad v_{xi} < 0 \\ \infty & \text{if} \quad v_{xi} = 0 \end{cases}, \tag{1}$$

and similarly the time until collision with a horizontal wall is given by

$$\Delta t = \begin{cases} (1 - r_i - y_i)/v_{y_i} & \text{if} \quad v_{y_i} > 0 \\ (r_i - y_i)/v_{y_i} & \text{if} \quad v_{y_i} < 0 \\ \infty & \text{if} \quad v_{y_i} = 0 \end{cases}. \tag{2}$$

When a particle actually collides with a wall, we need to calculate the velocity after the collision. For a particle with velocity $\mathbf{v}_i = [v_{xi}, v_{yi}]$, colliding with a vertical wall, the velocity after the collision, $\mathbf{v}_i'$ is given by:

$$\mathbf{v}_i' = [-\xi v_{xi}, \xi v_{yi}] \tag{3}$$

and similarly for a horizontal wall:

$$\mathbf{v}_i' = [\xi v_{xi}, -\xi v_{yi}] \tag{4}$$

Here, ξ is known as the restitution coefficient, and represents the degree of inelasticity in the collision. If $\xi = 1$, the collision is fully elastic, and no energy is lost.

2.2 Collision between two particles

In order to find the time of a future collision between two particles, i and j, we need to solve an equation to find out if their trajectories will ever bring them into contact.

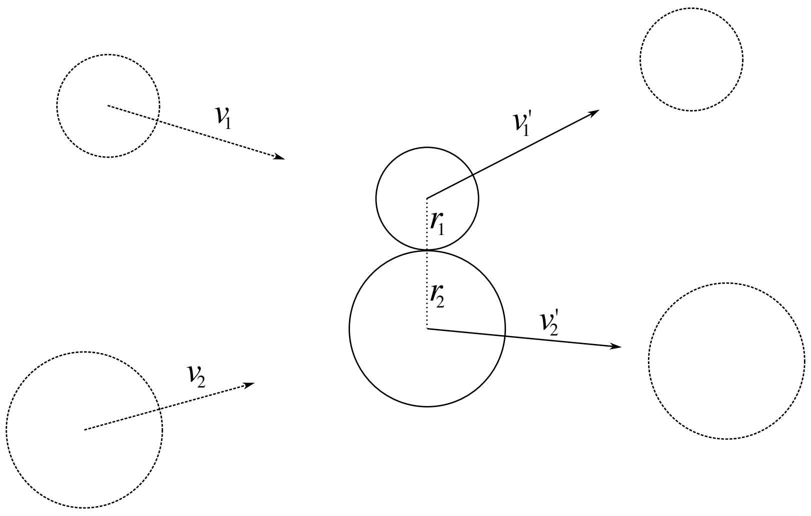


Figure 3: Collision between two particles.

Contact between particles i and j happens when the distance between their centers, R_{ij}, is equal to $r_i + r_j$. Let $\mathbf{x}_i' = [x_i', y_i']$ and $\mathbf{x}_j' = [x_j', y_j']$ be the positions of particles i and j at the time of collision, and let the collision happen at time $t + \Delta t$. Then we have

$$R_{ij}^2 = (x_j' - x_i')^2 + (y_j' - y_i')^2. \tag{5}$$

The positions of the particles at time $t + \Delta t$ are given by

$$\begin{aligned} \mathbf{x}_i' &= \mathbf{x}_i + \mathbf{v}_i \Delta t \\ \mathbf{x}_j' &= \mathbf{x}_j + \mathbf{v}_j \Delta t \end{aligned} \tag{6}$$

where $\mathbf{x}_i$ and $\mathbf{x}_j$ are the positions at time t, and $\mathbf{v}_i$ and $\mathbf{v}_j$ are the velocities at time t (the velocities remain constant until the collision). Inserting Eq. (6) into Eq. (5), we get an equation for Δt which only has a solution if the particles will collide:

