TP 5 : Modèles multi-agents

Cours de modélisation numérique

24 mars 2023

On modélise dans cette série le comportement de bactéries affamées se déplaçant dans un milieu où est placée une source de nourriture.

Modèle

On se figure des bactéries longues de 1 μm placées dans une boîte de Petri carrée de 100 μm de côté, supposée périodique. Par souci de simplification, on admettra que plusieurs bactéries peuvent occuper la même position simultanément. On définit à présent un "champ de densité de nourriture" $\rho(x)$.

Les bactéries se déplacent par pas de temps $\Delta t = 0.2$ s. À chaque pas de temps, elles peuvent soit continuer d'avancer en ligne droite, soit emprunter une nouvelle direction aléatoire (tirée uniformément) avant d'avancer. Dans un cas comme dans l'autre, leur vitesse de déplacement est toujours de $v = 20$ μm/s. La dynamique est la suivante : si $\rho(x(t)) > \rho(x(t-\Delta t))$, la bactérie poursuit son mouvement rectiligne avec probabilité $P_1 = 0.9$. Sinon, cette probabilité n'est que de $P_2 = 0.5$.

Nous considérons deux types de concentrations possibles. Dans le premier cas, on considérera que

$$\rho_A(x) = \frac{1}{1 + d(x, C)} \tag{1}$$

où C désigne le centre du domaine et $d(a, b)$ la distance euclidienne entre deux points. Dans le second cas, on considérera une concentration

$$\rho_B(x) = 1 \tag{2}$$

si $d(x, C) \leq 15\mu m$ et nulle partout ailleurs.

Méthode

Nous vous proposons d'utiliser le formalisme des *classes* pour modéliser les bactéries. C'est-à-dire que vous devez implémenter une classe `Bacterium`, qui contient des attributs tels que sa position et vitesse de déplacement v, les probabilités P_1 et P_2, et une méthode qui définit son mouvement. Vous utiliserez ensuite une boucle temporelle, qui mettra à jour l'état du système {bactéries, $\rho(x)$} dans l'état suivant.

Voici une brève introduction au concept de classes en *Python*.

Déclaration d'une classe et initialisation

La déclaration d'une classe en Python est similaire aux déclarations des classes en Java et C++ :

```
class MyClass:
    # Déclarations de la classe ...
```

Contrairement à Java et C++ le constructeur des classes en Python ne se définit pas avec le nom de la classe mais avec la méthode `__init__(self)`. Pour continuer l'exemple de la classe précédente :

```
class MyClass:
    # initialiseur de la classe
    def __init__(self):
        # actions de l'initialiseur
```

Il est très important de noter que l'on accède aux méthodes ou attributs d'une classe à l'aide du pointeur `self` (équivalent du `this` en Java et C++). De plus ce pointeur doit faire partie de toutes les signatures des méthodes de la classe et doit impérativement être le premier argument. Par exemple :

```
class MyClass:
    def __init__(self, A=0):
        self.A = A

    def aMethod(self, word):
        print "La classe dit: ", word
```

Dans l'exemple ci-dessus on illustre aussi les arguments par défaut. Dans le constructeur `A` est un argument qui possède une valeur par défaut égale à 0. Ceci permet d'instancier un objet de type `MyClass` de la manière suivante :

```
my_class = MyClass()
```

Dans ce cas l'instance `my_class` aura 0 comme valeur pour l'attribut `A`.

Attributs et méthode d'une classe

En ce qui concerne les attributs, le fonctionnement est aussi peu différent que celui de Java. En Python il n'y a que deux modificateurs possibles : privé et publique. Tous les attributs commençant par « __ » (deux *underscores*) et qui ne se terminent pas par « __ » sont privés. Les autres sont publics. De plus, il n'est pas nécessaire de *déclarer* les attributs. Une simple assignation à `self.attr_name` est suffisante.

Les méthodes se déclarent comme des fonctions, mais on rappelle qu'il ne faut pas oublier de mettre le pointeur `self` comme premier argument dans la signature de la méthode.

Exemple d'utilisation des classes

Voici un exemple simple d'utilisation de classe. On crée une *classe* `Student`, et puis on instancie 3 *objets* `student1, student2, student3`. Enfin, on appelle les *méthodes* `compareAge` et `isSameClass`, qui comparent les *attributs* des objets.

```
class Student:
  def __init__(self, name, age, year=1):
    self.name = name
    self.age = age
    self.year = year

  def compareAge(self, anotherStudent):
    if (self.age > anotherStudent.age):
      print self.name+" is older than " + anotherStudent.name
    elif (self.age < anotherStudent.age):
      print self.name+" is younger than " + anotherStudent.name
    else:
      print self.name+" and " + anotherStudent.name + " have the same age!"

  def isSameClass(self, anotherStudent):
    if (self.year == anotherStudent.year):
      print self.name+" and " + anotherStudent.name + " are in the same class!"
    else:
      print self.name+" and " + anotherStudent.name + " don't know each other, sad..."

if __name__ == "__main__":
  student1 = Student("Alice", 21, 1)
  student2 = Student("Bob", 22, 2)
  student3 = Student("Charlie", 25)        # here year = 1 by default

  student1.compareAge(student2)
  student2.compareAge(student1)
  student1.isSameClass(student3)
  student1.isSameClass(student2)
```

Travail à faire

On considère une population de 100 bactéries. Mesurez la fraction de cette population se trouvant à moins de 15 μm du centre, après N itérations.

Écrivez le code *Python* qui simule la situation de l'énoncé, et qui réalise l'étude demandée. Vous pouvez ensuite représenter vos résultats dans des graphes pour $N = 1, 10, 100, 1000$ itérations. Discutez du comportement observé.