Event driven simulation of a 2-dimensional gas

Sondre Duna Lundemo†
Department of Physics, Norwegian University of Science and Technology, Trondheim
Norway
TFY4235 - Computational physics
(*Last updated on February 8, 2021*)

An event driven approach is used to simulate a gas of hard discs in two dimensions. The framework built is used to study statistical properties of the gas at and towards equilibrium. The numerical results are compared with well known results from classical statistical mechanics.

†sondre@duna.no

Contents

1 Introduction and background

An event driven approach is used to simulate a gas of hard discs in two dimensions. The framework built is used to study statistical properties of the gas at and towards equilibrium. The numerical results are compared with well known results from classical statistical mechanics.

The central algorithm for event-driven simulations is the following [1]:

```
Set velocities and positions of all particles in the gas;
Choose a stop criterion ;
for each particle in gas do
    Calculate if and when the particle will collide with all the other particles and
     the walls ;
    Store all the collision times;
end
while not reached stop criterion do
    Identify the earliest collision ;
    if collision is valid then
        Move all particles in straight lines until the earliset collision ;
        for each particle involved in collision do
            Calculate if and when the particle will collide with all the other
             particles and the walls ;
            Store all the collision times;
        end
    else
        discard collsion ;
    end
end
```

Algorithm 1: Event driven simulation of a gas.

In the above algorithm, a collision is *valid* if the particle(s) involved in the collision has *not* collided since the time the collision time was stored.

2 Overview of code

In this section we briefly present how the algorithm is implemented and which considerations are put into the choices of data-structures. The code is written in `python`.

Although the main algorithm presented in the introduction 1 invites to a fully object-oriented approach, I have chosen to restrain myself somewhat in that respect. The main machinery of the code is collected in a class called `Gas`, which in essence is a collection of N `particles` represented their coordinates in phase space, and various methods to manipulate the coordinates of each particle to simulate the gas' time evolution according to the algorithm in section 1. The most prominent caveat preventing me from doing this is that it might affect the speed of the calculations. By choosing not to create separate object representing each particle, I found it very simple to move the particles at each time step and also accessing them by ordinary slicing and indexing of arrays. Although

it is possible to overload operators such that an array of self defined object can be added *like* `numpy`-arrays, I found this to be impractically slow. A quick test of adding the self-made particles compared to simply adding $4 \times N$ arrays establishes this observation quite firmly. The listings below shows the time spent on adding two arrays of 50 000 particles with each of the mentioned methods.

```
>>>%time particles_array_1 += particles_array_2
CPU times: user 443 us, sys: 101 us, total: 544 us
Wall time: 306 us
```

```
>>>%time particles_class_1 += particles_class_2
CPU times: user 34.1 ms, sys: 87 us, total: 34.2 ms
Wall time: 33.2 ms
```

The code itself is well documented and should be easy to undestand on its own. However, I would like to point out some solutions that I found to work particularly well. The part of the code that undoubtedly is the most computationally heavy is the one devoted to calculating if and when the particles will collide with all others. The naive approach would be to iterate over each particle and do the calculation for each of them separately. In `python`, these kind of nested loops will often become impractically slow. I found considerable improvements through vectorising this calculation.

By essentially replacing the piece of calculation in **??** by that in **??** I was able to reduce one of the loops over all of the particles.

```
for j in range(self.N):
  delta_x = self.particles[:2,j] - self.particles[:2,i]
  delta_v = self.particles[2:,j] - self.particles[2:,i]
  R_ij    = self.radii[i] + self.radii[j]
  d       = (delta_x @ delta_v)**2 - (delta_v @ delta_v) * ((delta_x @
    delta_x) - R_ij**2)
```

```
mask = np.arange(self.N-1)
mask[i:] += 1

r_ij = self.radii[mask] + self.radii[i]

delta_x = self.particles[:2,mask] - np.reshape(self.particles[:2,i
    ],(2,1))
delta_v = self.particles[2:,mask] - np.reshape(self.particles[2:,i
    ],(2,1))

vv = np.einsum('ij,ij->j',delta_v,delta_v)
vx = np.einsum('ij,ij->j',delta_v,delta_x)
xx = np.einsum('ij,ij->j',delta_x,delta_x)

d = vx ** 2 - vv * ( xx - r_ij**2 )
```

3 Results

In this section, I will present the results from the simulations in the assigned problems.

