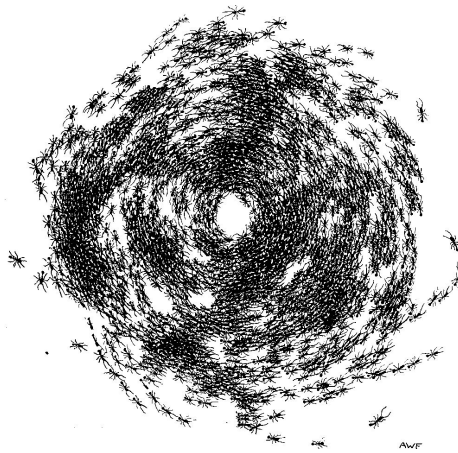
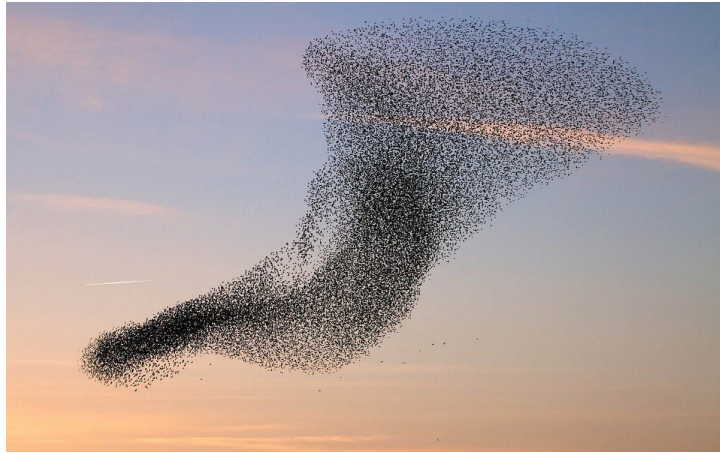


# Modélisation de comportements collectifs

Alexis Raulin-Foissac  
mail: alexis.raulin-foissac@univ-lyon1.fr



# Objectifs du cours

- (première) Expérience d'**application concrète** de méthodes informatiques
- Comprendre la **démarche** de modélisation
- Implémenter efficacement un modèle
- **Observer, analyser, et communiquer** des résultats
- Proposer des améliorations à la modélisation

## Outils et méthodes utilisés:

- python: matplotlib, numpy, ...
- dynamique à N corps: listes de voisin/grille

# Organisation du cours

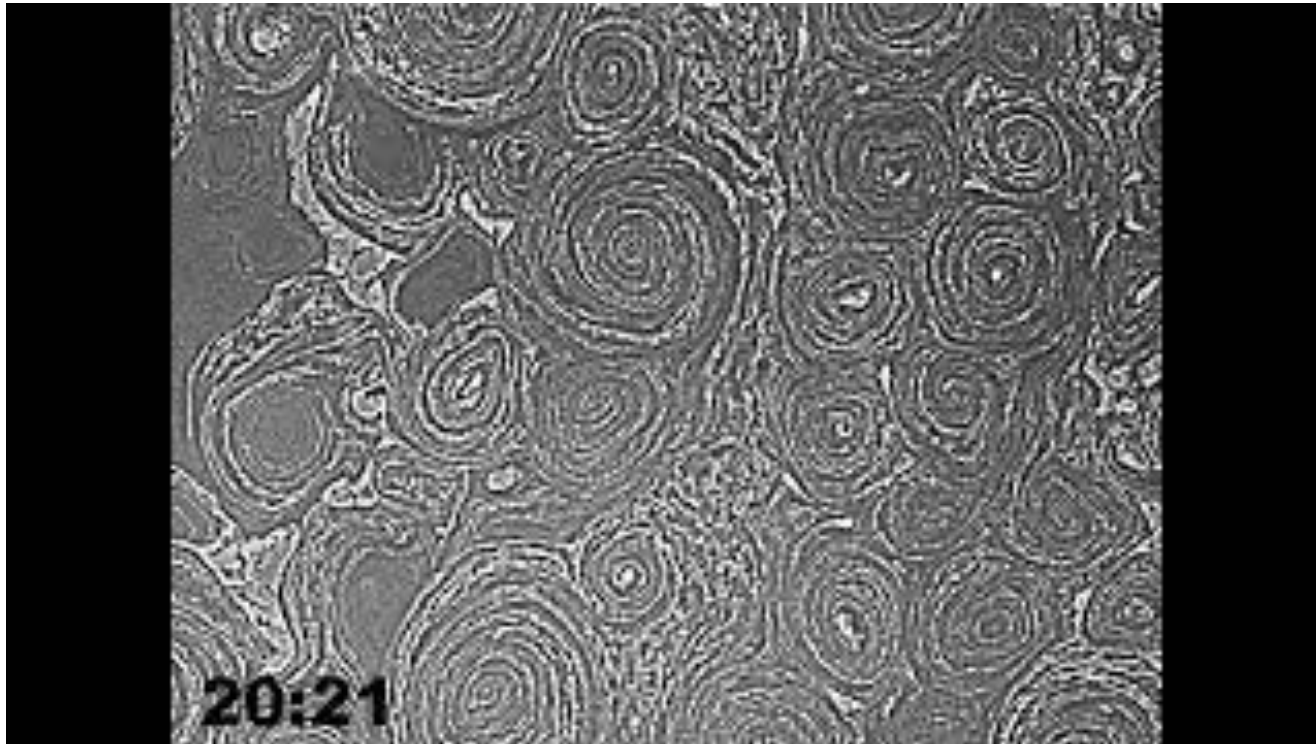
Alternance entre cours et applications sur ordinateur

