Documentation

James Lamb, Pawel Manikowski

17 December 2018

mpirun -N 7 python cw2_ex2.py terminal output:

```
plmanikowski@nicpon:~/Documents/Spyder/Coursework2$ mpirun -N 5 python cw2_ex2.py
--------------------------Process  4 ------------------------------------
Average energy for    100  particles is equal to:  0.33789347354768723
Average pressure for  100  particles is equal to:  13.870072083491262
-----------------------------------------------------------------------------
--------------------------Process  3 ------------------------------------
Average energy for    100  particles is equal to:  0.3457053347484726
Average pressure for  100  particles is equal to:  13.100521027149385
-----------------------------------------------------------------------------
--------------------------Process  2 ------------------------------------
Average energy for    100  particles is equal to:  0.309210272671474
Average pressure for  100  particles is equal to:  12.24451556097858
-----------------------------------------------------------------------------
--------------------------Process  1 ------------------------------------
Average energy for    100  particles is equal to:  0.33886632521877574
Average pressure for  100  particles is equal to:  14.179851356061016
-----------------------------------------------------------------------------
--------------------------Process  0 ------------------------------------
Average energy for    100  particles is equal to:  0.3527236506930011
Average energy for    5  processes is equal to:  0.33687981137588213
Average pressure for  100  particles is equal to:  14.219381500655983
-----------------------------------------------------------------------------
-----------------------------SUMMARY---------------------------------
Total number of particles (N*size):    500
Number of iterations (Niter):          10000
Time interval (dt):                    0.01
Lenght of the side of the square: (L)  1.5811388300841898
Volume (L^2):                          2.5
-------------------------------------------------------------------
Total average pressure (P)             67.61434152833623
Total average energy: (Ntot*K):        168.43990568794106
P*V:                                   169.03585382084057
-------------------------------------------------------------------
```

(a) Describe the class `Particles`. How is initialised? What is the role of each method? (3 marks)

Initialisation:

```
class Particle(object):
        pos = np.array([0,0])
        vel = np.array([0,0])
        mass=1
        radius = 0.01

        def __init__(self, pos, vel):
            self.pos = pos
            self.vel = vel
```

The class `Particles` is initialised with position vector `pos` and velocity vector `vel` of the particle. Both vectors have horizontal and vertical components which are stored in numpy array: `np.array([0,0])`. Each particle has also mass equal one and radius equal 0.01.

Methods:

```
def energy(self):
    self.energy = 0.5*self.mass*np.sum(self.vel**2)
    return(self.energy)
```

The `energy` method defines a new property of the instance of the class, `self.energy`. Note: We modified the method so that it returns the kinetic energy of the particle. The following formula has been implemented:

$$E_k = \frac{mv^2}{2}$$

```
def evolve(self,dt):
    self.pos = self.pos + self.vel * dt
    self.vel = self.vel
```

The `evolve` method redefines the position of the instance of the particle based on change of time (`dt`):

$$\underline{r}(x, y) = \underline{r}_0 + \underline{v}t$$

```
def flip_vx(self):
    self.vel[0] = -self.vel[0]

def flip_vy(self):
    self.vel[1] = -self.vel[1]
```

The `flip_vx` method changes the direction of x component of the velocity when v_x is equal to L or 0. Where L is the size of the box. Similiarly, the `flip_vy` method changes the direction of y component of the velocity when v_y is equal to L or 0.

```
def info(self):
            print("position:",self.pos,"  velocity: ",self.vel)
```

The `info` method prints the position and the velocity of the particle.

(b) What are the distributions of the initial positions and velocities? (3 marks)

The distributions of the initial positions and velocities are random such that:

Position vector

The initial position of the particle is obtained using the `random_position` function:

```
def random_position(L):
        return L * np.random.rand(2)
```

It depends on the variable L which is the length of the square box in each direction. The values of the x and y components are a randomly generated number between 0 and 1 multiplied by L and are stored in a numpy array. Note: This is a uniform distribution, whose scalar L sets its minimum and maximum to be defined by the area of the box.

Velocity vector

```
rand_v =  np.random.rand(2)
```

Each component of the velocity has a uniform distribution in the interval 0 and 1 and is stored in the numpy array.

(c) Parallelise the code so that each process simulates the motion of N particles. The part of the code generating the plot and the animation should only be run by process 0. (5 marks)

Initialisation:

```
from mpi4py import MPI

comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()
```

Plot and animation on process 0:

```
if rank == 0:
    plot_flag = True
else:
    plot_flag = False
```

(d) Modify the code to compute numerically the averaged kinetic energy K per particle.

Defining the list of particles energy:

```
energy_particles = []
```

Values of energy of each particle are stored in the list:

```
energy_particles.append((list_particles[i].energy()))
```

Average energy of N particles for each process:

```
average_energy = sum(energy_particles)/N
```

Total energy in process 0:

```
sum_of_average_energy = comm.reduce(average_energy, op=MPI.SUM)
```

Calculating the total average energy of all particles:

```
if rank == 0:
    K = sum_of_average_energy/size
    print("Average energy for   ",size," processes is equal to: ",K)
```

We can see, from the terminal output, that average kinetic energy per particle is equal to:

```
Average energy for 5 processes is equal to: 0.33687981137588213
```

(e) Compute the average pressure during an interval dt over N_{iter} steps for each process and globaly.

