Calcul des termes successifs d'une suite récurrente

On considère f une fonction définie sur un domaine D stable par f, c'est à dire que $f(D) \subset D$. On note (u_n) la suite à valeurs réelles définie par :

$$\begin{cases} u_0 \in D \\ \forall\, n \in \mathbb{N},\ u_{n+1} = f(u_n) \end{cases}$$

Généralement, on dit qu'une telle suite représente un **système dynamique discret**. Dans ce cas particulier, on pourra même remarquer qu'il s'agit d'une **suite récurrente à un pas**, puisque la seule connaissance de u_n nous permettra de déterminer u_{n+1}.

Concrètement si u_0 est donné, on calcule ainsi les premiers termes de suite :

$$u_1 = f(u_0),\ u_2 = f(u_1),\ u_3 = f(u_2),\ \ldots,\ u_{n+1} = f(u_n)$$

Une première approche itérative

Bien entendu, en informatique, on peut toujours programmer une telle suite en stockant les résultats dans des variables successives, mais on sera vite limité par la mémoire de la machine...
En fait, on peut se limiter à l'utilisation de deux variables :

- une première variable U_0 contenant le terme précédent ;
- une seconde variable U_1 contenant le terme suivant.

de sorte qu'on procèdera de la façon suivante :

$$\underbrace{u_1}_{U1} = f(\underbrace{u_0}_{U0}),\ U0 = U1,\ \underbrace{u_2}_{U1} = f(\underbrace{u_1}_{U0}),\ U0 = U1,\ \underbrace{u_3}_{U1} = f(\underbrace{u_2}_{U0}),\ U0 = U1,\ \ldots,\underbrace{u_{n+1}}_{U1} = f(\underbrace{u_n}_{U0})$$

Il faudra juste veiller à déplacer la valeur obtenue de $U1$ à $U0$ entre deux étapes successives.

Remarques

1. On pourra retenir que c'est le premier calcul qui nous donne le **schéma numérique** de travail. Ici, on a :

   $u_1 = f(u_0)$ et donc, on aura à chaque étape :

   $$U1 = f(U0),\ U0 = U1$$

2. Pour une suite récurrente à pas multiples, on peut facilement adapter les choses. Par exemple, si un terme est défini à partir des deux précédents :

   $u2 = f(u_0, u_1)$ et donc, on aura à chaque étape :

   $$U2 = f(U1, U0),\ U0, U1 = U1, U2$$

Dans le langage Python, il est donc très simple de construire un algorithme itératif qui affiche les termes successifs d'une telle suite, puis renvoie le terme u_n :

```python
def u(n):
    U0=u0     # on initialise
    if n==0:
        return U0
    else:
        for k in range(1,n+1):
            U1=f(U0)     # on calcule le terme suivant à partir du terme précédent
            print(k,U1)     # on affiche l'indice et le résultat obtenu
            U0=U1     # on déplace la valeur pour le prochain calcul
        return U1
```

On peut aussi utiliser une boucle **while** pour calculer les termes de la suite à condition d'utiliser un compteur :

```python
def u(n):
    k,U0=0,u0     # on initialise
    if n==0:
        return U0
    else:
        while k<=n:
            U1=f(U0)     # on calcule le terme suivant à partir du terme précédent
            k=k+1
            print(k,U1)     # on affiche l'indice et le résultat obtenu
            U0=U1     # on déplace la valeur pour le prochain calcul
        return U1
```

Exercice 1 - Etude d'une suite arithmetico-géométrique

On appelle suite arithmetico-géométrique toute suite (u_n) vérifiant pour tout $n \in \mathbb{N},\ u_{n+1} = au_n + b$, avec $(a,b) \in \mathbb{R}^2, a \neq 1$. On pose pour le reste de l'exercice $u_0 = b = 1$ de sorte que (u_n) est définie par :

$$\begin{cases} u_0 = 1 \\ \forall\, n \in \mathbb{N},\ u_{n+1} = au_n + 1 \end{cases}$$

1. Calculer les cinq premiers termes de la suite pour $a = 2$ et $a = \frac{1}{2}$.
2. Dans le langage Python, construire la fonction u qui pour tout couple (a,n) donné, affiche les termes successifs, puis renvoie la valeur de u_n.
3. Quelle hypothèse pouvez-vous faire quant à son comportement asymptotique pour $a = 2$ et $a = \frac{1}{2}$?

On définit alors la suite (v_n) telle que pour tout $n \in \mathbb{N}$, $v_n = u_n - \frac{1}{1-a}$.

