Polynésie, septembre 2017

- while

Remarque : presque tous les algorithmes des sujets de ES et STMG sont semblables à celui-ci.

On s'intéresse à une population de tortues vivant sur une île et dont le nombre d'individus diminue de façon inquiétante.

Au début de l'an 2000, on comptait 300 tortues. Une étude a permis de modéliser ce nombre de tortues par la suite (u_n) définie par :

$$\begin{cases} u_0 &= 0{,}3 \\ u_{n+1} &= 0{,}9u_n(1-u_n) \end{cases}$$

où pour tout entier naturel n, u_n modélise le nombre de tortues, en milliers, au début de l'année $2000+n$.

On souhaite qu'à la fin de son exécution, l'algorithme ci-dessous affiche la dernière année **avant** laquelle il reste au moins 30 tortues.

```
1 u prend la valeur 0,3 ;
2 n prend la valeur 0 ;
3 tant que ... faire
4 |
  |
  |
  |
5 Afficher ... ;
```

```
1 # -*- coding:utf8 -*-
2 # python 3
3
4 u, n = .3, 0
5 while u > .03:
6     u = .9*u*(1-u)
7     n=n+1
8
9 print("années = ", n)
```

Nouvelle Calédonie, novembre 2017

- Affectation en parallèle
- while
- or

Des scientifiques modélisent, pour tout entier naturel n, cette évolution par :

$$\begin{cases} b_0 = 1\,000 \\ c_0 = 1\,500 \\ b_{n+1} = 0{,}3b_n + 0{,}5c_n \\ c_{n+1} = -0{,}5b_n + 1{,}3c_n \end{cases}$$

où b_n représente approximativement le nombre de buses et c_n le nombre approximatif de campagnols le 1er juin de l'année $2000+n$ (où n désigne un entier naturel).

Lucie exécute l'algorithme ci-dessous et obtient en sortie N = 40. Quelle conclusion Lucie peut-elle énoncer pour les buses et les campagnols ?

```
N prend la valeur 0
B prend la valeur 1 000
C prend la valeur 1 500
tant que B > 2 ou C > 2 faire
    N prend la valeur N + 1
    R prend la valeur B
    B prend la valeur 0,3R + 0,5C
    C prend la valeur
     −0,5R + 1,3C
Afficher N
```

```python
# -*- coding:utf8 -*-
# python 3

N, B, C = 0, 1000, 1500
while (B>2 or C>2):
    N = N+1
    B, C = .3*B + .5*C, -.5*B + 1.3*C
print(N)
```

Centres Étrangers (spé), juin 2017

```
m et n sont des entiers naturels
 non nuls et premiers entre eux
tant que m ≠ n faire
    si m < n alors
        Afficher « gauche »
        n prend la valeur n − m
    sinon
        Afficher « droite »
        m prend la valeur m − n
```

```python
# -*- coding:utf8 -*-
# python 3

n=int(input("n = "))
m=int(input("m = "))
while m !=n:
    if m<n:
        print("gauche")
        n=n-m
    else:
        print("droite")
        m=m-n
```

- test!=
- while
- input() et conversion du type

Centres Étrangers, juin 2017

```
n prend la valeur 4
tant que √(2 / (n sin(2π/n))) > 0,58 faire
    n prend la valeur n + 1
Afficher n
```

```python
# -*- coding:utf8 -*-
# python 3

from math import *

n=4
while sqrt(2/(n * sin(2 *
    ↪pi/n))) > 0.58:
    n = n+1
print("n = ",n)
```

- module math
- while

Quelle valeur numérique de n va afficher en sortie cet algorithme ?

Pondichéry, avril 2017

- module math
- log
- def
- int(input())
- ** (puissance)

... la fonction f définie sur l'intervalle $[-2{,}5\,;2{,}5]$ par : $f(x) = \ln\left(-2x^2 + 13{,}5\right)$.

L'algorithme donné permet de calculer une valeur approchée par défaut de $I = \int_0^{2{,}5} f(x)\,dx$.