3.1 Speed distribution at equilibrium

The first test is devoted to investigating the distribution of the velocities of the particles after sufficiently long time. From statistical mechanics, we know that the velocities of a 2-dimensional gas at equilibrium will distribute according to Maxwell's velocity distribution:

$$p(v) = \frac{mv}{k_B T} \exp\left(-\frac{mv^2}{2k_B T}\right), \tag{1}$$

where k_B is Boltzmann's constant, T the absolute temperature, and m the mass of the particles. It is evident that the particles will have to have the same mass for comparing with the known distribution in equation 1. The case of dissimilar masses is considered in section 3.2 and 3.3.

We initialise an system of particles with velocities $\mathbf{v} = [v_0 \cos\theta, v_0 \sin\theta]$ with $\theta \sim \mathcal{U}[0, 2\pi]$. The initial distribution of the velocities is therefore $C\delta(v - v_0)$. After simulating the system until equilibrium is reached, i.e. the the average number of particle collision is $\gg 1$, the distribution is as shown in figure 1. For this simulation, the stop criterion used was that the average number of collisions surpassed 50. Judging from the plot, it seems to be sufficient for equilibrium to be reached.

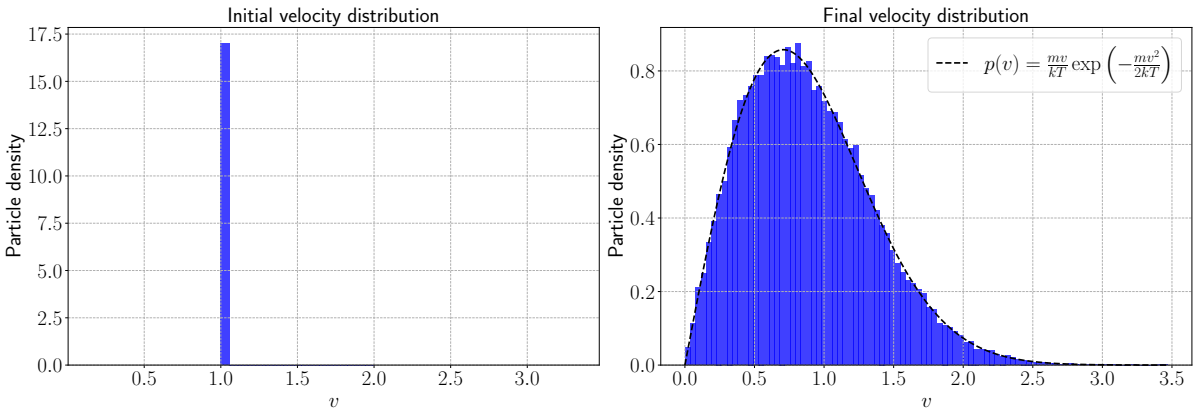

Figure 1: Distribution of velocities in a gas of 50000 particles.

To make the comparison more quantitative, we plot the difference between the Gaussian kernel density estimation of the final distribution and the Maxwell distribution. This is shown in figure 2.

3.2 Mixture of two gases

We repeat the exact same procedure as in section 3.1, except that we make half of the masses 4 times as big as the rest. The velocity distributions after equilibrium is reached is plotted in 3.

