1 Algèbre linéaire et utilisation de tableaux

1.1 Utilisation de tableaux

Pour mettre en mémoire les valeurs successives d'une suite, on peut utiliser ce que l'on appelle un tableau (ce que l'on appelait une mémoire indexée). Par exemple pour définir le terme initial d'une suite récurrente $(u_n)_{n\in\mathbb{N}}$, on peut écrire sous MAPLE $> u[0] := 1;$ si $u_0 = 1$. Ensuite l'écriture de $> u[0];$ nous renverra la valeur de u_0. Le seul problème que vous pouvez rencontrer est l'utilisation de variables déjà définies. Par exemple, essayez : $> x := 3; x[2] := 1; x;$ et $y[1] := 3; y := 4; y[1];$.

1.1.1 Programmation de suites récurrentes simples

L'idée est de programmer une suite définie par la donnée de son premier terme et une relation de récurrence du type : $u_{n+1} = f(u_n)$. On veut de plus mettre en mémoire toutes les valeurs de la suite jusqu'à un certain rang N.

1. On va par exemple considérer la suite vérifiant

$$\forall n \in \mathbb{N}, u_{n+1} = \frac{u_n + \frac{3}{u_n}}{2} \quad \text{et} \quad u_0 = 2$$

   À l'aide d'une boucle, calculez et mettez en mémoire toutes les valeurs de la suite jusqu'au 10 ème terme. Est-il possible d'aller jusqu'à des rangs plus élevés ? Quels problèmes rencontrez-vous ? Comment les résoudre ? Sachant que la suite converge vers $\sqrt{3}$, évaluer la différence entre u_7 et la limite (utiliser **Digits**).
   Pour dessiner le graphe d'une suite on pourra se servir de la procédure suivante :

```
> dessin:=proc(u,N)
> local i;
> plot([seq([i,u[i]],i=0..N)]);
> end;
```

2. En utilisant la fonction **isprime(n)** qui vous renvoie "true" lorsque n est premier et "false" dans le cas contraire, définir la suite (v_n) telle que v_n est égal au nombre de nombres premiers inférieurs ou égaux à n. Observer sur des graphiques l'évolution de (v_n)

.

1.1.2 Calcul d'une suite récurrente double

Programmez et calculez les premiers termes de la suite de Fibonacci :

$$\forall n \in \mathbb{N}, \quad u_{n+2} = u_{n+1} + u_n, \quad \text{avec } u_0 = 1 \text{ et } u_1 = 1$$

Observer l'évolution de la suite $v_n = \ln u_n$. Conclusion ?

1.2 Algèbre linéaire : calculs de puissances de matrices

D'abord précisons certaines fonctions de MAPLE :

- La fonction **time()** renvoie le temps de calcul effectué depuis l'ouverture de MAPLE. Voici une utilisation : **deb:=time():10000!:fin:=time():fin-deb;**. Une autre façon d'utiliser la fonction est de mettre la commande en arguments. Par exemple **time(10000!)**
- Pour utiliser des commandes d'algèbre linéaire, inclure la librairie : **with(linalg):**
- Pour définir une matrice : **matrix(2,2,[1,2,3,4])**, pour définir une matrice aléatoire : **randmatrix(7,7);**, pour définir une matrice diagonale : **diag(1,2,3,4);** ou bien **diag(seq(i,i=1..7));**
- Pour multiplier deux matrices : **multiply(A,B);** ou bien **evalm(A&*B);**
- Pour diagonaliser une matrice : **>jordan(A);** ou bien **>jordan(A,P);** P contient alors la matrice de passage, c'est à dire que $P^{-1}AP$ est la matrice "jordanisée" (diagonale en général).

1. Lorsqu'on multiplie deux matrices 30×30, combien de multiplications, combien d'additions sont effectuées ? Lorsqu'on met une matrice 30×30 à la puissance 1000 combien, a priori, de multiplications et d'additions sont effectuées ? Fabriquer une matrice aléatoire 30×30 et évaluer le temps de calculs. Modifier les tailles et les puissances pour essayer d'étudier la complexité du problème dans le cas général. Par exemple comparer le nombre d'opérations nécessaires pour calculer une matrice 20×20 à la puissance 3000, et une matrice 30×30 à la puissance 500. Comparer ensuite les temps de calculs.
2. On suppose que l'on ne dispose pas de la fonction puissance pour les matrices. Fabriquer une procédure qui permet de calculer la puissance d'une matrice, à l'aide d'une récurrence simple.
3. (difficile) On se propose d'écrire un algorithme de calcul de puissance beaucoup plus performant. L'idée est la suivante : on va mettre en mémoire les puissances dont l'ordre est une puissance de 2 : A^1, A^2, A^4, A^8, A^{16} ...jusqu'à la dernière puissance qui ne dépasse pas la puissance voulue. Par exemple, pour calculer A^{10}, on ne calculera pas A^{16} mais on s'arrêtera à A^8. Puis on fera le calcul :

   $$A^{10} = A^8 \times A^2$$

   Ce qui représente un gain important en terme de calculs.
   Indication : $2^n \leq k \iff n \ln 2 \leq \ln k \iff n \leq \frac{\ln k}{\ln 2} \iff n \leq E(\frac{\ln k}{\ln 2})$. La fonction partie entière s'écrit **floor** sous MAPLE.
4. Enfin, une dernière méthode consiste à diagonaliser la matrice. Si on a $\Delta = P^{-1}AP$ alors on montre facilement que $A = P\Delta P^{-1}$ puis par récurrence que $A^k = P\Delta^k P^{-1}$ pour tout $k \in \mathbb{N}$. Or il est très facile de calculer Δ^k ! Par exemple, considérons la matrice :

   $$A = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$$

   Si on définit la suite de fibonacci comme à l'exercice précédent, on peut alors définir une suite de matrices colonnes :

   $$U_n = \begin{pmatrix} u_n \\ u_{n+1} \end{pmatrix}$$

   (4.1) Vérifier qu'on a bien $\forall n \in \mathbb{N}, \quad U_{n+1} = AU_n$ et donc $U_n = A^n U_0$ d'où l'intérêt de calculer A^n.

   (4.2) Diagonaliser A à l'aide de Maple, et en déduire A^n, puis u_n pour tout $n \in \mathbb{N}$. Retrouver les résultats trouvés à l'exercice sur les suites de Fibonacci.