Informatique Appliquée au Calcul Scientifique

Cours 2

Quelques rudiments de programmation

1) Un principe de base
2) Sommer une série ; boucles
3) Séries divergentes
4) Branchement

François Dubois, 12 octobre 2004, édition 03 août 2006, 12 pages.

CNAM Paris

IACS (2)

Quelques rudiments de programmation

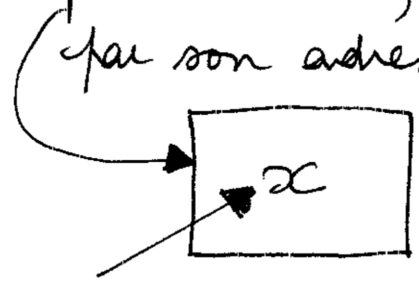
(1) Un principe de base

- Nous avons vu dans la première leçon que l'ordinateur manipule des nombres "réels flottants" avec le logiciel 'octave', en les stockant avec 64 bits élémentaires. En fait le 'mot binaire' (de 64 bits) qui permet de coder un nombre x est physiquement rangé dans une place de la mémoire de l'ordinateur, sur le disque dur a priori. Nous imaginons maintenant effectuer les trois opérations suivants:

1) aller chercher en mémoire le nombre 'x' qui est stocké

2) ajouter 'un' au nombre x au sein de l'unité de calcul de l'ordinateur

3) Remettre le résultat obtenu dans la place mémoire qui contenait initialement le nombre 'x'.



<u>Figure 1</u> Un nombre x est rangé en mémoire.

- Ces opérations sont très claires et simples en soi. Toutefois, dans les langages 'évolués' de programmation, on confond la place physique en mémoire, ou son identification symbolique via son adresse dans la mémoire et le contenu de la mémoire. Cette confusion va créer des abus d'écriture, choquants pour les mathématiciens, mais consacrés par l'usage. On pourrait représenter les trois opérations précédentes (aller chercher x, lui ajouter 1, remettre le résultat à la place initiale) par une symbolique comme

(1) $\qquad x \leftarrow x+1.$

Toutefois, cette notation n'est pas consacrée par l'usage, au moins pour les logiciels qui effectuent des calculs numériques, comme 'octave' ou son clone commercial 'matlab'. On note la suite d'instructions (1) sous la forme

(2) $\qquad x = x+1.$

- Il faut faire attention que la relation (2) n'est pas une phrase mathématique, mais un codage d'instructions qui signifie en fait (1) et ordonne à la machine d'effectuer les trois opérations introduites au début de ce paragraphe.

Ceci étant dit, une instruction telle que (2) est d'usage constant en programmation. Faire effectuer un calcul, une simulation, à un ordinateur demande de disposer au départ de données, introduites d'une manière ou d'une autre dans la mémoire, d'en faire un certain traitement, puis de transmettre le résultat à l'utilisateur, en stockant à nouveau les données dans la mémoire.

(2) Sommer une série, boucles.

- Une instruction telle que (2) permet, quand on l'itère, de calculer des sommes. Ainsi, une suite d'instructions transmises à 'octave' telles que

$$x = 1$$
$$S = 0$$
$$S = S + x$$
$$S = S + x$$
$$S = S + x$$

va avoir l'effet suivant : introduire le nombre 1 dans la machine, dans une 'case mémoire' qui le contiendra sous l'appellation 'x', introduire dans une case mémoire qui contient une variable 's' le nombre 'zéro', aller chercher les contenus des deux cases mémoires précédentes et les ajouter. Replacer le résultat dans la case qui contenait la variable 's'. On a alors s = 1 à l'issue de cette première étape de calcul. Recommencer l'opération précédente. Alors s = 2 à l'issue de cette seconde étape. Recommencer une fois encore. Alors s = 3, ie le contenu mémoire de la variable 's' contient une chaîne de symboles binaires qu'on interprète avec la norme IEEE comme le nombre 'trois'.

- Au lieu d'écrire trois fois de suite l'instruction 's=s+x', on utilise une "boucle", une instruction particulière qui signifie qu'il faut répéter le paquet d'instruction situé entre le début et la fin de la boucle. Ainsi, le 'programme' précédent peut aussi s'écrire

```
x = 1 , s = 0
boucle , i va de 1 à 3
  | s = s + x
fin de boucle
afficher s
```

en ajoutant une instruction d'affichage à l'écran, d'écriture. Le résultat est bien entendu $s = 3$ comme dans le premier cas.

- Une instruction usuelle de boucle propose d'introduire un <u>indice</u> entier noté "i" dans l'exemple précédent, qui prend des valeurs successives allant <u>dans cet ordre</u> de 1 à 3.

- Si l'on veut calculer (avec la précision numérique permise par la machine, voir la première leçon) la somme de la série géométrique

$$S = \sum_{k=1}^{\infty} \left(\frac{1}{4}\right)^k \tag{3}$$

on va obtenir un résultat non banal. Nous retenons d'abord les <u>deux</u> principes de base de l'Art du calcul scientifique

$\boxed{1}$ L'ordinateur calcule toujours faux !

