dimanche 9 mai 2010

TP 5 d'analyse numérique

Interpolations

Le but de ce TP est d'interpoler une fonction qui n'est connue qu'en un nombre de points $\{(x_i, y_i)\ ,\ i = 0, \ldots, n\}$ (ces points sont souvent contenus dans des fichiers à plusieurs colonnes).

L'interpolation consiste à remplacer ces points par une certaine fonction connue $y = f(x)$, le choix de f étant dicté par des critères physiques ou autres. Dans le cas où aucun critère n'est disponible, nous choisirons pour f la forme la plus simple, par exemple un polynôme $P_n(x)$ de degré inférieur ou égal à n qui vérifie $P_n(x_i) = y_i$: c'est l'interpolation polynômiale.

Ce TP est long, et est divisé en plusieurs parties. Patience !

PARTIE I.

1. Ecrire en C une fonction qui calcule le minimum ainsi que le maximum d'un ensemble de nombres réels contenu dans le tableau `A[i] (i=0, ... , n)`. Vous appellerez cette fonction "`minmax`" et elle devra avoir 4 arguments, comme l'exemple suivant :

```
int n=15; double A[n+1]={ ... } ; double xmin,xmax;
minmax( A, n , &xmin, &xmax);
```

Remarquez que les 2 derniers arguments sont des pointeurs.

2. Ecrire en C une fonction qui calcule de manière optimisée $P_n(x) = \sum_{i=0}^{n} A_i\, x^i$, les A_i étant supposés connus. Vous appellerez cette fonction "`polynome`" et elle devra avoir 3 arguments comme ceci :

```
polynome(A,n,x);
          /* A (tableau de doubles), n (entier>0), x (double) */
```

3. Ecrire en C une fonction qui calcule de manière optimisée la valeur du polynôme de Legendre $P_n(x)$ au point x. Rappel :

$$P_n(x) = \sum_{i=0}^{n} Y_i \prod_{\substack{j=0 \\ (j\neq i)}}^{n} \frac{x - X_j}{X_i - X_j}$$

Vous appellerez cette fonction "`polynome_legendre`" et elle devra avoir 4 arguments comme ceci :

```
polynome_legendre(X,Y,n,x);
          /* X,Y (tableaux), n (entier>0), x (double) */
```

4. Ecrire un programme en C qui teste chacune des fonctions "`minmax`", "`polynome`" et "`polynome_legendre`", définies précédemment. Vous devrez vous-mêmes déclarer,

définir, initialiser et/ou générer les différents tableaux (et peut-être les afficher) ; voici un exemple pour générer des valeurs :

```
n=200 ;
for(i=0; i<=n ; i++) {
      x = - 10.  + 20.*(double)i/(double)n ;
      X[i] = exp(x*x) - x ;
      Y[i] = sqrt(3*x*x + 2*x) ;
      A[i] = 2.*x*x*x/(x*x + 16) ;
}
```

5. Ecrire en C une fonction qui calcule la table des différences divisées correspondant aux $n+1$ points $\{(x_i, y_i)\ ,\ i = 0, \ldots, n\}$ supposés connus. Cette fonction sera par exemple utilisée dans un programme comme ceci :

```
differences_divisees(x, y, B, n) ;
```

`n`, `x`, `y` et `B` sont supposés avoir été declarés (et peut-être initialisés) convenablement. Exemple :

```
int n=15; double x[n+1], y[n+1], B[(n+1)*(n+1)] ;
x[0]  = ...  ; y[0]  = ...  ;
x[1]  = ...  ; y[1]  = ...  ;
.............................
x[n]  = ...  ; y[n]  = ...  ;
differences_divisees(x, y, B, n) ;
     /* Mettre les résultats dans B(i,j) = B[(i)*n+(j)] */
```

Nous verrons plus loin comment lire des donnés à partir d'un fichier contenant plusieurs colonnes de nombres réels enregistrés.

Rappel :

Etant donné un ensemble de $n+1$ points $\{(X_i, Y_i)\ ,\ i = 0, \ldots, n\}$, il existe un seul polynôme $P_n(x)$ de degré égal ou inférieur à n, qui vérifie

$$P_n(X_i) = Y_i \quad \text{pour} \quad i = 0, \ldots, n. \tag{1}$$

Le polynôme $P_n(x)$, étant unique, peut s'écrire sous la forme (1 ère formule d'interpolation de Newton)

$$\begin{aligned} P_n(x) = a_0 \ & + a_1\,(x - X_0) \\ & + a_2\,(x - X_0)(x - X_1) \\ & + \ \ldots \\ & + a_n\,(x - X_0)(x - X_1)\ldots(x - X_{n-1}) \end{aligned} \tag{2}$$

où les coefficients a_i sont déterminés par le système d'équations (1). La solution s'écrit

$$a_i \equiv [Y_0, Y_1, \ldots, Y_i] \qquad \text{avec} \qquad a_0 \equiv [Y_0, Y_0] \equiv [Y_0] \equiv Y_0. \tag{3}$$