Données : R et S sont des réels
n et k sont des entiers

1 S prend la valeur 0
2 Demander la valeur de n
3 **pour** k **de** 1 **jusque** n **faire**
4 R **prend la valeur** $\frac{2{,}5}{n} \times f\left(\frac{2{,}5}{n} \times k\right)$
5 S **prend la valeur** S + R
6 **Afficher** S

```
# -*- coding:utf8 -*-
# python3

from math import log

def f(x):
    return(log(-2*x**2+13.5))

n=int(input("un entier :"))
S=0
for k in range(1,n+1): #de 1 à n (borne sup exclue)
    R = 2.5/n * f(2.5/n *k)
    S=S+R
print(S)
```

Liban, juin 2017

- utilisation des chaîne : indice, opérateur + *
- indice des chaînes / des listes
- reste de la division euclidienne (%)
- if
- affectation en parallèle
- for in <liste>

Un numéro de carte bancaire est de la forme : $a_1a_2a_3a_4a_5a_6a_7a_8a_9a_{10}a_{11}a_{12}a_{13}a_{14}a_{15}c$ où a_1, a_2, ..., a_{15} et c sont des chiffres compris entre 0 et 9.

Les quinze premiers chiffres contiennent des informations sur le type de carte, la banque et le numéro de compte bancaire. c est la clé de validation du numéro. Ce chiffre est calculé à partir des quinze autres.

L'algorithme suivant permet de valider la conformité d'un numéro de carte donné.

```
I prend la valeur 0
P prend la valeur 0
R prend la valeur 0
pour k de 0 jusque 7 faire
    R prend la valeur du reste de la division euclidienne de 2a_{2k+1} par 9
    I prend la valeur I + R
pour k de 1 jusque 7 faire
    P prend la valeur P + a_{2k}
S prend la valeur I + P + c
si S est un multiple de 10 alors
    Afficher « Le numéro de la carte est correct. »
sinon
    Afficher « Le numéro de la carte n'est pas correct. »
```

```
# -*- coding:utf8 -*-
# python 3

numCarte = input() # "5635400295613411"
# attention indices des chaînes commencent à 0

I, P, R = 0, 0, 0
for k in range(8):
    R = (2*int(numCarte[2*k])) % 9
    I = I + R
for k in range(7):
    P = P + int(numCarte[2*k+1])
S = I + P + int(numCarte[15])
if S % 10 == 0:
    print("le numéro "+numCarte+" est valide")
else:
    print("le numéro "+numCarte+" n'est pas valide")
```

Utiliser, modifier l'algorithme pour répondre aux questions (ce sont à peu près celles du sujet)

1. Justifier que le numéro de la carte 5635 4002 9561 3411 est correct.
2. On modifie le numéro de cette carte en changeant les deux premiers chiffres. Le premier chiffre (initialement 5) est changé en 6.
   Quel doit être le deuxième chiffre a pour que le numéro de carte obtenu $6a35400295613411$ reste correct ?
3. Un numéro de carte dont les chiffres sont tous égaux peut-il être correct ? Si oui, donner tous les numéros de carte possibles de ce type.

Asie, juin 2017

- module random
- module mathplotlib
- module numpy
- listes par compréhension ; longueur d'une liste ; ajouter des éléments à une liste

Soit g la fonction définie sur l'intervalle $[0\,;1]$ par $g(x) = \dfrac{1}{1+x^2}$.

On note $\mathcal{C}_g$ sa courbe représentative dans un repère orthonormé du plan et $J = \displaystyle\int_0^1 \frac{1}{1+x^2}\,\mathrm{d}x$.

Le but de cette partie est d'évaluer l'intégrale J à l'aide de la méthode probabiliste décrite ci-après.

On choisit au hasard un point M$(x\,;y)$ en tirant de façon indépendante ses coordonnées x et y au hasard selon la loi uniforme sur $[0\,;1]$.

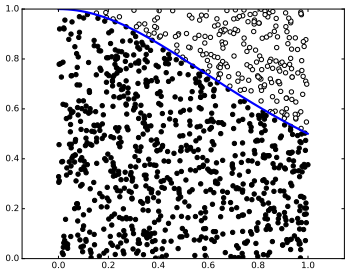
On admet que la probabilité p qu'un point tiré de cette manière soit situé sous la courbe $\mathcal{C}_g$ est égale à l'intégrale J.

En pratique, on initialise un compteur c à 0, on fixe un entier naturel n et on répète n fois le processus suivant :

- on choisit au hasard et indépendamment deux nombres x et y, selon la loi uniforme sur $[0\,;1]$;
- si M$(x\,;y)$ est au-dessous de la courbe $\mathcal{C}_g$, on incrémente le compteur c de 1.