$\boxed{2}$ On teste un algorithme pour un problème que l'on connaît déjà

On commence, en vertu de ce second principe, par calculer <u>exactement</u> le nombre S, qui vaut (exercice !) $S = 1/3$. De cette façon, la connaissance <u>explicite</u> du résultat à obtenir permet de relever très simplement des erreurs de programmation (on en commet toujours !) qui conduiraient à un résultat inexact.

• Pour évaluer approximativement S (relation (3)), on introduit la suite géométrique $(y_k)_{k\in\mathbb{N}}$, telle que

(4) $y_0 = 1, \quad y_{k+1} = \frac{1}{4} y_k \; , \; k\in\mathbb{N}$

et S "à l'itération N" représente la somme

(5) $S_N = y_1 + \dots + y_N = S_{N-1} + y_N$.

Il est commode de poser S_0, ainsi $S_1 = S_0 + y_1$. On affecte alors les valeurs successives de la suite géométrique $(y_k)_{k\in\mathbb{N}}$ à une 'variable' informatique notée y et les valeurs successives de la somme (partielle) de la série géométrique associée $(S_N)_{N\in\mathbb{N}}$ à une autre variable notée S. On initialise l'algorithme par les instructions

$y = 1 \; , \; S = 0$.

Puis on effectue "un certain nombre de fois" l'opération qui consiste à calculer le nouveau terme y de la suite géométrique (cf (4))

$y = y/4$

puis à ajouter le résultat obtenu à la somme partielle antérieure (relation (5)) :

$S = S + y$.

Le 'programme' permettant de calculer la somme donnée en (3) s'écrit donc (en langage 'octave' par exemple)

```
y = 1, S = 0 , n = 26
for i = 1: n
   y = y/4 ; S = S + y ;
end; S
```

et se termine avec l'affichage de la somme S, qui doit être une excellente (2) approximation de 1/3.

- Attention ! L'ordre des instructions est impératif ! Un programme tel que

```
y = 1, S = 0, n = 26
[ boucle par rapport à i allant de 1 à 26
[   S = S + y , y = y/4
```

donnera un résultat différent du programme du haut de cette page. Lequel ? Pour répondre à cette question, il faut exécuter soi-même, "à la main" les diverses instructions du programme, en suivant avec soin le contenu en mémoire des variables 'y' et 'S'.

- Exercice La suite d'instructions :

```
y = 1, S = 1, n = 20
[ boucle % i : 1 → n
[   y = y/i , S = S + y
```

permet de calculer le nombre $e \equiv \sum_{k=0}^{\infty} \frac{1}{k!}$. Pourquoi ?

- <u>Application</u>. Calcul d'intégrales

Pour évaluer une intégrale telle que $\int_a^b f(t)\,dt$, on peut faire un calcul approché en utilisant une <u>somme de Riemann</u> d'ordre N. On divise l'intervalle $[a,b]$ en N intervalles égaux, on pose

$$(6) \quad x_j = a + (j-1)\frac{b-a}{N}, \quad 1 \leq j \leq N+1$$

et on fixe $\xi_{j+1/2} \in [x_j, x_{j+1}]$ pour $1 \leq j \leq N$.

On a alors

$$(7) \quad \int_a^b f(t)\,dt = \lim_{N \nearrow \infty} \frac{1}{N} \sum_{j=1}^{N} f(\xi_{j+1/2})$$

si $f(\cdot)$ est intégrable au sens de Riemann. Le calcul du membre de droite de la relation (7) peut s'effectuer de la façon suivante, en prenant successivement $\xi_{j+1/2} = x_j$ puis $\xi_{j+1/2} = x_{j+1}$ pour fixer les idées.

$N = 1000$, $dx = 1/N$, $S = 0$, $Z = 0$,

boucle sur j : $1 \to N$
$S = S + dx * f\left(\frac{j-1}{N}\right)$
$Z = Z + dx * f\left(\frac{j}{N}\right)$
fin de boucle.

Il convient bien entendu de <u>préciser</u> ce que vaut la fonction $f(\cdot)$, quitte à faire appel à un autre programme (ou sous programme) qui calcule $[0,1] \ni x \mapsto f(x) \in \mathbb{R}$ pour les valeurs de l'argument x.

③ Séries divergentes

- Quand on se donne une suite $(u_n)_{n\in\mathbb{N}}$ de nombres positifs tels que $u_n \to 0$ si $n\to\infty$, il est naturel d'essayer d'évaluer la somme $\sum_{k=0}^{\infty} u_k$, c'est à dire la limite pour $N\nearrow\infty$ de la somme partielle S_N donnée par

$$(8) \qquad S_N = \sum_{k=0}^{N} u_k\,, \quad u_k \geq 0,\ u_k\to 0 \text{ si } k\to\infty.$$

- Il n'est pas évident a priori de savoir si $\sum_0^\infty u_k$ représente un nombre, ou bien si $\sum_0^N u_k = S_N$, qui est une suite croissante avec N si N croît (puisque $u_N \geq 0$) tend vers une limite réelle finie, ou au contraire devient de plus en plus grand lorsque N croît vers l'infini. Dans le premier cas, on dit que la série $\sum_0^\infty u_k$ converge et dans le second, qu'elle diverge. Le calcul numérique permet de découvrir expérimentalement qu'une série telle que (8) converge. Après un nombre fini (qui peut-être grand !) d'étapes, on a S_N qui devient une suite stationnaire si N continue à croître, et l'expérience numérique indique alors une convergence.
- Mais si $S_N \nearrow \infty$ si $N\nearrow\infty$, la suite S_N est de plus en plus grande si $N\nearrow\infty$, on peut toujours espérer

qu'elle va finir par converger, et ce n'est **pas** l'expérience numérique qui permet de conclure que $S_N \uparrow \infty$ si $N \nearrow \infty$. C'est l'analyse mathématique (abstraite) du problème.