Notation: investissement/sérieux au fil des séances

## **Plan:**

- I. Introduction et exemples de comportements collectifs
- II. Modélisation d'agents
- III. Implémentation
- IV. Optimisation du code
- V. Exploitation du modèle
- VI. Extensions potentielles

# Mouvements collectifs à travers les échelles



Cellules qui s'organisent pour remplir l'espace efficacement

## Mouvements collectifs à travers les échelles



“Spirale de la mort” de fourmis suivant chacune leur voisin de devant

# Mouvements collectifs à travers les échelles



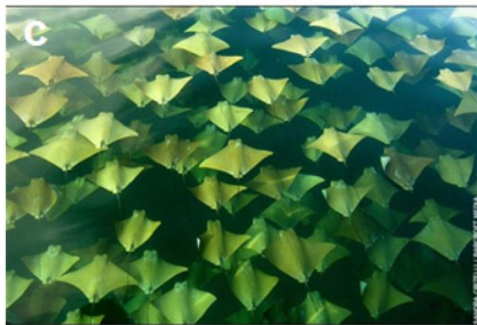
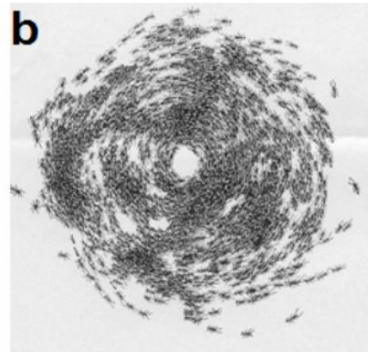
Banc de poissons et d'oiseau

## Mouvements collectifs à travers les échelles



Ondes se propageant dans la foule (Concert d'Oasis, Manchester 2005)

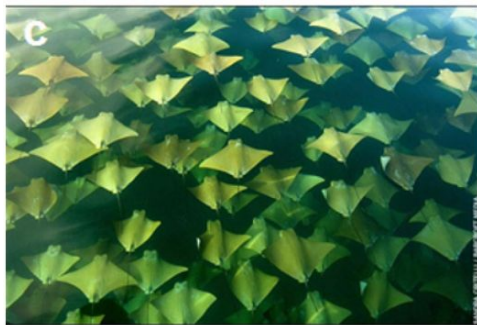
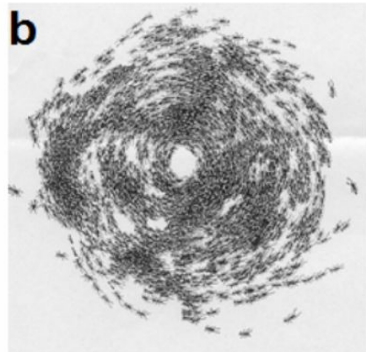
# Mouvements collectifs à travers les échelles



Existe-il des **règles générales** qui régissent ces comportements ?  
Peut-on les **modéliser** efficacement ?



# Mouvements collectifs à travers les échelles



Objectif du cours: **Modéliser** et **reproduire** numériquement ces mouvements collectifs

# Modélisation de ces assemblées

Ingrédients pour réaliser une bonne modélisation:



# Modélisation de ces assemblées

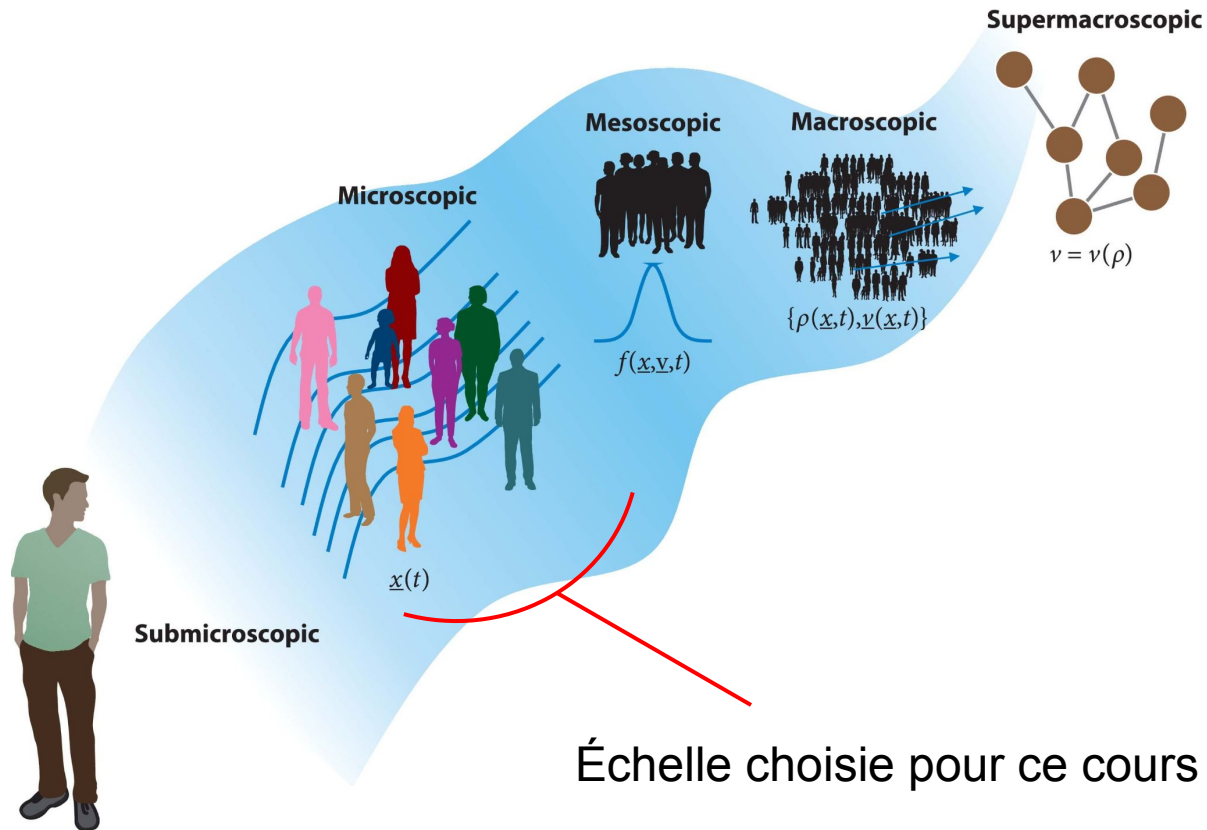
Ingrédients pour réaliser une bonne modélisation:

- **Variables** simples mais exhaustives décrivant les entités (tailles, positions...)
- **Description** de l'environnement (obstacles, conditions aux bords...)
- Ensemble de **règles** simples modélisant:
  - l'évolution des entités
  - interaction inter-particules
  - interactions avec l'environnement

# Variables descriptives

## Différentes échelles

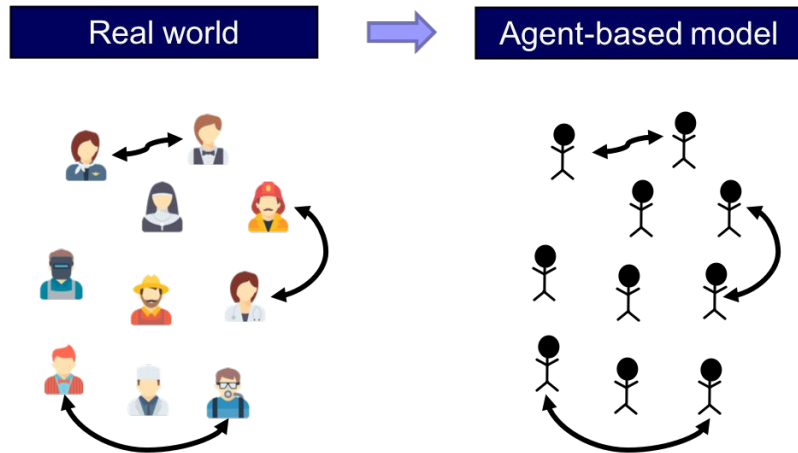
→ différentes **variables** pour décrire le système



# À l'échelle micro

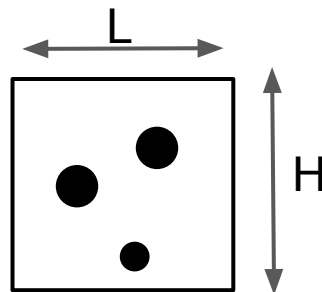
Représentation du système:

- **Coordonnées** :  $(x_i(t), \theta_i(t))$
- **Vitesses** :  $v_i(t)$
- **Taille** des entités (ou forme) :  $R_i$

