# 3 Simulation de la radioactivité

## Programme à compléter

L'objectif de cette activité est de tracer l'évolution temporelle d'une population de noyaux radioactifs, grâce à un programme Python et d'étudier cette évolution.

**Activité complémentaire**
Avec un tableur ou python
hatier-clic.fr/pct150b

**Prérequis théoriques**

- Loi de la décroissance radioactive

La question **1** vise à comprendre comment sont construites les deux fonctions permettant de simuler les tirages aléatoires :

- La fonction `untirage`, dont l'objectif est de renvoyer 0 avec une probabilité passée en argument, de renvoyer 1 sinon ;
- La fonction `destirages`, qui utilise la fonction précédente pour calculer le nombre d'éléments restants dans un échantillon après un tirage avec une certaine probabilité de succès.

Le plus facile est d'utiliser une analogie de tirage de dés, pour ensuite utiliser ces fonctions afin de simuler la décroissance radioactive d'un échantillon de noyaux radioactifs.
C'est le but de la question **2** qui adapte et utilise les fonctions ci-dessus pour le suivi au cours du temps des désintégrations au sein d'un échantillon de noyaux.
La question **3** étudie les influences des paramètres sur l'adéquation entre la courbe de décroissance radioactive théorique et les simulations calculées.
Enfin, la question **4** est une application avec des valeurs numériques réalistes, pour un échantillon de noyaux de carbone 14.

## Programme à compléter

```
1  from pylab import *
2  from random import *
3
4  print("************************************")
5  print("*    Simulateur de radioactivite    *")
6  print("* Alternative a l'activite 3 p. 150 *")
7  print("************************Hatier 2020**")
8
9  print("Attention, le separateur decimal est le point")
10
11 ### Donnees
12 UNITE="s"     # symbole de l'unite de temps
13 dt=1          # pas de temps en UNITE
14 l=1/6      # constante radioactive lambda en 1/UNITE
15 N0=long(1e4) # nombre de noyaux initial
16 tmax=25       # date de fin en UNITE
17
18 ### Ne pas modifier : nombre de calculs
19 nt=int(tmax/dt)+1
20
21 ### Ne pas modifier
22 def untirage(p):
23         a=1
24         if random()<p:
25                 a=0
26         return a
27
28 def destirages(N,p):
29         s=0
30         for i in range(N):
31                 s=s+untirage(p)
32         return s
33
34 ### A modifier : nombre de noyaux restants a une date donnee
35 N=[N0]
36 t=[0]
37 for i in range(nt):
38         t.append(t[-1]+dt)
39         N.append(...)
40
41 ### Ne pas modifier
42 Nth=[N0]
43 tt=[0]
44 for i in range(10*nt):
45         tt.append(tt[-1]+dt/10)
46         Nth.append(N0*exp(-l*tt[-1]))
47 plot(t,N,"+",label="Nombre N de noyaux restants simule")
48 plot(tt,Nth,label="Nombre Nth de noyaux restants theorique")
49 xlim(0,tmax)
50 ylim(0,N0)
51 xlabel("t en "+UNITE)
52 ylabel("N")
53 legend()
54 grid(True)
55 show()
```

### Module importé

Le module `pylab` fournit les instructions graphiques utiles à l'activité. Le module `random` fournit la fonction `random()` qui réalise un tirage aléatoire entre 0 et 1.

### À modifier : données (questions 3 et 4)

L'énoncé amène à modifier les données :
- dans la question **3**, pour évaluer l'influence des paramètres `dt`, `tmax` et `N0` ;
- dans la question **4**, pour traiter un cas réel. (La fonction logarithme népérien est `log()` dans Python.)

**Attention** Le séparateur décimal est le point, pas la virgule.
La donnée `N0` et donc toutes les autres valeurs de `N` et `Nth` sont de format `long`, c'est-à-dire que ce sont des grands nombres entiers. La syntaxe `long(1e4)` permet de transformer le nombre 1e4 ($1 \times 10^4$, reconnu par défaut comme un nombre réel à virgule flottante, type `float`) en un long entier.

### À ne pas modifier : calcul du nombre d'itérations nt

Le pas de temps et la durée totale sont les paramètres modifiables. Le nombre d'itérations `nt` en est déduit.

### À ne pas modifier : fonctions aléatoires

Les fonctions `untirage()` et `destirages()`, dont la compréhension est l'objet de la question **1**, utilisent la fonction `random()`, faisant le tirage aléatoire d'un réel entre 0 et 1.

### À compléter : nombre de noyaux restants

La liste `t` et la liste `N` sont initialisées, puis une boucle les complète pas à pas, à l'aide de l'instruction `append`.
Pour ajouter l'élément `x` à la fin de la liste `L`, on écrit `L.append(x)`
Le dernier élément de la liste `L` est `L[-1]`
Cette ligne d'instruction est à compléter à la question **2b** de l'activité pour construire la liste `N`.

### Ne pas modifier : calcul de la courbe théorique et tracés

La liste `Nth` contient les nombres de noyaux restants calculés avec la loi de décroissance radioactive.
Pour obtenir une courbe lissée, ces nombres sont calculés avec un pas de temps dix fois plus petit que `dt`, d'où la création d'une liste `tt` contenant dix fois plus de valeurs que la liste `t`, dix fois plus serrées, donc allant jusqu'à la même valeur maximale du temps.
La fin du programme réalise les tracés.