Programmation : Suites et applications

Nicolas Gast - nicolas.gast@ens.fr

21 novembre 2005

ATTENTION : pensez à sauvegarder régulièrement votre travail.

Ce TD sera consacré au calcul de suite et notemment plusieurs exemple de calcul de valeurs approchées à l'aide de suite. Le premier exercice est une courte introduction ayant pour but de vous sensibiliser aux problèmes de complexité afin d'accélérer un peu les calculs que vous ferez faire à votre machine.

Exercice 1. Suite de Fibonacci et complexité

Une suite récursive est souvent définie par un terme initale et une fonction de transition f tel que $u_{n+k} = f(u_{n+k-1}, \dots, u_n)$. Par exemple la suite de Fibonacci est définie par :

$$(u_n)_{n\geq 0} = \begin{cases} u_0 = u_1 = 1 \\ u_{n+2} = u_{n+1} + u_n \end{cases}$$

Pour programmer cette suite, la méthode la plus naturelle est d'utiliser une fonction récursive.

```
fibo := proc(n)
  if n = 0 or n = 1 then 1
  else fibo(n-1) + fibo(n-2);
  fi;
end;
```

a. Programmer cet algorithme et tester le pour $n = 10$ puis $n = 30$. Que pensez vous de sa complexité (*ie* le nombre d'opérations que effectuées par l'algorithme) ?

Le problème de cet algorithme est qu'il calcule plusieurs fois les termes de la suite u_n. Pour éviter ce problème, plusieurs méthodes sont possibles. La première (la plus simple) est d'ajouter l'option remember à la procedure en tapant `option remember;` après `fibo := proc(n)`. La deuxième, plus compliquée mais qui a l'avantage de marcher dans tous les langages de programmation, est de faire un programme *itératif* et non *récursif* qui stoque lui-même les valeurs au fur et à mesure des calcul.

```
fibo_ite := proc(n)
local fib,i;
fib[0] := 1; fib[1] := 1;
  for i from 2 to n do
    fib[i] := fib[i-1]+fib[i-2];
  od;
end;
```

b. Programmer ces deux méthodes et essayez les pour calculer `fibo(100)` (et trouver 573147844013817084101).

Exercice 2. Suite convergente ?

$$\text{Soit } (u_n)_{n\geq 0} = \begin{cases} u_0 = \frac{3}{2} \\ u_1 = \frac{5}{3} \\ u_{n+1} = 2003 - \frac{6002}{u_n} + \frac{4000}{u_n . u_{n+1}} \end{cases}$$

a. Montrer que $u_n = \frac{1+2^{n+1}}{1+2^n}$

b. Écrire un programme qui calcule le n$^{\text{ième}}$ terme de la suite $(u_n)_{n\geq 0}$ en faisant des *calculs exacts*.

c. Adapter le programme ci-dessus afin que les opérations faites soient approchées (par exemple remplacer 3/2 par 1.5).

d. Y a-t-il une différence entre les résultats ? D'où peuvent-il provenir ?

Exercice 3. Calcul de racine carré

On définit la suite $\left(x_n\right)_{n\geq 0}$ par $x_n = \frac{x_n}{2} + \frac{a}{2x_n}$ et $x_0 = 1$

a. Montrer que x_n converge vers $\sqrt{a}$ (indication : Montrez que $\forall x > 0 : f(x) > x \Leftrightarrow x < \sqrt{a}$)

b. Écrire un programme `x(n,a)` qui calcule le n$^{\text{ième}}$ terme de la suite $\left(x_n\right)_{n\geq 0}$.

c. Taper `Digits := 20` et comparer pour plusieurs valeurs de `a` les valeurs obtenues par $x(10, a)$ et $\sqrt{a}$. La suite converge très rapidement, essayez d'expliquer pourquoi.

Exercice 4. * Approximation de racine

Il est souvent utile de résoudre l'équation $f(x) = b$. Malheureusement même en connaissant l'expression exacte de f, il est parfois difficile de trouver une expression de l'inverse ou d'en calculer une valeur approchée.

Un algorithme possible pour calculer une valeur approchée est de choisir une valeur initiale x_0 puis de construire par récurrence une suite tel que x_{n+1} soit la racine de la tangente à la courbe en x_n.

a. Appliquer cet idée à la fonction $1/x$ et en déduire un algorithme permettant de calculer l'inverse d'un nombre x avec une bonne précision en utilisant que des multiplications et des additions.

b. Remarquer que cet algorithme appliqué à $\sqrt{}$ donne les mêmes formules qu'à l'exercices précédant.

Exercice 5. * Suite de Syracuse

On définit la suite de Syracuse par :

$$\begin{cases} u_o = N \\ \text{Si } 2|u_n : u_{n+1} = \frac{u_n}{2} \\ \text{Sinon } u_{n+1} = 3u_n + 1 \end{cases}$$

On définit les notions suivantes :
- *le temps de vol* : c'est le plus petit indice n tel que un = 1
- *le temps de vol en altitude* : c'est le plus petit indice n tel que un+1¡ N
- *l'altitude maximale* : c'est la valeur maximale de la suite
- *le facteur d'expansion* : c'est le rapport entre le nombre de départ et l'altitude maximale

a. Écrire une (ou plusieurs) fonctions qui calculent les différents paramètres énoncés ci dessus.

b. Si le cœur vous en dit, ecrivez un programme qui dessine la suite sur un graphique (en abscisse n, en ordonné $u(n)$).

(Note : bien que tout le monde en soit acutellement convaincu, on ne sait toujours pas si le temps de vol d'une telle suite est finie, bien que beaucoup de mathématiciens se soient penchés dessus dans les années 60. Ce problème est connu sous le nom de Conjecture de Syracuse `http://fr.wikipedia.org/wiki/Conjecture_de_Syracuse`)