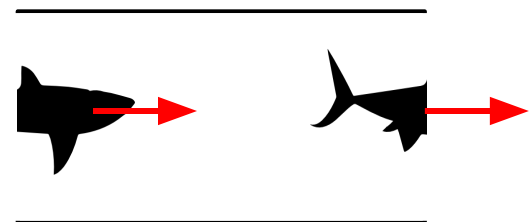


Et de l'environnement:

- Taille et forme
- Obstacles
- Conditions aux bords



Boîte fermée avec obstacles



Couloir infini avec conditions périodiques

# Conditions périodiques pour modéliser un espace infini

Duplique la boîte simulée

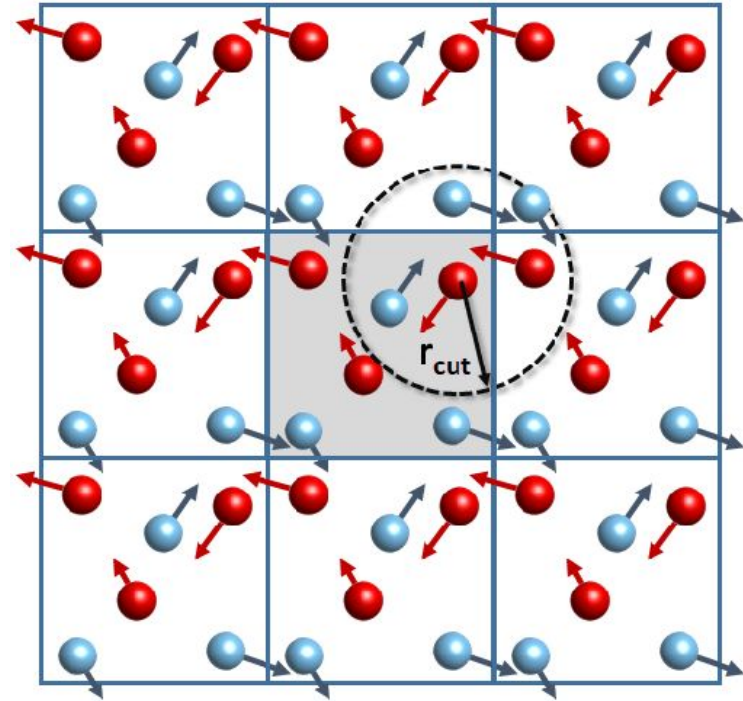
→ comment calculer les distances inter particules ?

Dans un carré de côté  $L$ :

$$\Delta x_{\text{périodique}} = \Delta x - L \times \text{round}\left(\frac{\Delta x}{L}\right)$$

→  $\Delta x_{\text{périodique}} \in [-L/2, L/2]$  est la plus courte distance

Enfinement 
$$d = \sqrt{\Delta x_{\text{périodique}}^2 + \Delta y_{\text{périodique}}^2}$$



Application: Coder `distance_périodique(L, point1, point2)` en python avec `numpy`

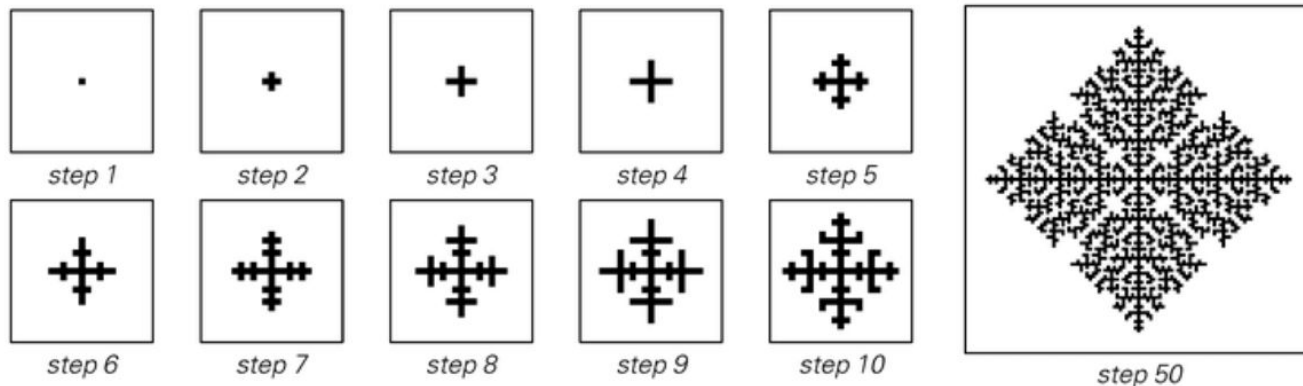
# Évolution du système

**Ensemble de règle** pour passer de  $(r_i(t), \theta_i(t), v_i(t))$  à  $(r_i(t + dt), \theta_i(t + dt), v_i(t + dt))$

pas de temps

Exemples:

Règles discrètes



Automates cellulaires (exemple: jeu de la vie)

# Évolution du système

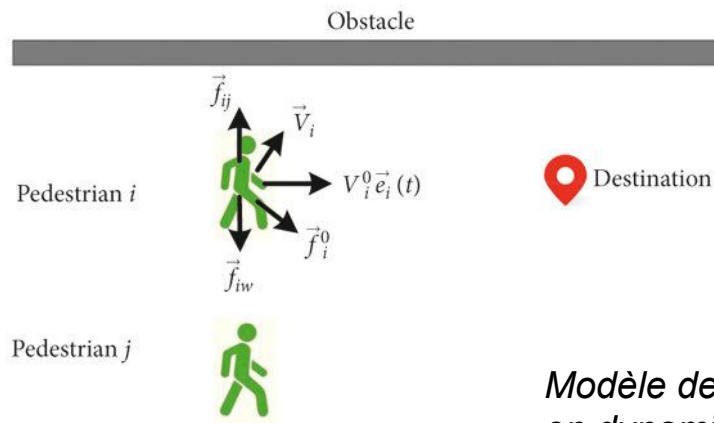
**Ensemble de règle** pour passer de  $(r_i(t), \theta_i(t), v_i(t))$  à  $(r_i(t + dt), \theta_i(t + dt), v_i(t + dt))$

pas de temps

Exemples:

Règles continues avec équations différentielles

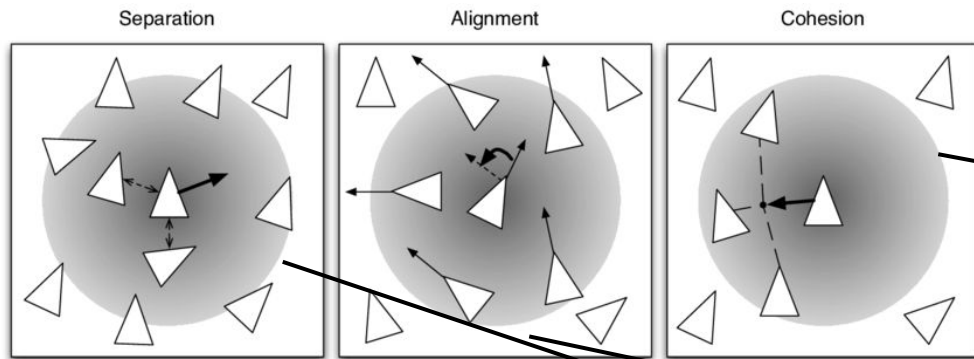
$$\left\{ \begin{array}{l} m \frac{d\vec{v}}{dt} = \sum \vec{F} \\ \frac{d\vec{r}}{dt} = \vec{v} \end{array} \right.$$



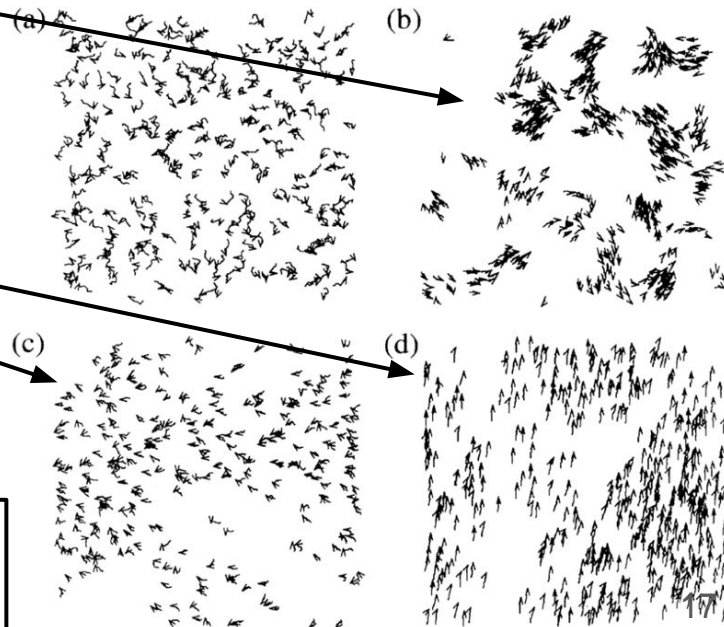


# Évolution du système

Approche à la physicienne: quelques **interactions simples** avec **peu de paramètres**



Exemple de règles simples dans le modèle de Reynolds (Bouraqadi, N., & Doniec, A. (2022, July 05). *Flocking-Ba Multi-Robot Exploration.*)



L'intensité des différents termes dicte la dynamique:

Jhavar, J., Morris, R. G., & Guttal, V. (2018). *Deriving mesoscopic models of collective behaviour for finite populations.*

Des **interactions locales simples** suffisent souvent pour décrire les transitions vers des **comportements collectifs**

# Modèle de Vicsek

“Novel Type of Phase Transition in a System of Self-Driven Particles”, Vicsek (1995):



**Règles** : Particules se déplaçant à vitesses constantes et qui **ajustent** leurs direction suivant la **direction moyenne** de leur voisin. Un **bruit aléatoire** est ajouté pour perturber l’alignement et simuler les incertitudes des agents.

