Université Paris XIII
Institut Galilée
Formation MACS - 2ème année

Année 2006-2007
N. Bereux, S. Ferrand

INITIATION AU FORTRAN 77
Feuille n° 3

Exercice 1 INTÉGRATION NUMÉRIQUE PAR LA MÉTHODE DES TRAPÈZES
Dans cet exercice, on veut calculer numériquement la valeur de l'intégrale d'une fonction C^2

$$x \mapsto f(x),$$

sur un intervalle $[a, b]$.
Méthode des trapèzes : (tiré de [2])
On divise l'intervalle $[a, b]$ en N sous-intervalles

$$[x_i,\, x_{i+1}] \quad i = 0, 1, \ldots, N.$$

Les sous-intervalles sont tous de longueur

$$h = \frac{b-a}{N}.$$

La méthodes des trapèzes est fondée sur la formule :

$$\int_a^b f(x)dx = h\left[\frac{1}{2}f(x_0) + f(x_1) + \ldots + f(x_{N-1}) + \frac{1}{2}f(x_N)\right] + O(\frac{(b-a)^3\, f''}{N^2}).$$

La quantité

$$I_N = \frac{b-a}{N}\left[\frac{1}{2}f(x_0) + f(x_1) + \ldots + f(x_{N-1}) + \frac{1}{2}f(x_N)\right],$$

fournit donc une valeur approchée de l'intégrale $\int_a^b f(x)dx$.
En pratique, on ne connait pas la valeur de N nécessaire à une bonne approximation. On procède donc par approximations successives et on utilise des subdivisions imbriquées de l'intervalle $[a, b]$ (voir Figure 1). De cette façon, les évaluations de f ne sont pas perdues lorsqu'on reprend le calcul avec une subdivision plus fine.
On appelle *approximation de niveau k*, l'approximation I_{2^k}, fondée sur une subdivision de l'intervalle $[a, b]$ en 2^k sous-intervalles.
Ainsi, l'approximation de niveau 0 est calculée par :

$$I_1 = \frac{(b-a)}{2}\left[f(x_0) + f(x_1)\right].$$

Supposons que l'on a calculé $I_{2^{k-1}}$. Alors, l'approximation de niveau k est donnée par :

$$I_{2^k} = \frac{1}{2}\, I_{2^{k-1}} + \frac{b-a}{2^k}\left[f(x_1) + f(x_3) + \ldots + f(x_{2^k-1}))\right].$$

On voit qu'on réutilise les calculs faits au niveau $k-1$ pour obtenir l'approximation de niveau k.
On arrête les calculs lorsque :

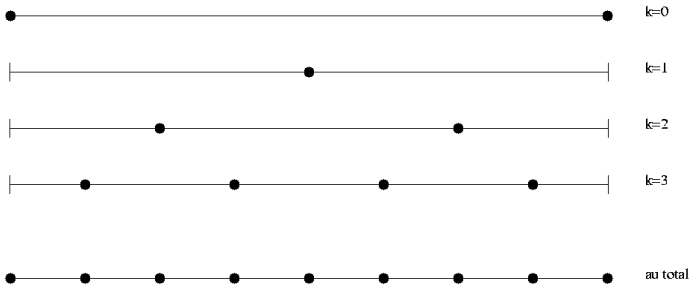

FIG. 1 – Les • représentent pour chaque niveau les nouveaux points où f est évaluée. La dernière ligne montre tous les points où f a été évaluée, pour obtenir une approximation de niveau 3.

- l'erreur relative entre deux approximations successives est petite (inférieure à $\varepsilon = 10^{-4}$) et $k \geq 2$,
- ou bien lorsqu'on a atteint le nombre maximal de niveaux ($k = 20$).

Programmation :

1. Écrire une fonction `integrande.f` qui prend en entrée un réel x et retourne la valeur de $f(x)$ (choisir par exemple $f(x) = x^2$).
2. Écrire une routine `trap.f` qui
   - prend en entrée le niveau k, les bornes a et b de l'intervalle d'intégration, et la dernière approximation calculée de l'intégrale, c'est à dire l'approximation de niveau $k-1$,
   - retourne à la place l'approximation de niveau k.
3. Écrire un programme principal `integ_trapeze.f` qui lit au clavier les bornes a et b de l'intervalle d'intégration, calcule une approximation de l'intégrale de la fonction et affiche la valeur trouvée ainsi que le nombre de niveaux nécessaires pour l'obtenir.

