

---

# T.D. D'INFORMATIQUE *PC\**

---

## *Mise en jambe*

18 septembre 2003

Le symbole (\*) signifie que la question est subsidiaire et/ou hors programme.

### § 1. ENTRE RATIONNELS ET FLOTTANT : LA QUESTION DE LA STABILITÉ

Considérons la suite suivante définie par la relation de récurrence :

$$U_{n+1} = 111 - \frac{1130}{U_n} + \frac{3000}{U_n U_{n-1}}$$

et par un couple de valeurs initiales réelles  $U_0 = a$  et  $U_1 = b$ .

#### 1.1.

Écrivez une procédure prenant en paramètres un entier  $n$  et deux valeurs  $a$  et  $b$  et calculant le  $n$ -ième terme de cette suite. Prenez par exemple  $a$  et  $b$  rationnels. Normalement, vous devriez voir apparaître un rationnel.

#### 1.2.

Comment lire le résultat en flottant ? Transformez votre procédure de manière à obtenir un résultat flottant. Utilisez la fonction `Digits` pour obliger MAPLE à produire un résultat avec  $d$  chiffre en base 10,  $d$  étant maintenant un paramètre de votre procédure.

#### 1.3.

Testez le résultat pour  $d = 10$  et par exemple  $n = 10$ ,  $b = 61/11$ ,  $a = 11/2$  ou bien  $a = 5, 5$ . Que constatez-vous ? À votre avis, à quoi cela est-il dû ?

#### 1.4. (\*)

L'explication : la suite  $(U_n)$  que nous venons de calculer peut aussi se décrire de la manière suivante : soit  $g : \mathbb{R}^2 \setminus \{(0, 0)\} \rightarrow \mathbb{R}^2 \setminus \{(0, 0)\}$  la fonction définie par :

$$g(x, y) = \left( y, 111 - \frac{1130}{y} + \frac{3000}{xy} \right).$$

Notons  $P_n$  le point  $(U_n, U_{n+1})$ . Alors  $g(P_n) = P_{n+1}$  pour tout  $n$ . Calculez les points fixes de  $g$ . On utilisera brutalement la fonction `solve`.

Maintenant on veut savoir si ces points fixes sont *attractifs* ou *répulsifs*, ou encore ni l'un ni l'autre. Une manière de classifier ces cas consiste à étudier les valeurs propres de la matrice jacobienne de la fonction aux points fixes.

Calculez les matrice jacobienne de  $g$  aux trois points fixes. On les appellera  $J_5$ ,  $J_6$  et  $J_{100}$ . On utilisera la fonction `jacobian` de la librairie `linalg`. Maintenant pouvez-vous dire qui est attractif et qui ne l'est pas ? On utilisera `Eigenvalues`.

## § 2. LA SUITE DE FIBONACCI OU LA MAÎTRISE DU "for" ET DU "if"

### 2.1.

Soit  $a$  et  $b$  deux réels. Nous définissons la suite de Fibonacci initialisée en  $a$  et  $b$  comme la suite réelle  $(F_n)$  définie par la récurrence suivante :

$$F_{n+2} = F_{n+1} + F_n,$$

et  $F_0 = a$  et  $F_1 = b$ . Construisez une procédure qui prend  $n$ ,  $a$  et  $b$  en paramètres et qui calcule le  $n$ -ième terme de cette suite. Nous l'appellerons `fibonacci_gene`. Il existe une fonction MAPLE qui calcule la suite de Fibonacci. C'est la fonction `fibonacci` du package `combinat`. On remarquera que cette suite est définie pour les valeurs initiales  $a = 0$  et  $b = 1$ . Vous pouvez alors vérifier votre calcul pour ces valeurs.

### 2.2.

Nous fixons maintenant  $a = b = 1$ . On a alors la propriété suivante :

**Proposition § 2.1** Pour tout entier  $n \in \mathbb{N}^*$ , on a :

$$F_{2n} = F_n^2 + F_{n-1}^2 \text{ et } F_{2n-1} = 2F_n F_{n-1} - F_{n-1}^2.$$

Cette propriété peut faire l'objet d'un petit exercice sur les récurrences linéaires. Libre à vous de le démontrer chez vous.

Écrivez une procédure qui calcule  $F_n$  avec ces formules. Nous l'appellerons par exemple `fibonacci_sq`. Pourquoi est-ce que `fibonacci_sq` est plus rapide que `fibonacci_gene`? Pouvez-vous estimer en fonction de  $n$  le nombre d'opérations élémentaires (multiplications additions) dans chacun des cas.

### 2.3.

En réalité, la philosophie de la question précédente est très générale. Il s'agit de décomposer une récurrence en fonction du développement en base 2 de l'indice. Nous allons donner un exemple de telle méthode. Mais tout d'abord, revenons à notre suite de Fibonacci  $(F_n)$ . La récurrence linéaire du début fournit l'égalité matricielle suivante :

$$\begin{pmatrix} F_{n+2} & F_{n+1} \\ F_{n+1} & F_n \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix}.$$

Et donc

$$\begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n+1}.$$

Déduisez-en une procédure facile de calcul de  $(F_n)$ . Attention, vous allez avoir besoin de manipuler des matrices, il faut donc charger la librairie `linalg`.

Cette procédure pourrait sembler tout aussi longue que `fibonacci_gene`. Mais il n'est rien, car le calcul de la puissance d'une matrice (ou d'autre chose), s'effectue par une méthode basée sur la décomposition de l'exposant en base 2. Cette méthode s'appelle la *méthode des carrés itérés*.

## 2.4. (\*)

Nous expliquons ici la méthode des carrés itérés en ce qui concerne les matrices (par exemple).

Nous souhaitons calculer  $X^n$  où  $X$  est une matrice carré et  $n$  un entier. Il y a une méthode naïve qui consiste à calculer  $n$  produits :  $X * X$ , puis  $(X * X) * X$ , etc... C'est évidemment beaucoup trop long ! L'idée dite de la méthode des "carrés itérés" est la suivante : pour calculer  $X^8$ , on calcule 3 carrés  $X^2$ ,  $X^4$  et  $X^8$ . Pour calculer  $X^{13}$ , on calcule

$$X^{13} = X * (X^2 * (X^2)^2)^2,$$

ce qui fait 3 calculs de carré et 2 multiplication, au lieu de 13 !

Si `exp_sq(X, n)` était une telle procédure de calcul de  $X^n$  par carré itérés. Alors pour  $X^{13}$ , la démarche serait la suivante :

`exp_sq(X, 13)` renvoie  $X * \text{exp\_sq}(X, 6)^2$ ,

`exp_sq(X, 6)` renvoie  $\text{exp\_sq}(X, 3)^2$ ,

`exp_sq(X, 3)` renvoie  $X * \text{exp\_sq}(X, 1)^2$

et enfin `exp_sq(X, 1)` vaut  $X$ .

Vous n'avez plus qu'à écrire l'algorithme correspondant. Pouvez-vous majorer le nombre d'opérations élémentaires qu'il nécessite ?