Suites et applications

Nicolas Gast - nicolas.gast@ens.fr

20 novembre 2006

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.

Procédure Récursive/Itérative : Une procédure *récursive* est une procédure qui "se rappelle", c'est à dire que pour calculer le résultat d'ordre n, elle utilise le résultat d'ordre $n-1$. Une procédure *itérative* est une procédure qui n'est pas récursive, elle utilisera souvent des boucles `for` ou `while`.

```
fac_recu := proc(n)
   if n = 0
     then 1
     else n*fac_recu(n-1)
   fi;
end;
fac_recu(10);
                  3628800
```

```
fac_ite := proc(n)
  r := 1;
  for i from 1 to n do
     r := r*i;
  od;
  r;
end;
fac_ite(10);
                  3628800
```

Complexité : la complexité d'un algorithme est l'ordre de grandeur du nombre d'opérations effectuées par le programme pour retourner le résultat. Dans le cas de la factorielle ci dessus, le nombre d'opérations est n.

Exercice 1. Suite de Fibonacci et complexité

Une suite La suite de Fibonacci est définie par :

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

a. Écrire une procédure récursive qui rend le n$^{\text{ième}}$ terme de la suite de Fibonacci.

b. (sur papier) En posant c_n le nombre d'opérations effectuées par l'algorithme, trouver une relation de récurence entre c_n, c_{n+1} et c_{n+2}.

c. Evaluer c_n pour n=10 et 20. Que pensez vous de l'algorithme récursif ?

d. Écrire une procédure *itérative* calculant le n$^{\text{ième}}$ terme de la suite de Fibonacci.

e. Comparer les performances des deux algorithmes.

Exercice 2. Suite convergente ?

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

a. (sur papier) 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 $x_{n+1} = \frac{x_n}{2} + \frac{a}{2x_n}$ et $x_0 = 1$

a. * (sur papier) 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 := 100` et comparer pour plusieurs nombres les valeurs obtenues par $x(10, a)$ et $\sqrt{a}$. Remarquer que la suite converge très rapidement.

Exercice 4. La méthode d'Euler

Une équation différentielle est une équation de la forme

$$\frac{\partial y}{\partial x} = f(x, y)$$

La résolution d'équation différentielles étant une opération difficile, le besoin de simulation nous pousse à utiliser des méthodes de résolution approchée. Pour faire cela, on utilise une *discrétisation* de la fonction et de sa dérivée :

$$y'(x) \simeq \frac{y(x+h) - y(x)}{h}$$

où h est un nombre "*petit*". $y(x+h) \simeq y(x) + h.y'$

On constuit ensuite les différentes valeurs de la fonction par récurrence :

$$x_n = x_{n-1} + h \quad (1)$$
$$y_n = y_{n-1} + h.f(x_{n-1}, y_{n-1}) \quad (2)$$

Pour visualiser la solution obtenue, on stockera les différents x[n] dans deux variables globale `X` et `Y` `X[N]` $= x_n$. Pour tracer la courbe, on utilisera ensuite la commande `plot([seq([X[i],Y[i]],i=1..n)])`.

a. Appliquer cette méthode à la résolution de $\frac{\partial y}{\partial x} = -2.x.y$ (resp $\frac{\partial y}{\partial x} = -2.y^2$) et $y(0) = 1$ en faisant varier le pas ($h \sim 0.01$ à 0.0001). Comparer les résultats obtenus à la solution exacte.

La méthode d'Euler se généralise aisément à un système de n équations différentielles du premier ordre (et donc aux équations différentielles d'ordre n).

b. Le système de Lotka-Volterra est un modèle de référence de l'écologie. F(t) représente le nombre de proies (Food) et P(t) représente le nombre de prédateurs présents. Appliquer la méthode d'Euler à la résolution du système et tracer les deux courbes F et P sur un même graphique.

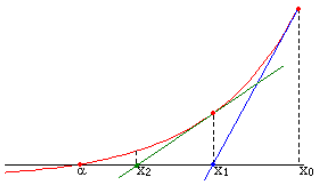
$$\begin{cases} \dot{F} = \alpha F - \beta FP \\ \dot{P} = \gamma PF - \delta P \end{cases}$$

Exercice 5. * Méthode de Newton : résolution approché de $f(x) = 0$

Il est souvent utile de résoudre l'équation $f(x) = 0$. 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.

La méthode de Newton, ou méthode de Newton-Raphson, est un algorithme efficace pour trouver des approximations du zéro (ou racine) d'une fonction à valeurs réelles.

L'approche graphique est la suivante : On choisit une valeur d'abscisse x_0 raisonnablement proche du vrai zéro. On remplace alors la courbe par sa tangente $D : y = f(x_0) + f'(x_0)(x - x_0)$ et on calcule le point d'intersection entre D et la droite $y = 0$. On nomme l'abscisse de ce point x_1. Ce point est généralement plus proche du zéro de la fonction, et la méthode peut être réitérée.



a. (sur papier) Montrez que

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

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

c. Appliquer ces idées à la fonction $1/x$ et en déduire un algorithme permettant de calculer l'inverse d'un nombre x en n'utilisant que des multiplications et des additions.