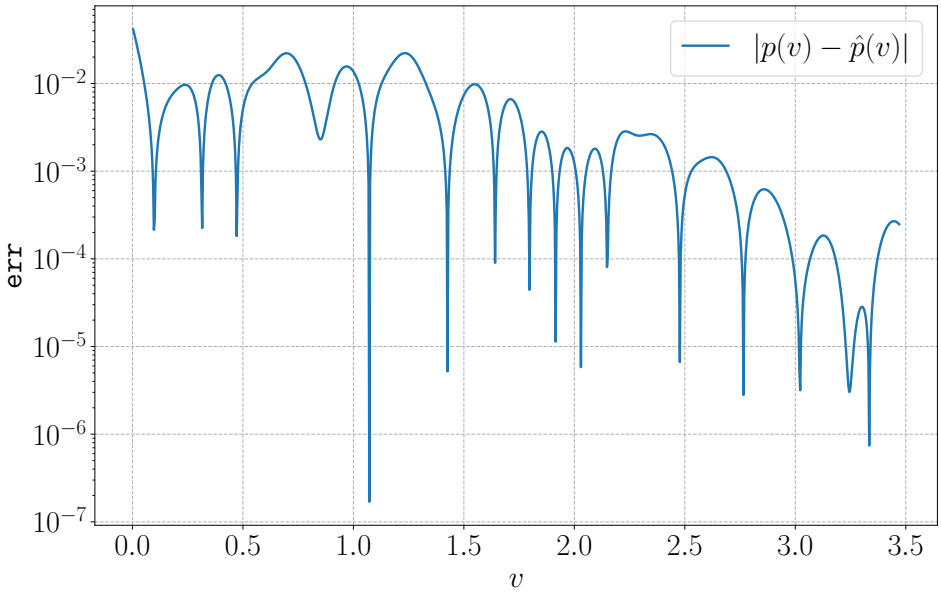

Figure 2: Deviation between Gaussian kernel density estimation of the velocity distribution, $\hat{p}(v)$ and the Maxwell-distribution $p(v)$.

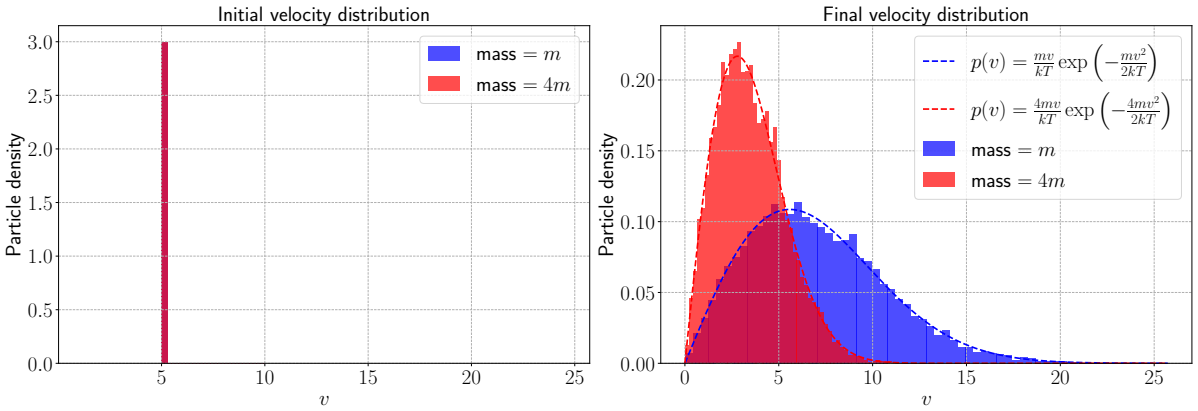

Figure 3: Distribution of velocities in a gas of 50000 particles. The red distributions correspond to the heavy particles, while the blue correspond to the lighter particles.

The average speed and kinetic energy is shown in table 1. These results suggests that although the masses are different, the mixture of the gases eventually reach equilibrium. The fact that the two gases reaches equilibrium is however most easily seen by actually plotting the time evolution of the average kinetic energy. This is shown in section 3.3, where we also simulate situations where this is *not* the case.

Table 1: Average speed and kinetic energy for the light and heavy particles

Mass	**Average speed**	**Average kinetic energy**
m	7.004963	311.848540
$4m$	3.512830	313.151460

3.3 Mixing in the presence of energy dissipation

3.4 Crater formation from projectile impact

References

[1] Aleksandar Donev, Salvatore Torquato, and Frank H. Stillinger. Neighbor list collision-driven molecular dynamics simulation for nonspherical hard particles. i. algorithmic details. *Journal of Computational Physics*, 202(2):737 – 764, 2005.

