Fly like a bird

- **Topic:** agent-based models, emergence, flocking, input files
- **Task:**
  1. Implement calculation of the cohesion force.
  2. Run the simulation and check that you get reasonable flocking behaviour.
  3. Try changing some simulation parameters and see how that changes the motion.
  4. Also add a few predators in the simulation.
- **Template:** flocking.py
- **Data:**
  - flock_system.txt
  - flock_dynamics.txt
- **Further reading:**
  - https://en.wikipedia.org/wiki/Agent-based_model
  - https://en.wikibooks.org/wiki/Boids
  - https://www.red3d.com/cwr/boids/

flocking.py

`class flocking.`**`Animal`**`(position, velocity, top_speed, mass=1.0)` [source]

A class representing an animal.

All animals are treated as point-like agents in this model.

Parameters:
- **position** (array) – coordinates $[x, y, z]$
- **velocity** (array) – velocity components $[v_x, v_y, v_z]$
- **top_speed** (float) – the maximum speed allowed for this animal
- **mass** (float) – mass m

`accelerate``(dt)` [source]

Set a new velocity as

$$\vec{v}(t + \Delta t) = \vec{v}(t) + \frac{1}{2m}\vec{F}\Delta t$$

If the speed would exceed the maximum speed of the animal, it is scaled down to the max value. Similarly, the animal must move at least with a speed that is half of the maximum speed.

Parameters: **dt** (float) – time step Δt

`move``(dt)` [source]

Move the animal.

Parameters: **dt** (float – time step Δt

`save_position``()` [source]

Save the current position.

Note: in a real large-scale simulation one would never save trajectories in memory. Instead, these would be written to a file for later analysis.

`class flocking.`**`PeriodicBox`**`(lattice)` [source]

Class representing a simulation box with periodic boundaries.

The box is orthogonal, i.e., a rectangular volume. As such, it is specified by the lengths of its edges (lattice constants).

Parameters: **lattice** (array) – lattice constants

`distance_squared``(position1, position2)` [source]

Calculates and returns the square of the distance between two points,

$$r_{ij}^2 = (x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2.$$

In a periodic system, each boid has an infinite number of periodic copies. Therefore the distance between two points is not unique. The function returns the shortest such distance, that is, the distance between the the periodic copies which are closest on each other.

Parameters:
- **position1** (array) – the first point
- **position2** (array) – the second point

Returns: the squared distance r_{ij}^2

Return type: float

`shift_inside_box``(position)` [source]

If the given position (3-vector) is outside the box, it is shifted by multiple of lattice vectors until the new position is inside the box. That is, the function transforms the position vector to an equivalen position inside the box.

Parameters: **position** (array) - the position to be shifted

`vector``(position1, position2)` [source]

Returns the vector pointing from position1 to position2,

$$\vec{r}_{ij} = \vec{r}_j - \vec{r}_i$$

In a periodic system, each boid has an infinite number of periodic copies. Therefore the displacement between two points is not unique. The function returns the shortest such displacement vector.

Parameters:
- **position1** (array) – the first point
- **position2** (array) – the second point

Returns: components of $\vec{r}_{ij} = [x_{ij}, y_{ij}, z_{ij}]$

Return type: array

`class flocking.`**`Predator`**`(position, velocity, prey, top_speed, mass=1.0)` [source]

A point-like model for a predator Animal.

Likes to harass Prey animals.

Parameters:
- **position** (array) – coordinates $[x, y, z]$
- **velocity** (array) – velocity components $[v_x, v_y, v_z]$
- **top_speed** (float) – the maximum speed allowed for this animal
- **mass** (float) – mass m

`add_chase_force``(box, parameters)` [source]

Adds a force driving the Predator towards its target Prey.

Parameters:
- **box** (flocking.PeriodicBox) – supercell
- **parameters** (list) – force parameters

`add_separation_force``(box, parameters)` [source]

Adds a force to make predators avoid collisions.

This is a short ranged repulsive force applied only if the animals are very close to each other.

Parameters:
- **box** (flocking.PeriodicBox) – supercell
- **parameters** (list) – force parameters

`calculate_forces``(box, force_parameters)` [source]

Calculates all forces acting on this animal.

The force sum is saved in Predator.force.

Parameters:
- **box** (flocking.PeriodicBox) – supercell
- **force_parameters** (list) – force parameters

`pick_target``(prey, box, radius)` [source]

Randomly picks one Prey within the given radius as the new target to chase.

If there are no prey within range, nothing happens.

Parameters:
- **prey** (list) – list of Prey objects
- **box** (flocking.PeriodicBox) – supercell
- **radius** (float) - maximum range for choosing a target

`class flocking.`**`Prey`**`(position, velocity, top_speed, mass=1.0)` [source]

A point-like model for a flock-forming Animal.

Can be harrassed by a Predator.