On admet que $f = \dfrac{c}{n}$ est une valeur approchée de J. *C'est le principe de la méthode dite de Monte-Carlo.*

exemple avec 1 000 *points*



La figure illustre la méthode :

n points ont été placés aléatoirement dans le carré.

Les disques noirs correspondent aux points sous la courbe, les disques blancs aux points au-dessus de la courbe.

Le rapport du nombre de disques noirs par le nombre total de disques donne une estimation de l'aire sous la courbe.

Recopier et compléter l'algorithme ci-après pour qu'il affiche une valeur approchée de J.

```
1 Lire la valeur de n
2 c prend la valeur ...
3 pour i de 1 jusque ... faire
4 |   x prend la valeur aléatoire
  |     entre 0 et 1
5 |   y prend la valeur...
6 |   si ... alors
7 |   |   ...prend la valeur ...
8 f prend la valeur ...
9 Afficher f
```

```
1  # -*- coding:utf8 -*-
2  # python 3
3
4  from random import uniform
5
6  n, c= 1000, 0
7  for i in range(n):
8      x, y = uniform(0,1),
       ↪uniform(0,1)
9      if y < 1/(1+x**2):
10         c = c +1
11
12 print("fréquence = ", c/n)
```

Proposition d'améliorations du programme

```
# -*- coding:utf8 -*-
# python 3

from random import uniform
from matplotlib import pyplot as plt

n, c= 1000, 0
for i in range(n):
    x, y = uniform(0,1), uniform(0,1)
    if y < 1/(1+x**2):
        plt.plot(x,y,'ko')
        c = c +1
    else:
        plt.plot(x,y,'wo') # si white ne donne rien :
    ↪changer la couleur

# une méthode possible pour afficher la fonction
X = [k/1000 for k in range(1000)]
Y = [1/(1+x**2) for x in X]
plt.plot(X,Y,'b-',linewidth=3)

plt.title("n = "+str(n)+"; f = "+str(c/n))
plt.axis('equal')
plt.axis('on')
plt.show()
```

```python
# -*- coding:utf8 -*-
# python 3

from random import uniform
from matplotlib import pyplot as plt
import numpy as np

# on accelere l'affichage en travaillant sur des listes
    ↪de points
# le package <numpy> contient la fonction <linspace>
# en mode console :
# import numpy as np
# help(np.linspace)

n = 10000
xSur, xSous, ySur, ySous = [],[],[],[] # liste des
    ↪coordonnées
for i in range(n):
    x, y = uniform(0,1), uniform(0,1)
    if y < 1/(1+x**2):
        xSous, ySous = xSous + [x], ySous + [y]
    else:
        xSur, ySur = xSur + [x], ySur + [y]

# Affichage des points dans le repère et de la courbe
plt.plot(xSous,ySous, marker='o', color='black',
    ↪linestyle = 'none')
plt.plot(xSur,ySur, marker='o', color='white', linestyle
    ↪ = 'none')

# Affichage de la fonction
x = np.linspace(0,1,200)
y = 1/(1+x**2)
plt.plot(x,y,  "r-", linewidth=3)
plt.title("n = "+str(n)+" et f = "+str(len(xSous)/n))
plt.axis('equal')
plt.axis('on')
plt.show()
```

Amérique du Sud, septembre 2017

- for ... range
- module numpy
- calcul matriciel
- module sympy
- Rational

$U_n = (a_n\,;b_n\,;s_n)$ est la matrice qui à l'étape n donne la probabilité qu'un joueur soit dans l'équipe A (a_n) ou dans l'équipe B (b_n) ou soit solitaire (s_n).

On donne l'algorithme suivant, où la commande « U[i] » renvoie le coefficient de la i-ème colonne d'une matrice ligne U.