$$\Delta t = \begin{cases} \infty & \text{if} \quad \Delta\mathbf{v} \cdot \Delta\mathbf{x} \geq 0 \\ \infty & \text{if} \quad d \leq 0 \\ -\frac{\Delta\mathbf{v} \cdot \Delta\mathbf{x} + \sqrt{d}}{\Delta\mathbf{v} \cdot \Delta\mathbf{v}} & \text{otherwise} \end{cases}, \tag{7}$$

where

$$\Delta\mathbf{x} = [x_j - x_i, y_j - y_i] \tag{8}$$

$$\Delta\mathbf{v} = [v_{xj} - v_{xi}, v_{yj} - v_{yi}] \tag{9}$$

$$d = (\Delta\mathbf{v} \cdot \Delta\mathbf{x})^2 - (\Delta\mathbf{v})^2((\Delta\mathbf{x})^2 - R_{ij}^2). \tag{10}$$

Here, if the time until collision of particles i and j is $\Delta t = \infty$, those particles will never collide (on their current trajectories), and that collision can be disregared.

When particles i and j collide, their new velocities are given by

$$\begin{aligned}\mathbf{v}_i' &= \mathbf{v}_i + \left((1+\xi)\frac{m_j}{m_i+m_j}\frac{\Delta\mathbf{v}'\cdot\Delta\mathbf{x}'}{R_{ij}^2}\right)\Delta\mathbf{x}' \\ \mathbf{v}_j' &= \mathbf{v}_j - \left((1+\xi)\frac{m_i}{m_i+m_j}\frac{\Delta\mathbf{v}'\cdot\Delta\mathbf{x}'}{R_{ij}^2}\right)\Delta\mathbf{x}'\end{aligned}, \tag{11}$$

where $\Delta\mathbf{x}'$ and $\Delta\mathbf{v}'$ are defined as in Eqs. (8) and (9), but evaluated *at the time of the collision*. Again, ξ is the restitution coefficient. If $\xi = 1$, the collision is fully elastic, and no energy is lost. In the opposite case, if $\xi = 0$, the collision is fully inelastic. Note that momentum is still conserved.

2.3 Inelastic collisions

If $\xi < 1$, the collisions are inelastic, and energy is lost at each collision. This can lead to a situation known as "inelastic collapse", where the number of collisions per time goes to infinity. This is obviously a problem for the event driven approach, since all collisions taking place before a time t' must be resolved before the simulation can reach time t'.

One possible way to deal with this problem is known as the TC model[2]. The central idea of the model is the observation that an infinite number of collisions in a finite amount of time is clearly unphysical, since a collision in reality takes a finite amount of time, whereas in the event driven approach described here, collisions are assumed to be instantaneous. The solution is to introduce a "duration of contact", t_c (from which the model takes its name). Any collisions that happen within a time t_c after a previous collision involving the same particle(s) are then assumed to be perfectly elastic ($\xi = 1$). To implement this, simply store the last collision time for each particle. When resolving a collision, check if the any of the involved particle(s) collided at a time less than t_c ago.

3 Data Structures

Depending on your choice of language and personal preference, you may or may not want to go for an object oriented approach. In any case, you will at the very least have to use a priority queue to keep track of future collisions, and potential future collisions will have to certain information associated with them.

3.1 Priority Queue

A priority queue (also sometimes called a "heap queue") is a data structure that implements (at least) the following methods:

- `enqueue`, a method that takes an element and inserts it into the queue (also known as `push` or `heappush`, and probably other names as well),
- `pop`, a method that returns the smallest (or in some implementations, the largest) element in the queue (the method is also known as `heappop`, and probably other names).

There are several different ways to implement a priority queue, but the essential features for efficient operation are that the time to execute the operations `pop` and `enqueue` must be $\mathcal{O}(\log n)$, where n is the number of elements currently on the queue. You are not required to implement the priority queue from scratch yourself. See the Appendix for some more information.

