Programmation pour le calcul scientifique
Année : 2019-2020
Formation : L3 Ingénierie Mathématique

TP6 : Récapitulatif

Ce TP est noté ! Vous pouvez consulter vos notes de cours ainsi que des polycopiés de cours.

Vous écrirez vos programmes en C++. La lisibilité du code et sa compréhension seront évaluées.

Une question sera considérée comme traitée si les intructions correspondantes, mais également l'affichage compréhensible des résultats de ces instructions, ont été programmés, sont compilables et fonctionnent à l'exécution. Cela signifie que tous les programmes doivent être validés par un test même si ce n'est pas demandé explicitement dans la question.

Si vous avez traité certaines questions mais que le code résultant ne produit pas les résultats escomptés, rajoutez à côté des lignes de code correspondantes des commentaires pour décrire le problème rencontré. Chaque exercice sera traité dans un programme indépendant.

L'exercice 1 sera traité dans un seul fichier. Les deux exercices suivants seront structuré en trois fichiers : un fichier contenant le programme principal, un autre les fonctions associées et un troisième (header) les éventuelles classes et prototypes des fonctions.

Pour les exercices 2 et 3, vous fournirez un Makefile qui permet de compiler vos programmes. Vous pourrez rendre deux fichiers Makefile renommés ou un seul fichier comprenant les cibles appropriées.

Vous déposerez vos programmes sur Moodle sous forme d'archive `tar.gz` que vous pouvez générer avec la commande suivante :

```
$ tar czf mleguebeTP6.tar.gz monFichier1 monDossier2 monFichier3 ...
```

Le nom de l'archive comprendra votre identifiant UB suivi de `TP6.tar.gz`. N'hésitez pas à indiquer aussi vos noms dans vos fichiers. Enfin, passez voir le surveillant pour vous assurer de la bonne réception.

1 Pointeurs, allocation dynamique

Dans un programme,

1. Créez un pointeur sur un entier et modifiez la valeur de cet entier en n'utilisant que le pointeur.
2. Écrivez une fonction qui ne retourne rien mais change le signe d'un entier passé en argument.
3. Définissez un type structuré `triangle` contenant six réels correspondant aux coordonnées 2D des sommets.
4. Sans utiliser la classe `std::vector`, allouez un tableau de 10 triangles.
5. Désallouez le tableau de triangles que vous avez créé.
6. Corrigez le(s) problème(s) que présente cette fonction censée effectuer une rotation des éléments d'un tableau d'un cran vers la gauche :

```
// Performs a rotation of pointers of 1 element to the left
void shift(std::vector<double*>* data) {
  for (uint i=0;i<data.size()-1;i++)
    data[i] = data[i+1];
  data[data.size()-1] = data[0];
}
```

2 Différences finies compactes

En général, les méthodes de différences finies vues en licence pour évaluer une dérivée sont toutes explicites. On obtient directement les valeurs de la dérivée à partir des seules valeurs de la fonction et de simples additions et multiplications.

Par exemple, pour une fonction régulière f :

$$f'(x) \sim \frac{f(x+\delta x) - f(x-\delta x)}{2\delta x}.$$

Les différences finies compactes se basent sur la résolution d'un système linéaire pour évaluer plus précisément cette dérivée.

On se place dans l'intervalle $[0,1]$ discrétisé par N points x_i ($x_i = ih$ où $h = 1/(N-1)$, $x_0 = 0$, $x_{N-1} = 1$). On souhaite calculer la dérivée d'une fonction f en chaque point de discrétisation. Si f_i désigne la valeur donnée de f en x_i et f'_i l'approximation de sa dérivée en ce même point, la relation suivante

$$\frac{1}{3}f'_{i-1} + f'_i + \frac{1}{3}f'_{i+1} = \frac{14}{9}\,\frac{f_{i+1} - f_{i-1}}{2h} + \frac{1}{9}\,\frac{f_{i+2} - f_{i-2}}{4h}$$

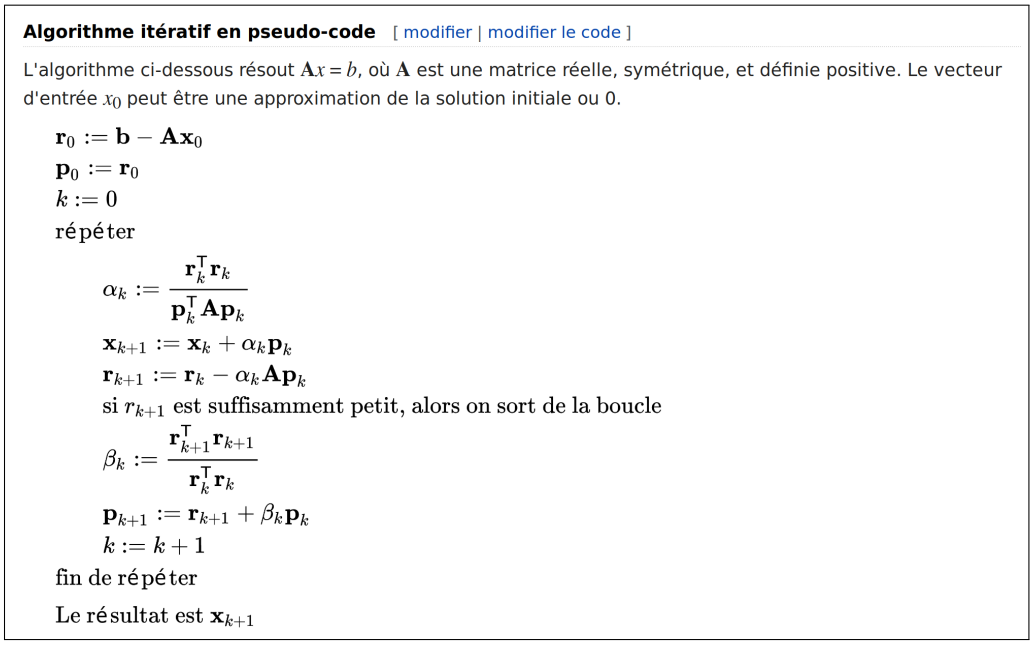
permet d'approcher f' à l'ordre 6 pour i compris entre 2 et $N-3$. Cette relation peut se mettre sous la forme $AF' = BF$ où F' et F sont les vecteurs formés par les valeurs f'_i et f_i, respectivement.