Parameters:
- **position** (array) – coordinates $[x, y, z]$
- **velocity** (array) – velocity components $[v_x, v_y, v_z]$
- **top_speed** (float) – the maximum speed allowed for this animal
- **mass** (float) – mass m

`add_alignment_force``(box, parameters)` [source]

Adds a force to make Prey move in the same direction

Alignment force is calculated by first determining the average velocity vector of the neighbors. The force is then proportional to the difference between the current velocity of the animal and the calculated average.

Parameters:
- **box** (flocking.PeriodicBox) – supercell
- **parameters** (list) – force parameters

`add_avoidance_force``(predators, box, parameters)` [source]

Adds a force to make Prey flee from Predator animals.

This force is calculated similarly to `Prey.add_separation_force()`, but it is triggered by predators instead of other prey. Typically, this force should also be stronger and have a longer range than the separation force.

Parameters:
- **box** (flocking.PeriodicBox) – supercell
- **parameters** (list) – force parameters

`add_cohesion_force``(box, parameters)` [source]

Adds a force to make Prey seek each other.

Cohesion force is a force pointing to the center (average coordinate) of all the neighbors of the animal.

Note
This function is incomplete!

Parameters:
- **box** (flocking.PeriodicBox) – supercell
- **parameters** (list) – force parameters

`add_separation_force``(box, parameters)` [source]

Adds a force to make prey avoid collisions.

This is a short ranged repulsive force applied only if the animals are very close to each other.

Parameters:
- **box** (flocking.PeriodicBox) – supercell
- **parameters** (list) – force parameters

`calculate_forces``(predators, box, force_parameters)` [source]

Calculates all forces acting on this animal.

The force sum is saved in Prey.force.

Parameters:
- **predators** (list – list of Predator objects
- **box** (flocking.PeriodicBox) – supercell
- **force_parameters** (list) – force parameters

`flocking.`**`animate`**`(prey, predators, box, multiply=(3, 3))` [source]

Animates the simulation.

Parameters:
- **prey** (list – list of Prey objects
- **predators** (list – list of Predator objects
- **box** (flocking.PeriodicBox) – supercell
- **multiply** (array) – number of periodic images to draw in x and y directions

`flocking.`**`calculate_forces`**`(prey, predators, box, force_parameters)` [source]

Calculates forces on all animals.

Parameters:
- **prey** (list – list of Prey objects
- **predators** (list – list of Predator objects
- **box** (flocking.PeriodicBox) – supercell
- **force_parameters** (list) – force parameters

`flocking.`**`create_random_flock`**`(prey_parameters, predator_parameters, box)` [source]

Creates a flock of Prey and possible also Predator animals.

The animals are placed randomly and they are given random velocities.

Parameters:
- **prey_parameters** (list) – amount and top speed of prey
- **predator_parameters** (list) – amount and top speed of predators
- **box** (flocking.PeriodicBox) – supercell

`flocking.`**`draw`**`(frame, xtraj, ytraj, ztraj, bounds)` [source]

Draws a representation of the system as a scatter plot.

Used for animation.

Parameters:
- **frame** (int) – index of the frame to be drawn
- **xtraj** (array) – x-coordinates of all animals at different animation frames
- **ytraj** (array) – y-coordinates at all animals at different animation frames
- **ztraj** (array) – z-coordinates at all animals at different animation frames
- **bounds** (array) – list of lower and upper bounds for the plot as [xmin, xmax], [ymin, ymax]

`flocking.`**`find_info`**`(lines, tag)` [source]

Searches for the information wrapped in the given tag among the given lines of text.

If tag is, e.g., "foo", the function searches for the start tag <foo> and the end tag </foo> and returns the lines of information between them.

The function only finds the first instance of the given tag. However, in order to catch multiple instances of the tag, the function also returns all the information in lines following the end tag.

For instance, if lines contains the strings:

```
aa
<foo>
bb
cc
</foo>
dd
ee
ff
```

the function will return two lists: ["bb", "cc"], ["dd", "ee", "ff"].

Parameters:
- **lines** (list) - the information as a list of strings
- **tag** (str) – the tag to search

Returns: the lines between start and end tags, the lines following the end tag

Return type: list, list

`flocking.`**`main`**`(dynamic_file='flock_dynamics.txt', system_file='flock_system.txt')` [source]

The main program.

The program reads info from files, runs the simulation, and plots the trajectory.

Parameters:
- **dynamic_file** (str) - file with force parameters
- **system_file** (str) - file with simulation parameters

`flocking.`**`parse_line`**`(line)` [source]

Separates tag and info on a line of text.

The function also removes extra whitespace and comments separated with #.

For instance if line is " x : 1.23 # the x coordinate", the function returns ("x", "1.23").

Parameters: line (str) - a string of information

Returns: tag, info

Return type: str, str

`flocking.`**`print_progress`**`(step, total)` [source]

Prints a progress bar.

