TP 2 : Erreurs d'arrondi, algorithmes simples

Exercice 1. *Erreurs d'arrondi : suite logistique*

Soit $x_0 \in [0,1]$ et posons $x_{n+1} = 4x_n(1-x_n)$ pour $n \geq 0$.

1. Étant donnée $x_0 = 0.23$ (de type `float`), calculer x_{10} puis x_{60}.

   On écrira un code qui utilise seulement deux variables simples (et pas de listes).

2. Définissons à présent la suite suivante : $y_0 = 0.23$, $y_{n+1} = 4y_n - 4y_n^2$ pour $n \geq 0$. Notons que formellement c'est la même suite que (x_n), mais définie avec une expression développée.

   Calculer y_{10} puis y_{60} (en utilisant l'expression développée).

   Est-ce qu'on obtient les mêmes valeurs que dans la question précédente ?

   Si non, d'où vient la différence ? Lequel de ces deux résultats vous semble correct ?

   On mettra les réponses dans le script (en commentaire).

Exercice 2. *Suite instable, suite stable aux erreurs d'arrondis*

Soit $a \in \mathbb{R}$, $a > 1$. Pour tout $n \in \mathbb{N}$, $n \geq 1$, on considère $v_n = \displaystyle\int_0^1 \frac{x^{n-1}}{a+x}\, dx$.

1. (*sur papier*) Montrer que pour tout $n \in \mathbb{N}^*$, on a : $\dfrac{1}{n(a+1)} \leq v_n \leq \dfrac{1}{na}$.

   Montrer que la suite (v_n) peut être définie par la récurrence :

   $$v_1 = \log\left(\frac{1+a}{a}\right), \quad v_n = \frac{1}{n-1} - a\, v_{n-1}, \text{ pour } n \geq 2. \tag{1}$$

   Indication : On pourra calculer $v_n + av_{n-1}$.

2. Le but est de calculer v_{40} à l'aide de Python en utilisant la relation de récurrence (1).

   Tester, en particulier, le cas $a = 3$. Le résultat vous paraît-il correct ? Afficher les valeurs $v_1, \ldots, v_{40}$.

3. Une autre stratégie consiste à faire le calcul en partant d'une valeur estimée de v_{60} (par exemple la moyenne des 2 bornes de l'encadrement précédent) et à utiliser la relation de récurrence pour "descendre" jusqu'à obtenir une valeur approchée de v_{40}.
   Compléter le script de l'exercice (prendre la même valeur de a que précédemment) pour obtenir une valeur approchée de v_{40} avec cette stratégie.

Exercice 3. *Écriture binaire*

1. Écrire une fonction `inverse(L)` qui retourne une liste `L` dans l'ordre inverse (cf. l'aide mémoire).

2. Dans la suite de cet exercice, on va manipuler des nombres en base 2. Pour les représenter dans la machine, on pourra utiliser une liste composée de 0 et de 1. Par exemple, le nombre 1101_2 en base 2 correspond naturellement à la liste `[1,1,0,1]`.

   Écrire une fonction `dec(B)` qui, étant donné un nombre en base 2 (représenté par une liste `B`), retourne sa valeur numérique en base 10 (cf. CM). Tester: `dec([1,1,0,1])` doit retourner 13.

3. A l'aide de la fonction `dec`, calculer $1101100_2 + 10101010_2$. On affichera le résultat en base 10.

4. Écrire une fonction `binaire(n)` qui, étant donné un entier retourne son écriture en base 2 (sous forme d'une liste de 0 et de 1). Tester : `binaire(13)` retourne `[1,1,0,1]`.

5. Calculer $1101100_2 + 10101010_2$ et afficher le résultat en base 2.

6. Afficher l'écriture binaire de la factorielle de 50. Quelle est sa longueur ? *Indication* : Pour utiliser la fonction factorielle (qui n'est pas définie dans `numpy`), on tape :
   `from math import factorial`.

Exercice 4. *Suite de Fibonacci*

Soit $(u_n)_{n\geq 0}$ la suite définie par $u_0 = 0$, $u_1 = 1$ et $u_n = u_{n-1} + u_{n-2} \quad$ pour tout $n \geq 2$.

1. Écrire une fonction `fibo_liste(n)` qui renvoie une liste contenant la suite de Fibonacci de u_0 jusqu'a u_n (inclus). Afficher les 20 premières valeurs de la suite.

