Université Pierre et Marie Curie
Master 2 IFMA
Programmation en `C++`

Projet (à rendre pour le 18 décembre 2015)

Le but de ce projet est de résoudre numériquement l'équation parabolique suivante :

$$\begin{cases} \partial_t u_t(x) = \partial_x(c_x \partial_x u_t(x)) & t > 0, 0 < x < 1, \\ u(0,x) = u_0(x) & 0 < x < 1, \\ u_t(0) = 0 & t \geq 0, \\ u_t(1) = 0 & t \geq 0. \end{cases} \tag{1}$$

Ici, u_0 est une condition initiale et la fonction $x \mapsto c_x$ est une fonction de $[0,1]$ dans $\mathbf{R}$, appelée coefficient de diffusion. L'équation (1) modélise l'évolution de la température d'une barre de métal dont les extrémités sont maintenues à température constante.

On a vu précédemment que la résolution de cette équation par un schéma explicite pouvait être instable avec des pas d'espace trop petits. On va donc utiliser un schéma *implicite*. Plus précisément, des approximations par différences finies permettent de déduire de (1) la relation approchée suivante :

$$\frac{u_t(x) - u_{t-\delta_t}(x)}{\delta_t} \simeq \frac{c_{x+\frac{1}{2}\delta_x} u_t(x+\delta_x) + c_{x-\frac{1}{2}\delta_x} u_t(x-\delta_x) - (c_{x+\frac{1}{2}\delta_x} + c_{x-\frac{1}{2}\delta_x}) u_t(x)}{\delta_x^2}.$$

En séparant dans l'équation précédente les termes en u_t et les termes en $u_{t-\delta_t}$, on obtient la relation suivante, exprimant $u_{t-\delta_t}$ en fonction de u_t :

$$u_{t-\delta_t}(x) \simeq u_t(x)\left(1 + (c_{x+\frac{1}{2}\delta_x} + c_{x-\frac{1}{2}\delta_x})\frac{\delta_t}{\delta_x^2}\right) - \frac{\delta_t}{\delta_x^2} c_{x+\frac{1}{2}\delta_x} u_t(x+\delta_x) - \frac{\delta_t}{\delta_x^2} c_{x-\frac{1}{2}\delta_x} u_t(x-\delta_x). \tag{2}$$

On cherche à approcher $(u_t(x))_{t\geq 0, x\in[0,1]}$ par une famille discrète de coefficients notés $(u_k^n)_{n\in\{0,\dots,N\},k\geq 0}$ où $u_k^n \simeq u_{k\delta_t}(n\delta_x)$. Ici $\delta_t > 0$ est le pas de discrétisation en temps et $\delta_x = 1/N$ est le pas de discrétisation en espace. Au vu de l'équation (2), il est naturel de chercher une famille (u_k^n) satisfaisant la relation

$$\begin{cases} u_{k-1}^n & = u_k^n(1 + (c_{(n+1/2)\delta_x} + c_{(n-1/2)\delta_x})\frac{\delta_t}{\delta_x^2}) - u_k^{n-1} c_{(n-1/2)\delta_x}\frac{\delta_t}{\delta_x^2} - u_k^{n+1} c_{(n+1/2)\delta_x}\frac{\delta_t}{\delta_x^2}, \\ u_0^n & = u_0(n\delta_x). \end{cases} \tag{3}$$

À première vue, la relation (3) ne définit pas forcément une unique famille (u_k^n). Toutefois, si on note u_k le vecteur $(u_k^n)_{n=1,\dots,N-1}$, l'équation (3) peut se récrire comme

$$\begin{cases} u_{k-1} & = Au_k, \\ u_0 & \text{donné}, \end{cases}$$

où la matrice A est définie par

$$A_{i,j} = \begin{cases} 1 + \frac{\delta_t}{\delta_x^2}(c_{(i+1/2)\delta_x} + c_{(i-1/2)\delta_x}) & \text{si } j = i, \\ -\frac{\delta_t}{\delta_x^2} c_{(i+1/2)\delta_x} & \text{si } j = i+1, \\ -\frac{\delta_t}{\delta_x^2} c_{(i-1/2)\delta_x} & \text{si } j = i-1, \\ 0 & \text{sinon.} \end{cases}$$

La matrice A est symétrique, donc diagonalisable dans une base orthonormée avec des valeurs propres réelles. On peut par ailleurs montrer que, quelles que soient les valeurs de δ_t et δ_x, les valeurs propres de A sont strictement supérieures à 1. Par conséquent, A est inversible, de sorte que la relation (3) admet pour unique solution la suite $u_k = A^{-k}u_0$. De plus, la suite $\|u_k\|_2$ est décroissante.

