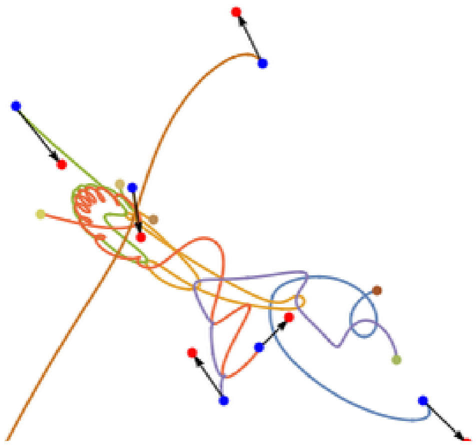
Homework: Parallelism - MPI

2D N-Bodies problem

N-Bodies problem: Evolution of a set of bodies that interact with each other. (wikipedia link)

Example in astrophysics: The motion of stars taking into account the forces of gravitation.



For each pair of stars, each star applies to the other a force f where G is a constant, m1, m2 are the respective masses of the two stars, and x1, x2 their 2 position vectors.

$$\vec{f} = \frac{G.m_1.m_2.(\overrightarrow{x_1} - \overrightarrow{x_2})}{|\overrightarrow{x_1} - \overrightarrow{x_2}|^3}$$

At each time step, we calculate the forces that apply to each star, then we update the data concerning it (position, velocity and acceleration).

The basic solution consists at each iteration and for each star in :

1. calculating the forces exerted by every other star
2. computing the resultant of forces by summing up all these forces
3. updating the position, velocity and acceleration of the star based on the resultant of forces

The sequential algorithm for a 2 dimension problem is as follows:

```
for t in 0, NB_STEPS
      force = np.zeros((N, 2)) # force[i] is a 2D vector
      for i in 0,N
            for j in 0, N
                  force[i] = force[i] + interaction(data[i], data[j])

      for i in 0, N
            data[i] = update(data[i], force[i])
```

where **data[i]** is a data structure (a vector of 6 floats) that includes the position and velocity of star **i**, **N** is the number of stars, **interaction(data[i],data[j])** is a function that returns the force applied by star **j** to star **i** (this function reads **data[i]** and **data[j]** but does not modify them), and **update(data[i],force[i])** is a function that computes and returns the new position, velocity and acceleration of star **i** based on the force it experiences. **interaction(data[i], data[i])** always returns a null vector.
It is assumed that only the process of rank **0** knows **data** and that the number of stars is a multiple of the number of processes. We will simulate the reading of data by process of rank 0 by using the function **init_world(N)**. A set of useful functions and variables is provided in **`n-bodies-base.py`**, see description below in Appendix.

All the programs must have the following arguments:

```
python3 n-bodies-base.py 12 1000
mpirun -n 3 python3 n-bodies-par.py 12 1000
```

where the first parameter is the number of bodies (**N** in the text), and the second one in the number of iterations. If you run the base file provided, it will display the initial position, wait two seconds and then print the signature. If with the same input the signatures of your codes are different, it means there is a difference in the resulting position of the stars, and thus almost certainly an error in the code. If you add a **-nodisplay** argument, the function to display the stars should not be used.

Upload on moodle a zip or a tgz file containing the following files with exactly the same name. An automatic tool will be used during grading. You can test a simplified version called **n-bodies-test.sh**
Question 1: Implement a sequential version of the previous algorithm and call the resulting code **`n-bodies-seq.py`** that will print on the last line the value of the signature alone (for automatic verification).

Question 2: Write a parallel MPI version of this algorithm in a file called **`n-bodies-par.py`**.

Question 3: The sequential algorithm does not take into account the fact that the forces are symmetrical: if **f(i,j)** represents the force applied by star **i** to star **j**, we have **f(i,j) = -f(j,i)**. It is therefore not necessary to calculate the intensity of these two forces separately.

To take this observation into account, one could modify the sequential algorithm for calculating the forces:

```
for t in 0 , NB_STEPS
    force = [[0,0] for _ in range(N)]
    for i in 0 , N
        for j in 0 , i
            force_j_on_i = interaction(data[i], data[j])
            force[i] = force[i] + force_j_on_i
            force[j] = force[j] - force_j_on_i
    for i in 0 , N
        data[i] = update(data[i], force[i])
```

Write a parallel MPI version of this code in a file called **`n-bodies-opt.py`**

Question 4: Run the code of question 2 and of question 3 with 1;2;3;4 processes. For each run use 12 bodies and 1000 iterations and no display of the simulation. Write in a file called **`timing.csv`** the duration of the computation part (excluding **init_world**). The content should resemble the following output. The first column is the number of processes; the second is the time for the sequential version; the third is the time for the result of question 2; the last is the time for the result of question 3.

```
1;5;6;7
2;5;3;2
3;5;9;11
4;5;3;10
```

Appendix (functions provided in n-bodies-base)

- Do not modify these functions
    - `init_world(n)`: this function returns a list containing all n bodies (n is passed as parameter). Each body has several attributes: position, speed, weight. However you should not have to manipulate them directly.
    - `update(d,f)` : this function takes a body d and a force f as parameters and returns the body d whose position and speed have been modified by the force f
    - `interaction(body1, body2)`: this function returns a list with two elements representing the force on each of the X and Y axes, resulting from the interaction of `body2` on `body1`
    - `display(m, l)` : this function allows to display the parameters of the bodies contained in the list l by prefixing this display with the message m
    - `displayPlot(data)` : display the bodies which are in data on the graphical window
    - `signature(data)` : compute a value based on the characteristics of the bodies, which represents the signature of this world. Used after the evolution iterations, to check that the evolution of the world calculated in sequential and in parallel is the same.
    - the other utility functions and the class are not intended to be used directly.