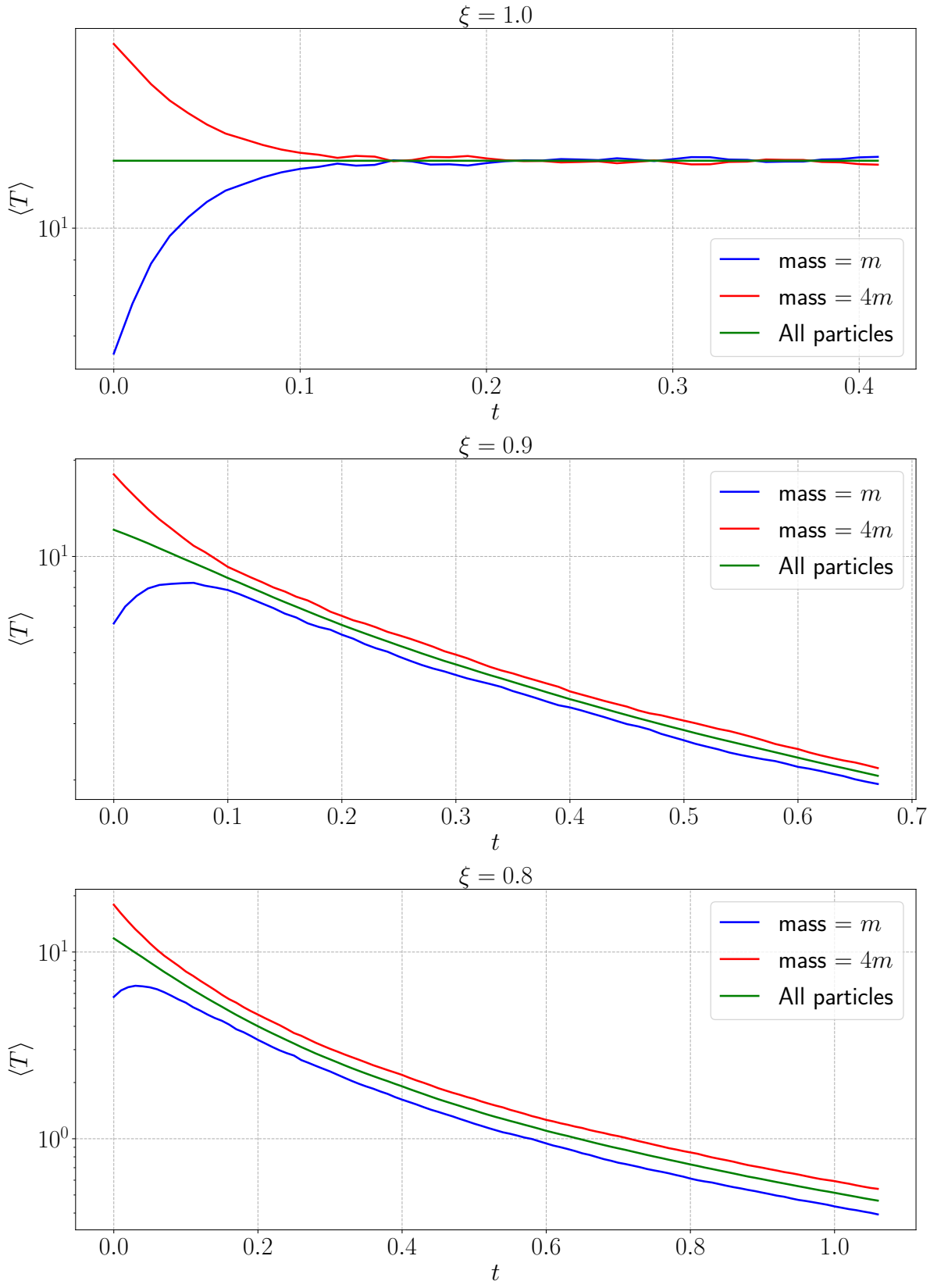

Figure 4: Time evolution of energy averages for restitution parameter $\xi = 1.0, 0.9, 0.8$.