3.2 Collision

The future collisions will have to be stored with the following attributes:

- the time of the collision
- the two colliding entities (one particle and either a wall or another particle)
- the collision count of the involved particle(s) when the collision was calculated.

If the collision count is no longer the same at the time of the collision, the involved particle(s) have collided with something else in the meantime, and the collision is discarded as invalid. The priority queue must be able to order collision events based on their time.

4 Before you begin

Before you begin solving the problems, I strongly recommend that you run through some test cases to make sure you have implemented everything correctly. Some tests are suggested in the Appendix, and you can come up with more tests yourself. The idea is to check a situation where you know what the solution should be, and confirm that your program produces the expected result. There is also some general advice included the Appendix, which you may find useful.

5 Problems

Problem 1

First, we will investigate the speed distribution of a gas of hard disks. Initialise a system with a large number of particles (at least a few thousand or maybe even tens of thousands of particles should be fine, but if you go much beyond that, the simulations may become impractically slow). Give the particles (uniformly distributed) random positions (make sure they do not overlap with each other or the walls), and random velocities given by $\mathbf{v} = [v_0 \cos\theta, v_0 \sin\theta]$, where v_0 is the same for all particles, and θ is a uniformly distributed random angle between 0 and 2π. In this problem, use $\xi = 1$ (i.e, elastic collisions).

Make a histogram showing the initial speed distribution (which should be a delta function, since all particles have the same speed). Let the system run until it has reached equilibrium (number of particle-particle collisions should be much larger than the number of particles), and make a new histogram of the speed distribution. In order to get enough samples for a smooth looking histogram, you will probably need to include the speeds at more than one instant (unless you have a very large number of particles). See the Appendix for some information.

Compare your result to the theoretically expected distribution, of speeds, v, which is the 2D Maxwell-Boltzmann distribution:

$$p(v) = \frac{mv}{kT} \exp\left(-\frac{mv^2}{2kT}\right), \tag{12}$$

where m is the mass of the particles and T is the temperature.

Problem 2

Here, we will simulate a mixture of two gases with different mass per particle. Repeat the setup of Problem 2, but now give half the particles mass $m = m_0$, and the other half $m = 4m_0$. Plot the histogram showing the speed distribution separately for the two particle masses, both initially and after the system has reached equilibrium. Calculate also the average speed and the average kinetic energy separately for the two particle masses. In this problem, use $\xi = 1$.

Problem 3

Repeat the setup of Problem 3. For this problem, you will write output at short intervals (average number of collisions per particle $\ll 1$ during an interval). At each output step, calculate the average kinetic energy over all particles, the average over those particles with mass $m = m_0$, and the average over those with mass $m = 4m_0$. Run the simulation until the average number of particle-particle collisions per particle reaches 10 or 20 or so, then make a plot showing the development of the three averages as a function of time. Repeat this procedure for $\xi = 1$, $\xi = 0.9$ and $\xi = 0.8$. See the comment about the TC model in the Appendix. The point of this task and the previous problem is to demonstrate that with $\xi = 1$, the system reaches equilibrium, and the average kinetic energy (and hence the temperature) of the two gases is the same. When $\xi < 1$, however, the system does not reach equlibrium, and the two gases do not have the same temperature.

Problem 4

For the last problem, you will use your code to study crater formation following a projectile impact. Set up a system with somewhere between 1000 and 10000 particles (depending on what you can simulate within a reasonable amount of time). One particle will be the projectile, and will have larger mass and radius than the others. Give it an initial position of $\mathbf{x}_0 = [0.5, 0.75]$, and a downwards velocity $\mathbf{v}_0 = [0, -v_0]$. The remaining particles will form a "wall". Give them zero initial velocity, and uniformly distributed random positions within the area bounded by $x = 0$, $x = 1$, $y = 0$, $y = 0.5$, i.e. occpying the lower half of the box. Make sure they do not overlap with each other or the walls. These particles should all have equal mass and radius. You will have to decide on the radius based on the number of particles, try adjusting the radius to give a packing fraction of approximately 1/2 (or just a little higher) in the area occupied by the small particles. Describe the procedure you use to set the initial positions.