- Considérons par exemple S_N définie par

(9) $$S_N = \sum_{k=1}^{N} \frac{1}{k}, \quad N \geq 1.$$

Le terme général de la série, u_k, tend vers 0 ; l'expérience numérique indique

$S_{10} = 2,929$; $S_{100} = 5,187$; $S_{1000} = 7,485$; $S_{10000} = 9,78$

soit une croissance lente. Comment prouver que $S_N \nearrow \infty$ si $N \uparrow \infty$?

- On remarque que $S_{2N} - S_N = \sum_{N+1}^{2N} \frac{1}{k} \geq N \times \frac{1}{2N}$, soit $S_{2N} - S_N \geq \frac{1}{2}$. On a alors

$$\begin{aligned}
S_2 - S_1 &\geq \frac{1}{2} \\
S_4 - S_2 &\geq \frac{1}{2} \\
S_8 - S_4 &\geq \frac{1}{2} \\
&\vdots \\
S_{2^k} - S_{2^{k-1}} &\geq \frac{1}{2}
\end{aligned}$$

donc par sommation

(10) $$S_{2^k} \geq \frac{k}{2} + S_1, \quad k \text{ entier} \geq 0.$$

Si $k \to \infty$, le minorant de la relation (10) tend effectivement vers $+\infty$, mais "lentement" par rapport à l'argument $N = 2^k$ puisque $k \simeq \log_2 N$.

- Le calcul numérique reste un outil qui ne peut **pas** résoudre tous les problèmes !

(4) Branchement

- Nous revenons à l'algorithme de dichotomie, étudié lors de la première leçon. Si $f(\cdot)$ est une fonction continue (la continuité de $f(\cdot)$ est essentielle pour la justesse de la conclusion!) de $[\alpha, \beta]$ dans $\mathbb{R}$ telle que $f(\alpha) < 0$, $f(\beta) > 0$ (pour fixer les idées) alors $\exists \tilde{\gamma} \in]\alpha, \beta[$ tel que $f(\tilde{\gamma}) = 0$.
- L'algorithme de dichotomie pose $\alpha_0 = \alpha$, $\beta_0 = \beta$; puis si $[\alpha_k, \beta_k]$ est donné (k entier ≥ 0), calcule γ_k selon

(11) $\quad \gamma_k = \frac{1}{2}(\alpha_k + \beta_k)$.

Alors ou bien $f(\gamma_k) < 0$, et le théorème des valeurs intermédiaires indique que $\tilde{\gamma} \in]\gamma_k, \beta_k[$, et on pose $\alpha_{k+1} = \gamma_k$, $\beta_{k+1} = \beta_k$

ou bien $f(\gamma_k) \geq 0$ et $\tilde{\gamma} \in]\alpha_k, \gamma_k[$; on pose alors $\alpha_{k+1} = \alpha_k$, $\beta_{k+1} = \gamma_k$.

- La mise en œuvre informatique de l'algorithme de dichotomie consiste à se donner deux variables α, β de sorte que $f(\alpha) < 0$, $f(\beta) > 0$ et à les modifier comme suit

$$\gamma = \frac{1}{2}(\alpha + \beta), \quad z = f(\gamma).$$

Si $z < 0$, $\alpha = \gamma$ et β est inchangé
Sinon, α est inchangé et $\beta = \gamma$
fin du test.

A la fin du test (ou branchement, ou "if" en anglais), on a réduit l'intervalle $[\alpha, \beta]$ de moitié (sous changer le nom des variables, voir le paragraphe 1 au détour de cette leçon !) Dans le cas du calcul approché de $\sqrt{2}$, la programmation de l'algorithme de dichotomie s'effectue comme suit (en suivant la syntaxe du branchement telle que l'impose le logiciel 'octave') :

```
α = 1, β = 2 ,
for i = 1:60
    γ = (α+β)/2 , ρ = γ^2 - 2
if (ρ<0) , α = γ , else
    β = γ , end
end .
```

La boucle en i, qui demande d'itérer l'algorithme de dichotomie encadre le branchement, avec son trait caractéristique

if (propriété vraie) faire xx--
  sinon (else) faire autre chose
fin de branchement (end)

- L'affectation, la boucle et le branchement sont des outils fondamentaux de la programmation du calcul scientifique. Ils permettent d'effectuer l'essentiel des instructions pour les algorithmes à mettre en œuvre.

FD, 12 oct 04.