L'équation (3) définit donc une solution approchée de u qui est stable (grâce à la décroissance de la norme) indépendamment quels que soient les pas de discrétisation. Le prix à payer est que l'itération du schéma demande d'inverser une matrice qui peut être de grande taille si le pas d'espace est petit.

Pour éviter de calculer l'inverse de A, on va calculer sa décomposition de Cholesky $A = LL^T$, de sorte que résoudre l'équation $u_{k-1} = Au_k$ se réduira à trouver les solution de deux systèmes linéaires triangulaires $u_{k-1} = L\tilde{u}_k$ et $\tilde{u}_k = L^T u_k$.

Question 1 :
Dans un ficher `vect.hpp`, déclarer et définir une classe `vect` "*template*", indexée par un paramètre entier n correspondant aux vecteurs de taille n à coefficients de type `double`.

Cette classe devra contenir
- un membre privé de type `double*` correspondant à la liste des n coefficients ;
- un constructeur par défaut initialisant tous les coefficients à 0 ;
- deux surcharges de l'opérateur `[]`, de prototypes `double& operator[](int);` et `double operator[](int) const;` permettant respectivement d'accéder aux coefficients du vecteur en écriture et en lecture ;
- une méthode `norme` renvoyant la norme 2 du vecteur.

Si besoin, on redéfinira le constructeur par copie, le destructeur et l'opérateur `=`.

Question 2 :
Dans un fichier `matrice.hpp`, définir
- une classe `triang` permettant de manipuler des matrices dont seules la diagonale et la première sous-diagonale sont non-nulles (autrement dit $A_{i,j} = 0$ si $i \neq j$ et $i \neq j+1$) ;
- une classe `tridiag` représentant les matrices tridiagonales symétriques (autrement dit $A_{i,i+1} = A_{i+1,i}$ et $A_{i,j} = 0$ si $j > i+1$ ou $j < i-1$).

Plus précisément, ces deux classes devront être "*template*" indexées par un entier `n` correspondant à la taille de la matrice, et contenir :
- comme membres privés, une variable `diag` de type `vect<n>`, correspondant à la liste des coefficients diagonaux de la matrice, et un tableau nommé `sousdiag`, de type `vect<n-1>`, correspondant à la liste des coefficients sur- et sous-diagonaux ;
- un constructeur par défaut initialisant `diag` et `sousdiag` grâce à leurs constructeurs par défaut ;
- deux surchages de l'opérateur `()`, de prototypes `double operator()(int, int) const;` et `double& operator()(int, int);` permettant d'accéder aux coefficients de la matrice respectivement en lecture et en écriture.

Si besoin, on redéfinira le constructeur par copie, le destructeur et l'opérateur `=`.

Question 3 :
Dans la classe `tridiag`, définir une méthode de prototype `triang<n> cholesky(void) const;` qui calcule la décomposition de Cholesky d'une matrice creuse symétrique. On rappelle que pour toute matrice symétrique tridiagonale S, il existe une matrice L triangulaire inférieure telle que $S = LL^T$. De plus, les seuls coefficients non-nuls de L

sont les $L_{i,i}$ et $L_{i,i-1}$. On remarquera que L et S satisfont les relations $L_{1,1}^2 = S_{1,1}$, $L_{i+1,i}L_{i,i} = S_{i+1,i}$ et $L_{i,i-1}^2 + L_{i,i}^2 = S_{i,i}$ $(i > 1)$.

Question 4 :
Implémenter dans la classe `triang` deux méthodes de prototypes respectifs `vect<n> solve(vect<n>&) const;` et `vect<n> solve_T(vect<n>&) const;`, renvoyant respectivement la solution du système linéaire $b = Lx$ et $b = L^T x$, où b est l'argument fourni à la méthode et L est la matrice courante.

Question 5 :
Résoudre numériquement l'équation (1) partant d'un "bruit blanc". Plus précisément, écrire un programme qui affiche la suite $(u_k^n)_{n=0,\dots,N}$ avec k tel que $k\delta_t = 1$, en prenant pour condition initiale des $(u_0^n)_{n=1,\dots,N-1}$ aléatoires indépendants, de loi $\mathcal{N}(0, 1/\delta_x)$.

On choisira $c(x) = 0.01 + \frac{0.1}{1+e^{10-20x}}$.

Question 6 :
Sur le même exemple que la question précédente, illustrer la décroissance de la suite des normes $(\|u_k\|_2)_{k\geq 0}$.