Les expressions $[Y_i, Y_{i+1}, \ldots, Y_{j-1}, Y_j]$, avec $i \leq j$, sont appelées les différences divisées, et nous les noterons ici B_{ij} puisqu'elles ne dépendent que du premier Y_i et du dernier Y_j (les Y_k formant une différence divisée sont toujours écrits dans un ordre où k est croissant : $k = i, i+1, \ldots, j-1, j$). Le calcul des B_{ij} devient simple grâce à la relation de récurrence suivante :

$$[Y_i, Y_{i+1}, \ldots, Y_{j-1}, Y_j] = \frac{[Y_{i+1}, \ldots, Y_{j-1}, Y_j] - [Y_i, Y_{i+1}, \ldots, Y_{j-1}]}{X_j - X_i} \quad \text{avec} \quad [Y_i, Y_i] \equiv Y_i$$

que l'on ré-écrit simplement comme :

$$B_{ij} = \frac{B_{i+1,j} - B_{i,j-1}}{X_j - X_i} \qquad \text{avec} \qquad B_{ii} \equiv Y_i \tag{4}$$

Cette notation (B une matrice triangulaire supérieure car $i \leq j$) est très simple à utiliser dans un programme écrit par exemple en C. Le résultat pour les a_i est donc $a_i = B_{0i}$.

Aide : Les B_{ij} doivent être calculés dans un certain ordre, comme cela a été fait en cours :

- *On calcule d'abord les $B_{ii} = [Y_i, Y_i] \equiv Y_i$ pour $i = 0, \ldots, n$*
- *On calcule ensuite les $B_{i,i+1} = [Y_i, Y_{i+1}]$ pour $i = 0, \ldots, n-1$ (en utilisant Eq. (4))*
- *etc ...*
- *L'avant dernière étape est de calculer les $B_{i,i+(n-1)} = [Y_i, Y_{i+(n-1)}]$ pour $i = 0, \ldots, 1$*
- *On calcule finalement les $B_{i,i+n} = [Y_i, Y_{i+n}]$ pour $i = 0$ (une seule valeur $B_{0,n}$).*

L'algorithme est donc :

1) Pour $i = 0, \ldots, n$, faire : $B_{ii} = Y_i$

2) Pour $m = 1, \ldots, n$, faire :
$$\begin{cases} \text{Pour } \; i = 0, \ldots, n-m, \;\; \text{faire :} \\ \qquad \begin{cases} j = i + m; \\ B_{ij} = \dfrac{B_{i+1,j} - B_{i,j-1}}{X_j - X_i} \end{cases} \end{cases}$$

---

6. Ecrire en C une fonction qui calcule $P_n(x)$ (Eq. (2)). Cette fonction sera par exemple utilisée dans un programme comme ceci :

```
polynome_newton1(X,B,n,x) ;
```

où `x` est la variable, et les tableaux `X[]` et `B[]` sont supposés connus, `B` étant la table des différences divisées correspondant aux $n+1$ points $\{(X_i, Y_i)\ ,\ i = 0, \ldots, n\}$.

7. Ecrire un programme qui compare les 201 valeurs $P_n(x)$ obtenues avec les polynômes de Legendre et Newton correspondant aux $n+1$ points $\{(X_i, Y_i)\ ,\ i = 0, \ldots, n\}$. Vous inventerez vous-mêmes ces points pour $n = 50$.

Note : vous êtes supposés trouver une différence égale à zéro puisque le polynôme $P_n(x)$ est unique ; mais à cause des erreurs d'arrondis, vous trouverez des valeurs proches de zéro pour ces différences.

Exemple de comment calculer et afficher ces différences :

```
n=50 ;
for(j=0; j<=200; j++) {
      x= X[0] + (double)j * (X[n]-X[0])/200.  :
      y= polynome_legendre(X,Y,n,x) - polynome_newton1(X,B,n,x) :
      printf( "j=%3d       diff = %20.16g\n" , j , y );
}
```

PARTIE II. (lecture de données à partir de fichiers)

Les données sont souvent contenues dans des fichiers formatés (chiffres décimaux sur plusieurs colonnes), et nous devons lire ces données, les convertir et les mettre dans des tableaux que nous aurons déclarés convenablement. Un tel fichier ressemblera à ceci :

```
# Exemple avec 6 points (x_i,y_i) i =0,...,5 que l'on représentera avec knuplot
# et que l'on comparera avec le polynome d'interpolation de Newton (1ère formule).
# La 3ème colonne représente les erreurs sur y_i (Dy_i).
1.1       12.1       1.21
1.15       1.5       0.15
1.2       -2.5       0.125

# Cette ligne est un commentaire que l'on négligera lors de la lecture

1.3       -16.35      0.8
1.45      0.13      0.06
1.65      18.34      1.8
```

Comment faut-il faire pour lire uniquement les nombres (ignorer les lignes vides ainsi que celles qui ne contiennent que des commentaires), puis convertir ces nombres en binaire, et ensuite les enregistrer dans divers tableaux (chaque colonne correspondant à un certain tableau en C) ? Une fois que nous avons ces nombres dans des tableaux, nous pouvons les manipuler comme nous voulons, à condition de savoir exactement combien de nombres nous avons lus et convertis.

Une réponse simple à cette question est de lire ce fichier une ligne à la fois (caractère par caractère), mettre ces caractères dans un certain tableau "char sss[1000]" que l'on aura auparavant déclaré en supposant que chaque ligne contient moins de 1000 caractères. Ensuite, nous inspectons ce tableau (en anglais, on dit "we scan") pour voir s'il contient des nombres ou pas. Si le tableau contient des nombres, nous les mettons dans nos tableaux (et nous incrémentons notre compteur pour savoir combien de nombres nous avons lus jusqu'ici). Sinon, nous négligeons cette ligne, et répétons ce processus sur la ligne suivante. Ce processus s'arrêtera lorsque nous aurons atteint la fin du fichier.