4. Justifier que les suites (u_n) et (v_n) sont de la même nature.
5. Construire la fonction v qui pour tout couple (a,n) donné, affiche les termes successifs, puis renvoie la valeur de v_n, puis calculer $v(2,100)$ et $v(\frac{1}{2},100)$.
6. Quelle hypothèse pouvez-vous alors faire quant au comportement asymptotique de la suite (u_n) pour a quelconque $(a \neq 1)$?

Exercice 2 - Calcul des termes successifs de la suite de Fibonacci

On définit la suite de Fibonacci par :

$$\begin{cases} u_0 = u_1 = 1 \\ \forall\, n \in \mathbb{N},\ u_{n+2} = u_{n+1} + u_n \end{cases}$$

Cette suite désigne en fait une **suite récurrente à deux pas**, puisque cette fois-ci la valeur de u_{n+2} dépend des deux termes précédents u_n et u_{n+1}.

1. Calculer les dix premiers termes de la suite de Fibonacci.
2. Dans le langage Python, construire la fonction *fibonacci* qui pour tout entier n donné, affiche les termes successifs, puis renvoie la valeur de u_n.
3. En utilisant votre programme, déterminer le plus petit entier n à partir duquel $u_n \geq 1000000$.
4. Construire le programme *indice* qui pour tout réel M donné, calcule les termes de la suite de Fibonacci tant que $u_n < M$ puis renvoie le plus petit entier n pour lequel $u_n \geq M$. On ajoutera un test conditionnel sur M afin que le programme renvoie un message d'erreur si $M < 3$.

Cas particulier des programmes récursifs

Il existe en fait une autre façon de déterminer les termes successifs de telles suites. En reprenant un système dynamique de la forme :

$$\begin{cases} u_0 = \lambda,\ \lambda \in \mathbb{R} \\ \forall\, n \in \mathbb{N},\ u_{n+1} = au_n + b \end{cases}$$

on peut adapter cette définition au langage Python et construire une fonction **récursive**, c'est à dire que le programme pourra s'appeler lui-même :

```python
def u(n):
    if n==0:
        return λ
    else:
        return a*u(n-1)+b
```

Concrètement, pour calculer la valeur de u_n, l'interpréteur lancera plusieurs fois l'exécution du programme pour des indices différents jusqu'à atteindre la valeur initiale, avant de **remonter la pile des calculs**. Il faudra ainsi s'assurer de la **terminaison** d'une telle procédure et on vérifiera à chaque fois que :

1. l'indice ou le compteur lié au calcul évolue bien vers son premier terme,
2. la condition initiale est bien atteinte.

Remarque C'est une façon très efficace de définir un algorithme de calcul des termes successifs d'une suite numérique, mais il y aura quelques limites... ne serait-ce que le nombre d'appels dans la pile de calculs qui est souvent limité par le logiciel lui-même.

Exercice 3 - Programmation récursive de la fonction factorielle

On rappelle que pour tout $n \in \mathbb{N}$,

$$n! = \begin{cases} 1 \text{ , si } n = 0 \\ \prod_{k=1}^{n} k \text{ , si } n \neq 0 \end{cases}$$

1. Donner une définition récursive de la fonction factorielle.

2. Dans le langage Python, construire la fonction récursive $facto$ qui pour tout entier n donné, renvoie la valeur de $n!$. On ajoutera un test conditionnel sur le paramètre n, afin de tester si $n \in \mathbb{N}$.

Exercice 4 - Calcul récursif des termes successifs de la suite de Fibonacci

On rappelle que la suite de Fibonacci est définie pour tout $n \in \mathbb{N}$ par :

$$\begin{cases} u_0 = u_1 = 1 \\ \forall\, n \in \mathbb{N},\ u_{n+2} = u_{n+1} + u_n \end{cases}$$

1. Construire le programme itérative $fibo$ qui, pour tout entier n donné, renvoie la valeur de u_n.

2. Construire le programme récursive $fiborec$ qui, pour tout entier n donné, renvoie la valeur de u_n.

3. On souhaite comparer le temps d'exécution de ces programmes en fonction de n. Pour cela, on pourra faire appel à la librairie **time** et la commande `time()` qui déclenche un chronomètre basé sur l'horloge du processeur.

   Construire le programme *comparaison* qui, pour tout entier n donné, calcule le temps d'exécution de $fibo(k)$ et $fiborec(k)$ pour k allant de 0 à n, puis renvoie les graphes représentant le temps d'exécution de chaque programme en fonction de k. On pensera à ajouter une légende pour illustrer ces graphes.