2. Écrire une fonction `fibo(n)` qui renvoie u_n en utilisant seulement des *variables simples* (pas de listes). Afficher les 20 premières valeurs de la suite.

3. En prenant en compte les 100000 premiers termes de la suite de Fibonacci, trouver la somme `S` des termes qui sont des nombres paires. Afficher `S` modulo 10000007.

   Attention ! Pour ne pas saturer la mémoire, dans cette question on **s'interdit d'utiliser des listes** (et donc la fonction `fibo_liste`) ! On pourra en revanche *s'inspirer* du code de la fonction `fibo` (sans appeler la fonction elle-même).

Exercice 5. *Algorithme d'Euclide*
L'algorithme d'Euclide permet de déterminer le plus grand commun diviseur de deux nombres naturels (si besoin, consulter Wikipédia)

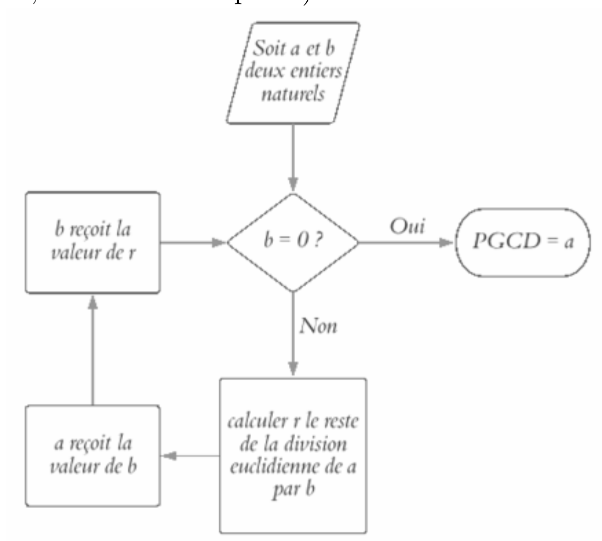

Image:Wikipédia

1. En utilisant une boucle `while`, écrire une fonction `pgcd(a, b)` qui, en effectuant l'algorithme d'Euclide, renvoie le plus grand commun diviseur de deux entiers positifs `a` et `b`.

2. Tester : `pgcd(495,275)` vaut 55; pensez à d'autres exemples faciles à vérifier à la main.

Exercice 6. *Suite de Farey*

1. Écrire une fonction `pgcd(a, b)` qui effectue l'algorithme d'Euclide et renvoie le plus grand commun diviseur de `a` et `b`. Calculer `pgcd(123456, 234567)`.

2. La fonction φ d'Euler est la fonction qui à tout entier n non nul associe le nombre d'entiers strictement positifs inférieurs ou égaux à n et premiers avec n, i.e. le nombre des entiers `k` tels que $0 < k \leq$ `n` et `pgcd(k, n) = 1`. Par exemple, $\varphi(10) = 4$.

   Écrire une fonction `phi(n)` qui calcule la fonction φ d'Euler.

   À l'aide de cette fonction, calculer le nombre de fractions irréductibles strictement positives et strictement inférieures à 1, ayant pour le dénominateur $q = 30$.

3. La suite de Farey d'ordre n est la suite des fractions irréductibles entre 0 et 1 (inclus) dont le dénominateur est inférieur ou égal à n et en ordre croissant. Chaque suite de Farey commence avec la valeur 0, décrite par la fraction $\frac{0}{1}$, et finit avec la valeur 1, décrite par la fraction $\frac{1}{1}$.

   Par exemple, la suite de Farey d'ordre 4 est $\dfrac{0}{1}, \dfrac{1}{4}, \dfrac{1}{3}, \dfrac{1}{2}, \dfrac{2}{3}, \dfrac{3}{4}, \dfrac{1}{1}$.

   Écrire une fonction `farey(n)` qui renvoie la *longueur* de la suite de Farey d'ordre `n`. Par exemple `farey(4)` retournera 7.

   Indication : Pour connaître la longueur de la suite de Farey d'ordre n, il suffit de compter les fractions irréductibles (entre 0 et 1) dont le dénominateur est inférieur ou égal à n.

4. Calculer et afficher la liste des valeurs `farey(n)`, pour `n` variant de 10 à 30 (on affichera un terme par ligne).