Particle Methods
lecture 4

Igor V. Pivkin

USI Lugano

Spring 2023

Agent-based Ecosystem

Lets assume that we have a habitat which is populated by 2 species of animals.
One species of predators, lets call them wolves, and one species of prey, lets call them rabbits.
The animals can move around the habitat and perform some actions, i.e. eat, replicate and die.
We will model this system using ABM.

The model

In the model, assume our habitat is a 2D domain, a square, L units in size, periodic in all directions.
Each animal is represented by an agent. We have to types of agents: rabbits and wolves.
The simulations start with agents randomly distributed in the domain: N_r rabbits and N_w wolves.
Rabbits behave according to the following rules:

- They perform a random walk in the domain
- They replicate with probability p_r^r
- They live short lives and die after t_d^r timesteps

Wolves behave according to the following rules:

- They perform a random walk in the domain
- They eat a rabbit with probability p_e^w every time they get within distance r_c of it
- They replicate with probability p_r^w every time they eat a rabbit
- They live long but die from hunger if they don't eat for t_d^w timesteps

Implementation

Usual steps in the implementation ..

Initialize system: distribute agents
Integrate in time: for each timestep = first .. last:

- For each agent:
  - update agent positions
  - compute agent interactions (eat, replicate, die)

Analyze results

Implementation features we will focus on this week:

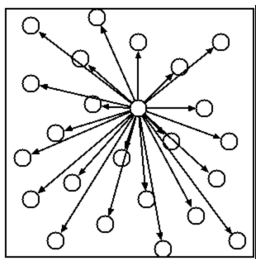
- How to compute agent pairwise interactions efficiently?
- How to treat periodic boundary conditions?

Efficient Pairwise Interaction Calculations

In order to compute pairwise interactions of agents (e.g. wolf eats rabbit), each agent needs to interact with other agents in the system.

- for each agent $a = 1, .., N$
  - for each agent $b = 1, .., N$
    if dist(a,b) $<= r_c$ compute interaction of a with b

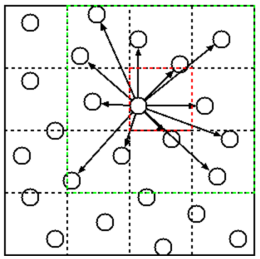
If the number of agents is N, the number of computations to perform to all interactions every time step is $O(N^2)$.



This can, however, be improved if the computation only requires an agent to interact with other agents within a local neighborhood, within cut off distance r_c.
In such cases, the number of computational operations can be reduced to $O(N)$ by using fast neighbor lists, such as **cell list** or **Verlet list**.
We will consider **cell list** here.

Cell List

Cell lists are fast neighbour lists in which the computational domain is partitioned into Cartesian cells of length r_c in each dimension.
The cell-list data structure stores agent indices that reside in each of these cells.
We will consider cell lists for two-dimensional computational domains, but the extension to three-dimensions is straightforward.



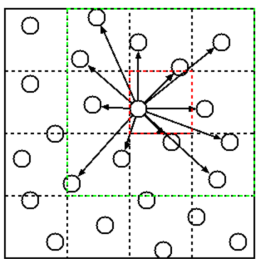
We divide the two-dimensional space into squares of size r_c.
The number of squares (or cells) of size r_c required to tile the two-dimensional space (L_x by L_y) in the x-direction is $N_x = L_x/r_c$ and in the y-direction is $N_y = L_y/r_c$.
Each square is indexed by two indices (i, j) in two-dimensions where $i = 1, ..., N_x$ and $j = 1, ..., N_y$.
Cell list is a data structure that gives the list of all agents in each cell i, j.

Cell List

A simple recipe to build cell list in two-dimensions is as follows:
For each agent $a = 1, ..., N$, with position (x_a, y_a) (in 2D):

- Compute the cell index (i, j) as $i = ceil(x_a/r_c)$ and $j = ceil(y_a/r_c)$
- Add agent a to the cell (i, j).

Once the cell list has been constructed, every agent a has to interact with all other agents in its cell and all adjacent cells.
The number of adjacent cell is 8 in 2D (26 in 3D).



The code for computing agent pairwise interactions will now look like:

- for each agent $a = 1$, .., N compute its cell (i, j)
  - for each adjacent cell (ii, jj) of (i, j)
    - for each agent b in cell (ii, jj)
      if dist(a,b) $<= r_c$ compute interaction of a with b

Cell List

The computational cost of constructing cell list is $O(N)$. If agent density in the domain is close to uniform, the number of neighbors within cutoff distance r_c from given agent is more or less constant, thus giving average cost of cell list of $O(N)$.
The computational cost increases if the agents are non-uniformly distributed within the computational domain. In the worst case, the cost is again $O(N^2)$.

Implementation

Usual steps in the implementation ..

Initialize system: distribute agents
Integrate in time: for each timestep = first .. last:

- For each particle:
  - update particle positions
  - compute particle interactions (eat, replicate, die)

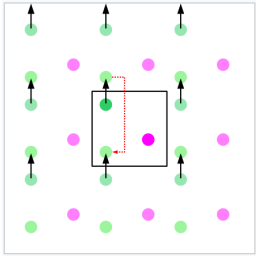
Analyze results

Implementation features we will focus on this week:

- How to compute agent pairwise interactions efficiently?
- How to treat periodic boundary conditions?

Periodic Boundary Conditions

Periodic boundary condition (PBCs) are a set of boundary conditions which are often chosen for approximating a large (infinite) system by using a small part called a unit cell.



When an agent passes through one side of the unit cell, it re-appears on the opposite side with the same velocity.

- for each agent a update its position (x_a, y_a)
  - if periodic_in_x then

    if $(x_a < 0)$ $x_a = x_a + L_x$

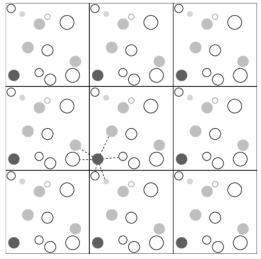
    if $(x_a >= L_x)$ $x_a = x_a - L_x$
  - if periodic_in_y then

    if $(y_a < 0)$ $y_a = y_a + L_y$

    if $(y_a >= L_y)$ $y_a = y_a - L_y$

Periodic Boundary Conditions

When computing interactions between agents a and b, periodic conditions should also be taken into account for computing distance between these agents.



- if periodic_in_x then

  $d_x = x_a - x_b$
  if $(d_x > L_x/2\ )\ d_x = d_x$ - L_x
  if $(d_x <=$ -$L_x/2\ )\ d_x = d_x + L_x$

- if periodic_in_y then

  $d_y = y_a - y_b$
  if $(d_y > L_y/2\ )\ d_y = d_y$ - L_y
  if $(d_y <=$ -$L_y/2\ )\ d_y = d_y + L_y$

Note that periodic boundary conditions will also affect indices of neighboring cells (ii, jj) in cell list implementation.