TP 5 : Multi-agent models

Cours de modélisation numérique

24 March 2023

In this TP we will model the behaviour of hungry bacteria moving in an anvironment where a food source is placed.

Model

Consider a picture of bacteria that are $1\mu m$ long, placed in a square Petri dish of side $100\mu m$, assumed to be periodic. For the sake of simplicity we will assume that several bacteria can occupy the same position simultaneously. We now define a "food density field" $\rho(x)$.

The bacteria move during a time step $\Delta t = 0.2$ s. At each time step, they can either continue to move in a straight line, or take new random directions (drawn uniformly) before moving on. In either case, their speed is always $v = 20$ μm/s. The dynamic is the following : if $\rho(x(t)) > \rho(x(t-\Delta t))$, the bacteria continue their rectilinear motion with probability $P_1 = 0.9$. Otherwise, this probability is only $P_2 = 0.5$.

We will consider two possible types of concentrations. In the first case we will consider that

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

where C denotes the center of the domain and $d(a, b)$ the Euclidean distance between two points. In the second case, we will consider a concentration

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

if $d(x, C) \leq 15\mu m$ and zero everywhere else.

Method

We suggest to use the *classes* formalism to model bacteria. That is, you must implement a class `Bacterium` which contains attributes such as the position of the bacteria and speed of displacement v, the probabilities P_1 and P_2, and a method that defines its movement. Then, you will use a time loop which will update the state of the system $\{$bacteria, $\rho(x)\}$ to the next time step.

Here is a brief introduction to the concept of classes in *Python*.

Declaration of a class and initialisation

Declaring a class in Python is similar to declaring a class in Java and C++ :

```
class MyClass:
    # Declaration of the class ...
```

Contrary to Java and C++ the constructor of classes in Python is not defined with the name of the class, but with the method `__init__(self)`. To continue the example from the previous class :

```
class MyClass:
    # class initialisation
    def __init__(self):
        # instantiation
```

it is very important to notice that one accesses the methods or the attributes of a class using the pointer `self` (equivalent to `this` in Java and C++). Moreover, this pointer must be part of all the signatures of the class methods and must always be the first argument. For example :

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

    def aMethod(self, word):
        print "The class says: ", word
```

The above example also shows the default arguments. In the constructor, `A` is an argument which has a default value of 0. This allows you to initiate and object of type `MyClass` in the following way :

```
my_class = MyClass()
```

In this case the instance `my_class` will have 0 as a value for the attribute `A`.

Attributes and methods of a class

As far as attribues are concerned, the way it works has very little differences with Java. In Python there are only two possible modifiers : private and public. All attributes starting with ≪ __ ≫ (two *underscores*) and which do not end with ≪ __ ≫ are private. The others are public. Moreover, it is not necessary to *declare* the attributes. A simple assignment to `self.attr_name` is sufficient.

The methods are declared as functions, but pay attention to not forget to put the pointer `self` as the first argument in the signature of the method.

Example of the use of classes

Here is a simple example of using a class. We create a *class* `Student`, and then we initiate 3 *objects* `student1, student2, student3`. Finally, we call the *methods* `compareAge` and `isSameClass`, which compare the *attributes* of objects.

```
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)
```

To do

Consider a population of 100 bacteria. Measure the fraction of this population that is within 15 μm from the center, after N iterations.

Write a *Python* code that simulates the situation of the initial statement, and that performs the requested study. You can then plot your results for $N = 1, 10, 100, 1000$ iterations. Discuss the observed behaviour.