U prend la valeur $\begin{pmatrix}0 & 0 & 1\end{pmatrix}$
T prend la valeur
$\begin{pmatrix} \frac{3}{5} & \frac{3}{20} & \frac{1}{4} \\ \frac{1}{5} & \frac{3}{5} & \frac{1}{5} \\ \frac{3}{14} & \frac{9}{14} & \frac{1}{7} \end{pmatrix}$
pour k de 1 jusque 7 faire
    U prend la valeur UT
Afficher U[1]

```python
# -*- coding:utf8 -*-
# python 3

import numpy as np # permet le calcul matriciel

U = np.array([0,0,1])
T = np.array([[3/5, 3/20, 1/4],
              [1/5, 1/5, 3/5],
              [3/14, 9/14, 1/7]])
for k in range(7):
    U = np.dot(U,T)

print(U[0]) # attention indice
```

1. Quelle est la valeur numérique arrondie au millième de la sortie de cet algorithme ? L'interpréter dans le contexte de l'exercice.
2. Recopier et modifier cet algorithme pour qu'il affiche la fréquence de joueurs solitaires au bout de 13 jours.

Python a un module « sympy » qui permet d'effectuer des calculs avec les rationnels, les lettres…

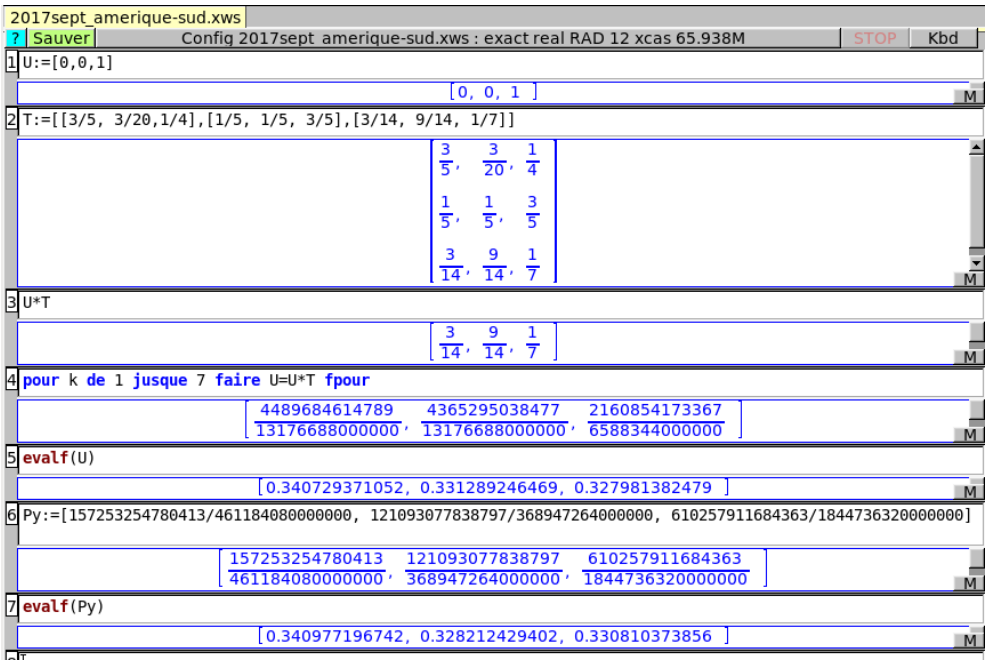
```python
# -*- coding:utf8 -*-
# python 3

from sympy import * # permet le calcul littéral

U = Matrix([[0,0,1]]) # attention crochets
T = Matrix([[Rational(3,5), Rational(3,20), Rational(1,4)],
            [Rational(1,5), Rational(1,5), Rational(3,5)],
            [Rational(3,14), Rational(9,14), Rational(1,7)]])
for k in range(8):
    U = simplify(U * T)

print(U)
```

mais je ne dois pas le maîtriser ou bien, j'ai un parti-pris pour Xcas ;-)


2017sept_amerique-sud.xws
? Sauver
Config 2017sept_amerique-sud.xws : exact real RAD 12 xcas 65.938M
STOP
Kbd
1 U:=[0,0,1]
[0, 0, 1]
2 T:=[[3/5, 3/20,1/4],[1/5, 1/5, 3/5],[3/14, 9/14, 1/7]]
3 U*T
4 pour k de 1 jusque 7 faire U=U*T fpour
5 evalf(U)
[0.340729371052, 0.331289246469, 0.327981382479]
6 Py:=[157253254780413/461184080000000, 121093077838797/368947264000000, 610257911684363/1844736320000000]
7 evalf(Py)
[0.340977196742, 0.328212429402, 0.330810373856]