As starting values, you may set the mass of the projectile to 25 times the mass of the smaller particles, and the radius of the projectile to 5 times the radius of the smaller particles. Set $\xi = 0.5$, and give the projectile a speed of $v_0 = 5$. Run the simulation until only 10% of the initial energy remains (see Fig. 4 for an example). See comments about the TC model in the Appendix.

The point of this task is to investigate the effect of the input parameters on the formation of the crater. Hence, we need a way to quantify the size of the crater. One option is to compare the positions of the particles in the “wall” before and after the simulation, and saying that those particles that have moved are part of the crater. The number of affected particles then gives an indication of crater size. Feel free to implement a different approach if you prefer.

Finally, the task is to do a parameter scan of one of the these input parameters:

- mass of projectile,
- speed of projectile,
- radius of projectile,
- ξ.

Run the simulation for some different values (10 different ones, for example) of the chosen parameter, and plot the size of the crater as a function of the parameter. If you choose to do a scan in ξ, you will probably have to think about a different criterion for stopping the simulation, as it may take a very long time before only 10% of the energy remains. Note that for lower values of ξ, inelastic collapse may be more likely to occur.

This last problem is somewhat more open ended than the others. You will be judged on the quality of you work, not so much on the results, since I’m not quite sure what the results should be.

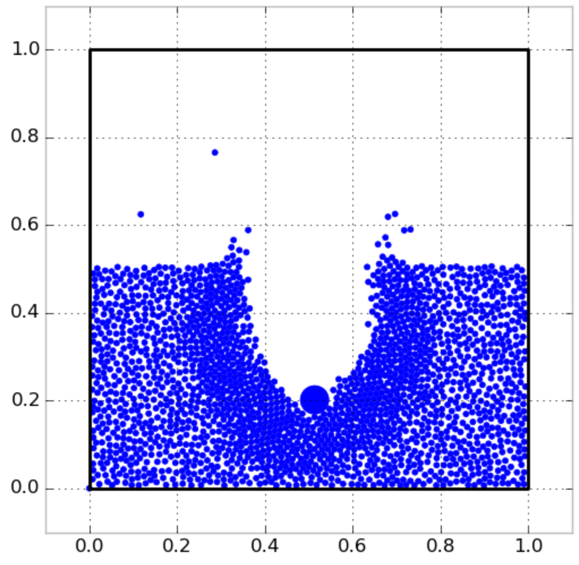


Figure 4: Example of crater formation.

Appendix

A Priority Queue

For a good introduction to priority queues, see Ref. [3]. It has a detailed discussion of the required steps in the implementation, as well as a downloadable implementation in java (the link called MinPQ.java). In python, check out the module `heapq` (part of the standard library). For a list of implementations in different languages, have a look at Ref [4]. I would also recommend a web search for priority queue or heap queue and the name of your favourite language.

B Test Cases

It is highly recommended to run through some test cases, as these can help uncover small errors in the implementation. These are some suggestions, feel free to use others.

B.1 One particle

Confirm that:

- with $\xi = 1$, a particle hitting a wall straight on bounces back at the same speed in the opposite direction,
- with $\xi = 1$, a particle hitting a wall at an angle obeys the law of reflection ($\theta_i = \theta_r$),
- with $\xi = 1$, a particle starting out with velocity $\mathbf{v} = [v_0, v_0]$ should hit all four walls once before coming back to where it started,
- with $\xi = 0$, a particle hitting a wall comes to a complete stop.