Parameters:
- **step** (int) – progress counter
- **total** (int) – counter at completion

`flocking.`**`read_animal_info`**`(lines, default=2)` [source]

Reads parameters from given lines.

Parameters:
- **lines** (list) - information as a list of strings
- **default** (float) - the default lattice parameter in all directions

Returns: parameters

Return type: list

`flocking.`**`read_box_info`**`(lines, default=2)` [source]

Reads parameters from given lines.

Parameters:
- **lines** (list) - information as a list of strings
- **default** (float) - the default lattice parameter in all directions

Returns: parameters

Return type: list

`flocking.`**`read_dynamics_file`**`(filename)` [source]

Reads the given file for force parameters.

Parameters: **filename** (str) - the file to read

Returns: cohesion, alignment, separation, avoidance, chase parameters

Return type: lists

`flocking.`**`read_force_info`**`(lines, default=1.0)` [source]

Reads parameters from given lines.

Parameters:
- **lines** (list) - information as a list of strings
- **default** (float) - the default lattice parameter in all directions

Returns: parameters

Return type: list

`flocking.`**`read_system_file`**`(filename)` [source]

Reads the given file for system parameters.

Parameters: **filename** (str) - the file to read

Returns: prey, predator, box, time parameters

Return type: lists

`flocking.`**`read_time_info`**`(lines, default=0.2)` [source]

Reads parameters from given lines.

Parameters:
- **lines** (list) - information as a list of strings
- **default** (float) - the default lattice parameter in all directions

Returns: parameters

Return type: list

`flocking.`**`update_neighbors`**`(animals, box, radius)` [source]

Finds for each animal the other animals of the same type within the given radius.

The animals can be both Prey or Predator.

The neighbors are saved in the animal object as a list of animal objects.

Parameters:
- **animals** (list - list of animals
- **box** (flocking.PeriodicBox) - supercell
- **radius** (float - maximum distance to neighbors

`flocking.`**`update_positions_no_force`**`(prey, predators, dt)` [source]

Update the positions of all animals.

$$\vec{r}(t + \Delta t) = \vec{r}(t) + \vec{v}\Delta t$$

Parameters:
- **prey** (list - a list of Prey objects
- **predators** (list - a list of Predator objects
- **dt** (float – time step Δt

`flocking.`**`update_targets`**`(prey, predators, box, radius)` [source]

Possible updates the target for every predator.

For each Predator, there is a 10 % chance it seeks a new target Prey via `Predator.pick_target()`.

Parameters:
- **prey** (list – list of Prey objects
- **predators** (list – list of Predator objects
- **box** (flocking.PeriodicBox) – supercell
- **radius** (float) - maximum distance to new targets

`flocking.`**`update_velocities`**`(prey, predators, dt)` [source]

Update the positions of all animals according to

$$\vec{v}(t + \Delta t) = \vec{v}(t) + \frac{1}{m}\vec{F}\Delta t$$

Parameters:
- **prey** (list - a list of Prey objects
- **predators** (list - a list of Predator objects
- **force** (array) - array of forces on all bodies
- **dt** (float – time step Δt

`flocking.`**`velocity_verlet`**`(prey, predators, box, force_parameters, dt, time, trajectory_dt=1.0)` [source]

Verlet algorithm for integrating the equations of motion, i.e., advancing time.

There are a few ways to implement Verlet. The leapfrog version works as follows. First, forces are calculated for all animals and velocities are updated for half a time step, $\vec{v}(t + \frac{1}{2}\Delta t) = \vec{v}(t) + \frac{1}{2m}\vec{F}\Delta t$. Then, these steps are repeated:

- Positions are updated by a full time step using velocities but not forces,

$$\vec{r}(t + \Delta t) = \vec{r}(t) + \vec{v}(t + \frac{1}{2}\Delta t)\Delta t.$$

- Forces are calculated at the new positions, $\vec{F}(t + \Delta t)$.
- Velocities are updated by a full time step using the forces

$$\vec{v}(t + \frac{3}{2}\Delta t) = \vec{v}(t + \frac{1}{2}\Delta t) + \frac{1}{m}\vec{F}(t + \Delta t)\Delta t$$

These operations are done using the methods `calculate_forces()`, `update_velocities()` and `update_positions_no_force()`.

Because velocities were updated by half a time step in the beginning of the simulation, positions and velocities are always offset by half a timestep. You always use the one that has advanced further to update the other and this results in a stable algorithm.

Parameters:
- **prey** (list - a list of Prey objects
- **predators** (list - a list of Predator objects
- **box** (flocking.PeriodicBox) - supercell
- **force_parameters** (list) – force parameters
- **dt** (float) – time step Δt
- **time** (float) – the total system time to be simulated
- **trajectory_dt** (float - the positions of prey are saved at these these time intervals – does not affect the dynamics in any way

Table of Contents
Fly like a bird
flocking.py
Previous topic
The heat is on
Next topic
Homework: It's a gas at the bridge
This Page
Show Source
Quick search
Go