The force applied by one particle on the wall located at $x = L$ during an interval dt reads:

$$f_x = \frac{2mv_x}{dt}$$

where v_x is the velocity of the particle in the x-direction. Therefore, when position of the particle in x-direction is equal L:

```
fx.append(2*m*p.vel[0]/dt)
```

The force F_x exerted by N particles colliding with the boundary in a dt interval is the sum of the individual forces:

```
forces_fx = sum(fx)/Niter
```

Average pressure during an interval dt over N_{iter} steps for each process:

```
average_pres = forces_fx/L
print("Average pressure for ",N," particles is equal to: ", average_pres)
```

Average pressure globaly:

```
P = comm.reduce(average_pres, op = MPI.SUM)
```

Average pressure for each process is equal to:

```
Average pressure for 100 particles is equal to: 14.219381500655983
```

Average global pressure for five processes is equal to:

```
Total average pressure (P) 67.61434152833623
```

(f) Plot PV and the averaged total kinetic energy $N_{\text{tot}}K$ for a few volumes.

When we divide PV by $N_{\text{tot}}K$, we get:

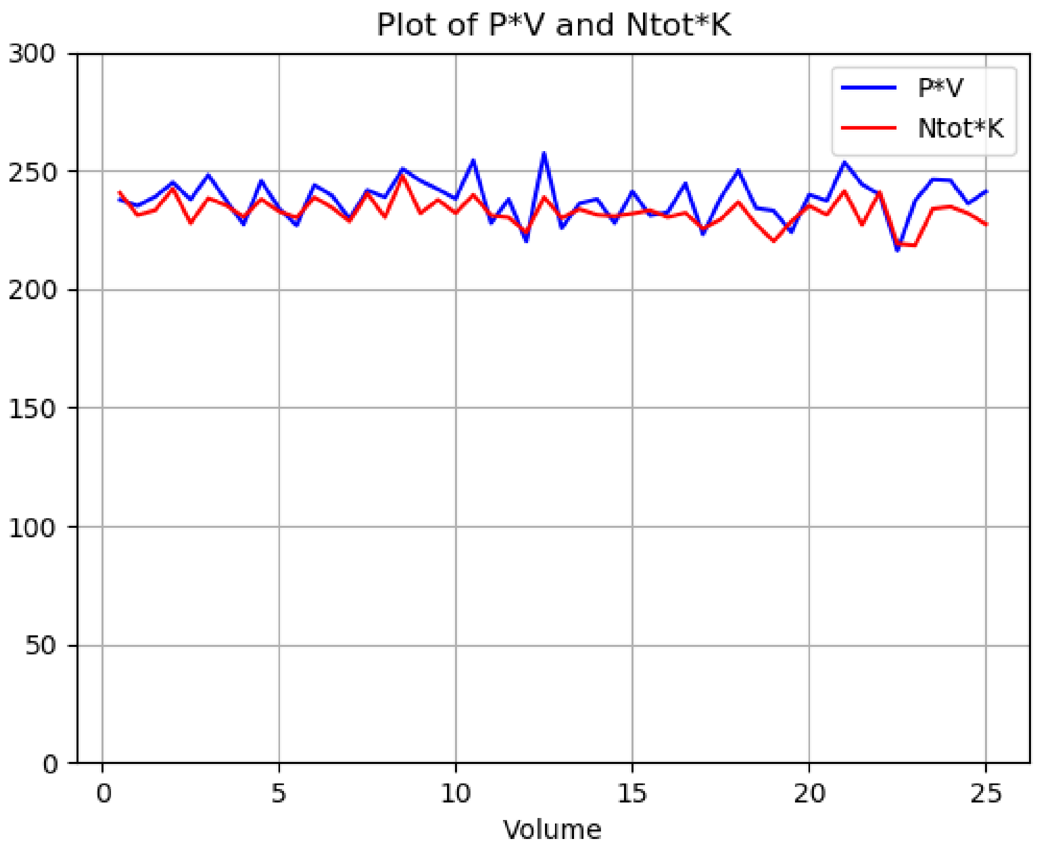
$$\text{Ratio} = \frac{PV}{N_{\text{tot}}K} = \frac{169.03585382084057}{168.43990568794106} = 1.003538$$

Therefore, we obtained a very good approximation. If we want to get a better results we need to increase the amount of particles and number of interations, change time interval and reduce the volume. The above result confirms that:

$$PV = N_{\text{tot}}K$$

for a large number of particles.

In order to plot PV and the averaged total kinetic energy $N_{\text{tot}}K$ we have to run `cw2_ex2_partf.py`, which is a modification of `cw2_ex2.py`.



We can see from the plot that for a given values of Volume V

$$PV \approx N_{\text{tot}}K$$

(g) Suppose now that particles interaction are switched on, briefly discuss how the code would have to be modified, and what would be the main challenges.

If we “switched on” the collisions of the particles the current parallelisation would not work. If, for example, particles from process one and particle from process two collide then it would not be taken into consideration until all positions of all particles are gathered to process zero. The gathering process would have to take place after each iteration. In case of colision, the velocity of two coliding particles would have to be adjusted. This would make the parallelisation very inefficient.