B.2 Two particles

Confirm that:

- with $\xi = 1$, two particles with equal mass and the same speed in opposite directions, hitting each other straight on, should bounce back with the same speed in the opposite direction.
- with $\xi = 1$, two particles with equal mass and the same speed in opposite directions, hitting each other with a collision parameter $b = (r_1 + r_2)/\sqrt{2}$ (see

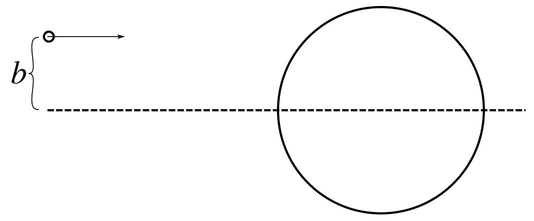
Problem 1), should bounce back with the same speed at right angles to the directions they had before the collision.
- with $\xi = 0$, two particles with equal mass and the same speed in opposite directions, hitting each other straight on, should both come to rest.

B.3 Many particles

Confirm that:

- with $\xi = 1$, the total energy remains exactly constant through the simulation.

B.4 Collision angle



Set up a system with one large and heavy particle ($r_1 = 0.1$, $m_1 = 10^6$) in the middle of the box, and one small and light ($r_2 = 0.001$, $m_2 = 1$) particle hitting it with velocity in the positive x direction only. The distance from the x axis to center of the small particle is called the impact parameter, b. Run the simulation for different values of the impact parameter, and make a graph showing the scattering angle as a function of b/R_{12}, where $R_{12} = r_1 + r_2$. The scattering angle is the angle between the velocity of the small particle before and after the collision. Make sure that the results correspond to what you expect. In this test, use $\xi = 1$.

C Some hints

These are some observations I have made, while solving the problems included in this exercise. Feel free to use or ignore any of the hints as you see fit.

C.1 Output

You may find it useful to plot snapshots of your system, for pedagogical and/or debugging reasons. While it is easy to plot the gas of particles with a typical scatter plot routine, be aware that the size of each marker in a scatter plot is typically not specified in the same units as the positions on the plot. Some experimentation may be required to get decent looking results.

In order to get output at fixed time intervals in an event driven simulation, you may find it convenient to specify a separate output timestep. At every event step, check if the next output is earlier than the next collision. If it is, then move all the particles forward in time until the output step, do the output (plotting or velocities or whatever). Repeat until the next collision happens earlier than the next output, and then resolve the collision as usual. Some approach like this will be required if you want to write output when no collisions have occured.

To turn a series of images into a video, check out the handy command line tool `mencoder`. This is not essential, but it can be useful for debugging.

C.2 Statistics

To get good statistics on the speeds and energies, you will probably need to include speeds from more than one snapshot in time. Two possible approaches can be to run several systems (an ensemble), or to take several snapshots from one system. In the latter case, make sure you let enough time pass between the snapshots (number of collisions per particle between snapshots $\gg 1$). Otherwise, the samples will not be independent.

C.3 Parallelisation

The part of the code that takes up the most of the time is likely to be the function to calculate if and when a particle will collide with any other particles. While it is certainly possible to parallelise this calculation, it could easily be more trouble than it is worth. A simpler approach to parallelisation is to run several copies of your program at the same time (as many as you have cores in your computer). Each copy could then simulate one member of an ensemble, in order to improve statistics, or one set of parameters, as in the crater formation problem.

C.4 TC model

With $\xi < 1$, inelastic collpse may occur, but it is not guaranteed to do so. From my experience, it seems like you may be able to run through all the problems in this exercise with $t_c = 0$, which is equivalent to not using the TC model at all. However, if you run into trouble, and the number of collisions per time suddenly increases dramatically, try figuring out what the average time between collisions is, and set t_c to something smaller than this. You may have to experiment a little. Remember to document what you have done in the report.

References

[1] `http://algs4.cs.princeton.edu/61event/`

[2] Luding, S. and McNamara, S., "How to handle the inelastic collapse of a dissipative hard-sphere gas with the TC model", Granular Matter 1, pp.113-128, 1998.

[3] `http://algs4.cs.princeton.edu/24pq/`

[4] `https://rosettacode.org/wiki/Priority_queue`