Exercice 2 OPÉRATIONS ÉLÉMENTAIRES D'ALGÈBRE LINÉAIRE SUR DES VECTEURS :

1. Créer avec un éditeur de texte deux fichiers `vec1.dat` et `vec2.dat`. Chaque fichier contient un vecteur colonne x de taille n sous le format suivant :

$$\begin{array}{l} n \\ x(1) \\ \vdots \\ x(n) \end{array} \tag{1}$$

   On choisira par exemple les vecteurs

$$(1.0\ 2.0\ 3.0) \quad (1.0\ 1.0\ 1.0)$$

   pour pouvoir valider facilement les routines.
2. Écrire une routine `lit_vec.f` qui relit un vecteur depuis un fichier du disque (identifié par un numéro d'unité logique).
3. Écrire une routine `affiche_vec.f` qui affiche à l'écran un vecteur.

4. Écrire un programme principal qui appelle ces deux routines et les tester sur les vecteurs vec1 et vec2.
5. Écrire une routine `comb_lin.f` qui prend en entrée un réel α, et deux vecteurs x et y et renvoie en sortie $\alpha\, x + y$ à la place de y, α et x étant inchangés.
6. Écrire une fonction `prod_scal.f` qui calcule le produit scalaire de x par y.
7. Écrire une fonction `norm2.f` qui calcule la norme L_2 d'un vecteur x :

$$\texttt{norm2} = (\sum_{i=1}^{n} x(i)^2)^{1/2}.$$

Ne pas oublier de tester chaque routine au fur et à mesure !

Exercice 3 Counting Sort : un algorithme de tri linéaire pour certains tableaux d'entiers

L'algorithme : (tiré de [1])

Soit *tab* un tableau de N entiers naturels. Les valeurs de *tab* sont comprises entre 1 et un entier $k = \max_{1 \leq i \leq N} tab(i)$.

Lorsque $k = O(N)$, l'algorithme du Counting Sort a une complexité de $O(N)$, inférieure à celle du Quick Sort (en moyenne $O(n \log n)$).

L'idée de base de l'algorithme est que si l'on sait déterminer pour un élément du tableau x combien d'éléments du tableau lui sont inférieurs, alors on sait où placer cet élément dans le tableau trié : en effet si 17 éléments du tableau sont inférieurs à x alors on place x à la $18^{ème}$ position dans le tableau trié.

Pour effectuer le tri on utilise trois tableaux :

- *tab* : le tableau à trier,
- *tab_trie* : le tableau trié
- *count* : un tableau auxiliaire de taille k.

L'algorithme du Counting-sort s'écrit :

```
CSORT(tab, tab_trie, count)
 1  for i ← 1 to k
 2       do
 3            ▷ Initialisation de count
 4            count(i) = 0
 5  for j ← 1 to N
 6       do
 7            ▷ count(i) est le nombre de termes de tab égaux à i
 8            count(tab(j)) ← count(tab(j)) + 1
 9  for j ← 2 to k
10       do
11            ▷ count(i) est le nombre de termes de tab ≤ i
12            count(i) ← count(i) + count(i − 1)
13  for j ← N downto 1
14       do
15            ▷ Insertion de tab(j) dans le tableau trié
16            tab_trie(count(tab(j)) ← tab(j)
17            ▷ On incrémente count(tab(j))
18            count(tab(j)) ← count(tab(j)) − 1
```

Remarques :

- L'incrémentation de $count(tab(j))$ à la ligne 18 permet de traiter les cas où le même entier apparaît plusieurs fois dans le tableau à trier.
- La boucle de la ligne 13 est décroissante pour que le tri soit stable, *i.e.* deux nombres égaux apparaissent dans le même ordre dans *tab* et dans *tab_trie*.

Application de l'algorithme :
Appliquer l'algorithme au tri du tableau (3 6 1 3) pour illustrer son fonctionnement.

Programmation :

1. Créer un fichier `tableau.dat` contenant le tableau

   (3 6 4 1 3 4 1 4)

   stocké avec le format défini par (1).
2. Écrire un programme principal `count_sort.f` qui
   – appelle une routine de lecture depuis le disque d'un tableau,
   – affiche à l'écran le tableau ainsi relu.
   Peut-on réutiliser les routines `lit_vec.f` et `affiche_vec.f` écrites pour l'exercice 2 ?
3. Écrire la routine de tri `csort.f`. Penser à tester que le plus grand élément du tableau à trier est bien inférieur à la taille du tableau *count*.

Références

[1] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *An Introduction to Algorithms*. The MIT Press, 2001.

[2] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical Recipes in Fortran 77, Second Edition*. Cambridge University Press, 1992.