Observation de divers comportements suivant les paramètres:

- alignements complets
- structures de groupes (~ bancs de poisson)
- mouvement désordonné

# Modèle de Vicsek

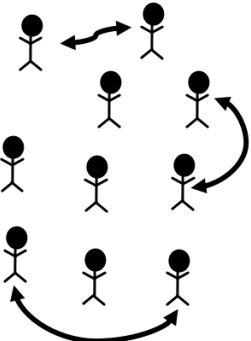
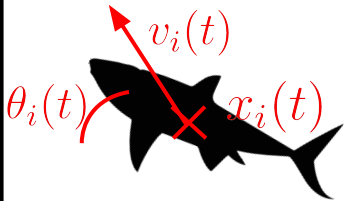
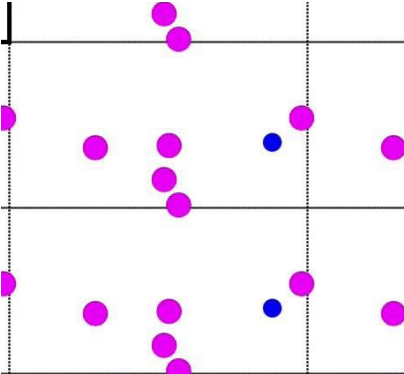
Particules ponctuelles se déplaçant à vitesse  $v_0$  dont la direction  $\theta$  est la moyenne de celle du voisinage plus un bruit:  $\theta_i(t + \Delta t) = \langle \theta_j(t) \rangle_{|r_i - r_j| < R} + \eta_i(t)$

Mise à jour de la position de chaque particule:  $r_i(t + \Delta t) = r_i(t) + v_0 \Delta t \begin{bmatrix} \cos \theta_i(t + \Delta t) \\ \sin \theta_i(t + \Delta t) \end{bmatrix}$

Paramètres:

- $R$  le rayon d'interaction
- $v_0$  la vitesse des particules
- $\Delta t$  le pas de temps
- $\rho$  la densité de particules
- $\eta$  tel que la bruit  $\eta_i(t)$  soit d'amplitude  $2\pi\eta$

# Choix de modélisation effectués:

Échelle	Description	Environnement	Règles d'évolution
Microscopique/individuelle	$x_i(t), v_i(t), \theta_i(t), R_i$	Carré à bords périodiques	Modèle de Vicsek
			$\Theta_i(t + \Delta t) = \langle \Theta_j \rangle_{ r_i - r_j  < r} + \eta_i(t)$ $\mathbf{r}_i(t + \Delta t) = \mathbf{r}_i(t) + v \Delta t \begin{pmatrix} \cos \Theta_i(t) \\ \sin \Theta_i(t) \end{pmatrix}$

# Implémentation: à vous

## Objectifs:

1. Définition des variables pour les positions et orientations
2. À chaque itération:
  - a. trouver les voisins de chaque particule (attention aux conditions périodiques)
  - b. moyenner leurs orientations et rajouter du bruit
  - c. En déduire leurs nouvelles positions et orientations
  - d. Mettre à jour la visualisation matplotlib

Rappel du modèle (avec  $R = \Delta t = v_0 = 1$ ):

$$\theta_i(t + \Delta t) = \langle \theta_j(t) \rangle_{|r_i - r_j| < R} + \eta_i(t)$$

$$\begin{bmatrix} x_i(t + \Delta t) \\ y_i(t + \Delta t) \end{bmatrix} = \begin{bmatrix} x_i(t) \\ y_i(t) \end{bmatrix} + v_0 \Delta t \begin{bmatrix} \cos \theta_i(t + \Delta t) \\ \sin \theta_i(t + \Delta t) \end{bmatrix}$$

Exemples de valeurs pour la taille du carré et la densité:

$$L = 20, N = 0.5 * L^2 = 200$$

# Observations et analyses: paramètre d'ordre

## Qualitativement:

- “clusters/bancs” lorsque le bruit est  $\sim$  nul
- alignement de grands ensemble à faible bruit
- comportement désordonnée à grand bruit

→ Introduction d'un paramètre  $\varphi$  qui caractérise quantitativement l'état du système:

$$\varphi = \frac{1}{Nv_0} \left| \sum_i \vec{v}_i \right|$$

Suivant les états:

- alignement/cohésion :  $\varphi \approx 1$
- désordre :  $\varphi \approx 0$

→  $\varphi$  est un **paramètre d'ordre**

# Observations et analyses: Transitions de phases

## Mesure du paramètre d'ordre

Calculer et afficher  $\phi$  au cours de la simulation avec

```
text = plt.text(x, y, str(phi))
```

```
text.set_text(str(phi)) # mise à jour de la valeur
```

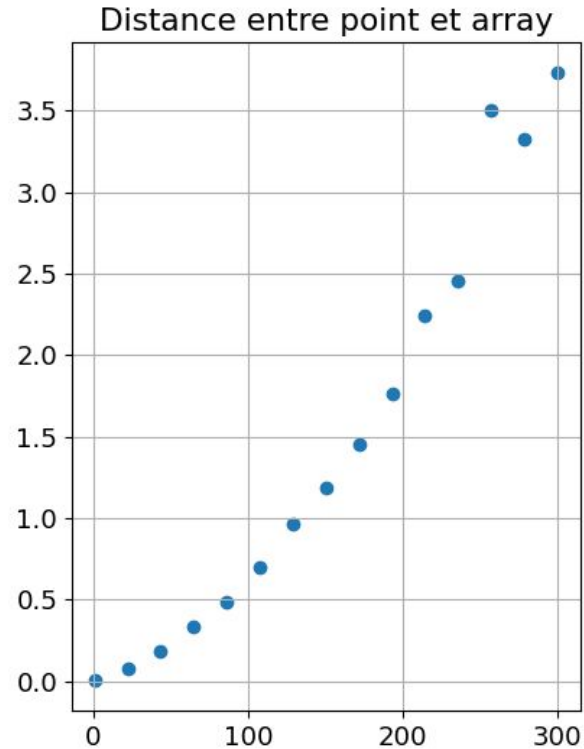
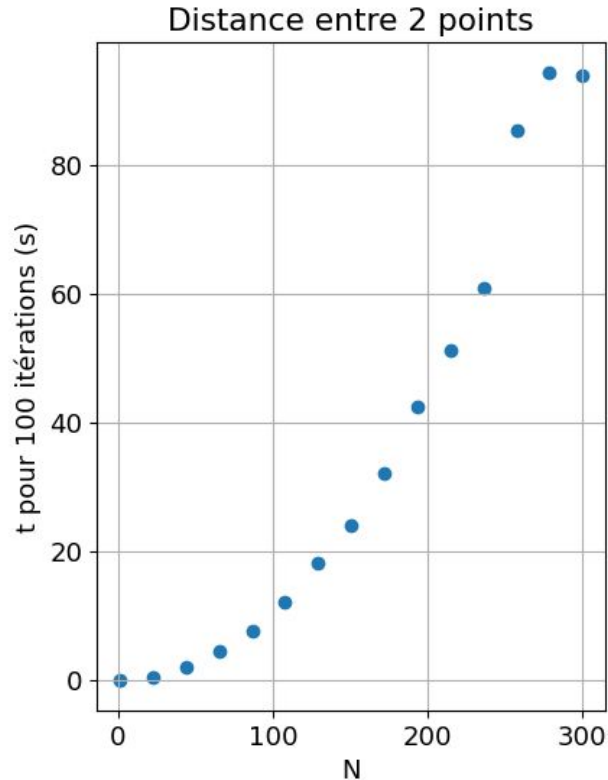
**Problème observé:** dès que l'on augmente trop le nombre d'agents le temps de calcul explose

À quelle complexité temporelle s'attend-on ?

→ mesurer numériquement le temps de calcul en fonction du nombre de particules et vérifier la loi attendue

# Optimisation du code:

Complexité temporelle des algorithmes "naifs"





# Optimisation du code: grille et listes de voisins

Calcul des interactions entre chaque particules → complexité en  $O(N^2)$

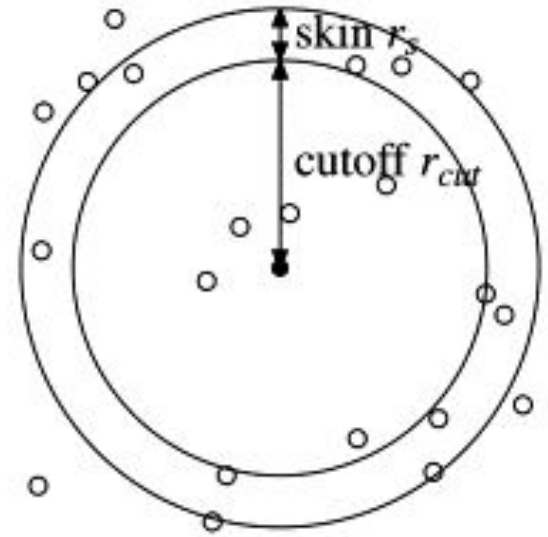
Plusieurs méthodes pour réduire la complexité:

- **Listes de verlet**: stocker les voisins dans une liste que l'on update toutes les  $n$  itérations
- **Discrétiser** l'espace en une **grille** dont chaque case contient les indices des particules qui s'y trouve

# Listes de Verlet

## Implémentation (à faire):

- Créer une liste *voisins* telle que *voisins[i]* contient la liste des potentiels voisins de *i* (de distance maximale légèrement supérieure à la distance d'interaction, par exemple 2)
- Dans *update* chercher les voisins de *i* uniquement parmi *voisins[i]*
- Mettre à jour *voisins* lorsque les particules ont suffisamment bougé (par exemple toutes les  $n = 10$  itérations)



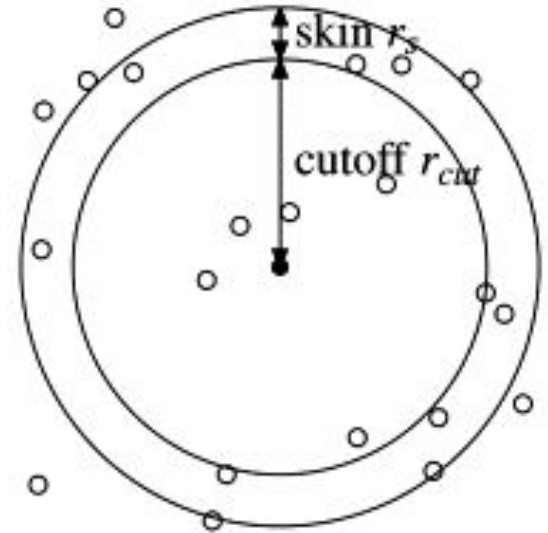
Yao, Z., Wang, J.-S., Liu, G.-R., & Cheng, M. (2004). Improved neighbor list algorithm in molecular simulations using cell decomposition and data sorting method.

Le calcul en  $O(N^2)$  est effectué  $n$  fois moins souvent  $\rightarrow$  complexité  $\sim O(N)$

# Listes de Verlet

## Tests et performances (à faire):

- Regarder l'influence des deux nouveaux paramètres (distance seuil pour mettre dans voisins et temps d'actualisation)
- Mesurer la complexité temporelle de l'algorithme et comparer à ce qui est attendu
- Observer/critiquer la précision de l'algorithme suivant la densité du système



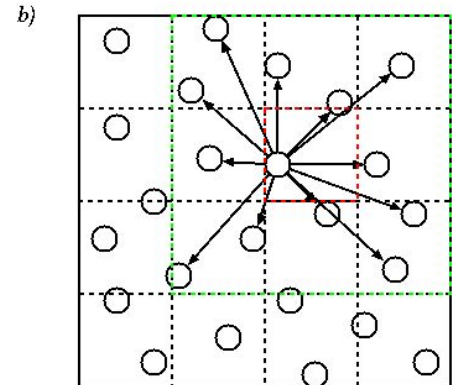
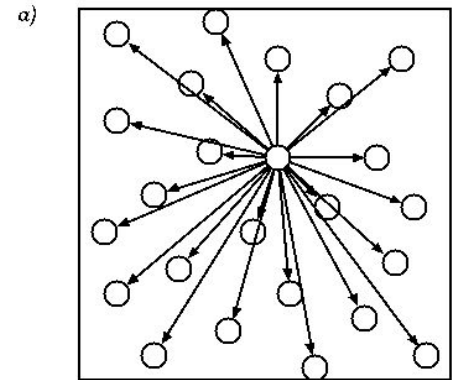
Yao, Z., Wang, J.-S., Liu, G.-R., & Cheng, M. (2004). *Improved neighbor list algorithm in molecular simulations using cell decomposition and data sorting method.*

# Maillage cellulaire

Principe:

- On sépare l'espace en un ensemble de case de la taille d'interaction (ici de taille 1 par 1)
- À chaque itération, attribuer à chaque particule sa case correspondante :  $O(N)$
- Calculer les interactions uniquement avec les particules des cellules voisines à distance d'interaction :  $O(N)$

Efficace à **densité élevée** car à faible densité la plupart des cases sont vides



# Maillage cellulaire

## Implémentation (à faire):

- Introduire un tableau *grille* tel que *grille[i,j]* contient la liste des particules se trouvant dans la case  $i, j$  c'est à dire pour  $i \leq x < i+1$  et  $j \leq y < j+1$
- Chercher les voisins de chaque particule parmi les 9 cases alentours

## Tests et performances:

- Comparer l'efficacité à celles méthodes précédentes
- Tester le module KDTree de scipy