Pour les points au bord :

$$\begin{aligned}
i = 0, &\quad f'_0 + 2f'_1 = \frac{1}{h}\left(-\frac{5}{2}f_0 + 2f_1 + \frac{1}{2}f_2\right),\\
i = 1, &\quad \frac{1}{4}f'_0 + f'_1 + \frac{1}{4}f'_2 = \frac{3}{4h}(f_2 - f_0),\\
i = N-2, &\quad \frac{1}{4}f'_{N-3} + f'_{N-2} + \frac{1}{4}f'_{N-1} = \frac{3}{4h}(f_{N-1} - f_{N-3}),\\
i = N-1, &\quad 2f'_{N-2} + f'_{N-1} = \frac{1}{h}\left(-\frac{1}{2}f_{N-3} - 2f_{N-2} + \frac{5}{2}f_{N-1}\right).
\end{aligned}$$

Ces relations sont à l'ordre 4 pour les lignes $i = 1$ et $i = N-2$, et à l'ordre 3 pour les lignes $i = 0$ et $i = N-1$.

1. Pour chacune des matrices A et B, écrire une fonction qui *retourne* cette matrice. Vous tiendrez bien sûr compte de la nature de ces matrices pour leur stockage.
2. Écrire une fonction `matMul` qui calcule le produit d'une matrice creuse par un vecteur dense.
3. Écrire une fonction `dotProd` qui retourne le produit scalaire $(u, v) = u^T v$ de deux vecteurs denses u et v.
4. Écrire une fonction `solveCG` qui résoud le système $Ax = B$ où A est creuse symmétrique définie positive par une méthode de gradient conjugué (cf algorithme en figure 1)
5. Pour $N = 100$, utilisez les différences finies compactes pour calculer la dérivée de la gaussienne suivante :
$$f(x) = \exp\left(-\frac{(x-0.5)^2}{0.1}\right).$$
6. Mesurez l'erreur commise en norme infinie sur $[0,1]$.
7. Mesurez cette erreur pour diverses valeurs de N et vérifiez l'ordre de la méthode. Que constatez-vous ? Comment l'expliquez-vous ?



Algorithme itératif en pseudo-code [modifier | modifier le code]

L'algorithme ci-dessous résout $\mathbf{A}x = b$, où $\mathbf{A}$ est une matrice réelle, symétrique, et définie positive. Le vecteur d'entrée x_0 peut être une approximation de la solution initiale ou 0.

$\mathbf{r}_0 := \mathbf{b} - \mathbf{A}\mathbf{x}_0$
$\mathbf{p}_0 := \mathbf{r}_0$
$k := 0$
répéter
  $\alpha_k := \frac{\mathbf{r}_k^\mathsf{T}\mathbf{r}_k}{\mathbf{p}_k^\mathsf{T}\mathbf{A}\mathbf{p}_k}$
  $\mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k\mathbf{p}_k$
  $\mathbf{r}_{k+1} := \mathbf{r}_k - \alpha_k\mathbf{A}\mathbf{p}_k$
  si r_{k+1} est suffisamment petit, alors on sort de la boucle
  $\beta_k := \frac{\mathbf{r}_{k+1}^\mathsf{T}\mathbf{r}_{k+1}}{\mathbf{r}_k^\mathsf{T}\mathbf{r}_k}$
  $\mathbf{p}_{k+1} := \mathbf{r}_{k+1} + \beta_k\mathbf{p}_k$
  $k := k + 1$
fin de répéter
Le résultat est $\mathbf{x}_{k+1}$

FIGURE 1 – Algorithme du gradient conjugué. CC-BY-SA 3.0 Wikipedia

3 Classe `Point`

Le but de cet exercice est de créer une classe `Point` de manière plus complète que vue précédemment, avec quelques fonctionnalités supplémentaires.

La classe comprendra

- Un constructeur par défaut qui initialise les coordonnées à 0.
- Un constructeur à partir de deux nombres réels.
- Un constructeur par copie.
- Une fonction d'affichage des coordonnées à l'écran.
- Un opérateur effectuant cet affichage, compatible avec le mode d'affichage standard du C++.
- Une fonction `shift(double tx, double ty)` qui effectue la translation du point par le vecteur de coordonnées (t_x, t_y).
- Une fonction `rotate(double theta)` qui effectue la rotation du point d'un angle θ.
- Une fonction `distance(Point& p2)` qui retourne la distance entre l'instance courante et le point P_2.

Vous écrirez également une fonction externe à la classe, nommée `midPoint(Point& p1, Point& p2)` qui retourne une instance de `Point` qui est le milieu de deux points donnés en argument.