MO102. Manipulations matricielles avancées
Patrick Ciarlet

Pour s'échauffer...

Exercice 1 Soient les vecteurs colonnes et la matrice suivants

$$\vec{u}_1 = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}, \ \vec{u}_2 = \begin{pmatrix} -5 \\ 2 \\ 1 \end{pmatrix}, \ \vec{u}_3 = \begin{pmatrix} -1 \\ -3 \\ 7 \end{pmatrix}, \ \mathbb{A}_0 = \begin{pmatrix} 2 & 3 & 4 \\ 7 & 6 & 5 \\ 2 & 8 & 7 \end{pmatrix}.$$

1. Entrer les données en ligne.
2. Calculer le produit $\mathbb{A}_0\vec{u}_1$.
3. Déterminer les commandes Matlab permettant de :
   (a) calculer $\|\vec{u}_1\|_2$, $\|\vec{u}_2\|_1$, $\|\vec{u}_3\|_\infty$;
   (b) déterminer les dimensions de la matrice $\mathbb{A}_0$, en extraire le nombre de colonnes ;
   (c) calculer le déterminant et l'inverse de $\mathbb{A}_0$.
4. Proposer deux méthodes permettant de résoudre le problème $\mathbb{A}_0\vec{x} = \vec{u}_1$, et déterminer les commandes Matlab associées. Quelle méthode doit-on retenir ?

Corrigé 1 Taper en ligne :

```
u1 = [ 1 ; 2 ; 3 ]
u2 = [ -5 ; 2 ; 1 ]
u3 = [ -1 ; -3 ; 7 ]
A0 = [ 2 3 4 ; 7 6 5 ; 2 8 7 ]
A0*u1
norm(u1,2)
norm(u2,1)
norm(u3,inf)
size(A0)
size(A0,2)
det(A0)
inv(A0)
x = inv(A0)*u1
x = A0\u1
```

Bien sûr, pour résoudre $\mathbb{A}_0\vec{x} = \vec{u}_1$, il faut utiliser **uniquement** la commande `x = A0\u1`...

Pour étudier un exemple plus pratique, on se propose de discrétiser le modèle du fil pesant : *trouver u tel que*

$$-u'' = f \text{ sur }]0,1[, \quad u(0) = u(1) = 0.$$

Si f est de classe $\mathcal{C}^2([0,1])$ (ou, ce qui est équivalent, si u est de classe $\mathcal{C}^4([0,1])$), on peut vérifier que pour $x \in]0,1[$ et h tel que $[x-h, x+h] \in [0,1]$, on peut écrire

$$-u''(x) = \frac{-u(x-h) + 2u(x) - u(x+h)}{h^2} + r(x,h), \text{ avec } |r(x,h)| \leq \left(\sup_{x\in[0,1]} |f''(x)|/12 \right) h^2.$$

Ce résultat *simple* fournit une méthode de discrétisation et d'approximation de l'équation du fil pesant. On parle de *schéma aux différences finies*. Comment procède-t-on ?
Pour commencer, on choisit $N \in \mathbb{N}$, et on fixe $h = \dfrac{1}{N+1}$. Remarquons tout de suite que pour avoir une "bonne" approximation de $u''(x)$, il convient que h soit petit. Ceci signifie que N est un paramètre de discrétisation qui aura vocation à devenir "grand", lors de la réalisation des expériences numériques.
Comment approcher la valeur de u aux points $x_i = ih$, pour $i \in \{0, 1, \cdots, N+1\}$, par des nombres, notés $(u_i)_{0 \leq i \leq N+1}$?
Comme on sait que $u(0) = u(1) = 0$, on choisira comme approximation $u_0 = u_{N+1} = 0$!
Ensuite, on définit $f_i = f(x_i)$, pour $i \in \{1, \cdots, N\}$, et on considère les équations

$$\frac{-u_{i-1} + 2u_i - u_{i+1}}{h^2} = f_i, \ 1 \leq i \leq N, \text{ avec } u_0 = u_{N+1} = 0.$$

Si on appelle $\vec{u}$ (resp. $\vec{f}$) le vecteur de $\mathbb{R}^N$ de composantes $(u_i)_{1 \leq i \leq N}$ (resp. $(f_i)_{1 \leq i \leq N}$), on peut réécrire l'ensemble des équations sous la *forme vectorielle équivalente*

$$\mathbb{A}_1 \, \vec{u} = \vec{f}, \text{ avec } \mathbb{A}_1 = \frac{1}{h^2} \begin{pmatrix} 2 & -1 & 0 & \dots & 0 \\ -1 & 2 & -1 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & -1 \\ 0 & \dots & 0 & -1 & 2 \end{pmatrix} \in \mathbb{R}^{N \times N}.$$

Exercice 2

1. Ecrire une fonction qui construise la matrice $\mathbb{A}_1$ d'ordre N obtenue par la discrétisation par différences finies du modèle du fil pesant.
2. Comparer les temps de construction ainsi que la faisabilité, selon que la structure associée à $\mathbb{A}_1$ est pleine ou creuse, en fonction de N.
3. Résoudre le système linéaire $\mathbb{A}_1 \vec{u} = \vec{f}$, lorsque $f \equiv 1$.

Corrigé 2

1. Dans le fichier `fil.m`, créer la fonction :

```
function A1 = fil(N)
A1 = 2*speye(N,N);
for i = 2:N
  A1(i,i-1) = -1;
end
for i = 1:N-1
  A1(i,i+1) = -1;
end
A1 = (N+1)^2*A1;
```

   On peut examiner la structure de la matrice en tapant `A1 = fil(100); spy(A1)`.

2. On peut créer une fonction `filplein` qui renvoie la même matrice, en *stockage plein*, en remplaçant `A1 = 2*speye(N,N)` par `A1 = 2*eye(N,N)`.
   Pour mesurer le temps de construction, on peut par exemple taper en ligne

```
tic; A1 = fil(1000); toc
tic; A1 = filplein(1000); toc
```

On constate que les temps d'exécution sont comparables pour N petit. Lorsque N devient grand, on remarque tout d'abord que les temps d'exécution en stockage plein deviennent supérieurs à ceux en stockage creux, puis des problèmes de capacité mémoire apparaissent pour le stockage plein !

```
tic; A1 = filplein(50000); toc
??? Error using ==> eye
Out of memory. Product of dimensions is greater than maximum integer.
```

Il serait également possible de mesurer le temps de construction à l'aide de la commande `cputime`.

3. Taper pour commencer `A1 = fil(1000); f=ones(1000,1);`
   Si, pour résoudre $\mathbb{A}_1\vec{u} = \vec{f}$, vous persistez à utiliser la commande `u = inv(A1)*f` au lieu de `u = A1\f`, vous remarquez que des problèmes se posent lorsque N croît...
   Pour visualiser le résultat, taper `plot(u)`.

Et plus si affinités...

Lorsque l'on doit résoudre plusieurs fois des systèmes linéaires avec la même matrice, la question se pose de savoir si l'utilisation de la commande \ est suffisamment rapide... Si tel n'est pas le cas, il est intéressant de réaliser la factorisation de la matrice en question *a priori*, puis d'utiliser cette factorisation pour résoudre très rapidement les systèmes linéaires. Qu'en est-il pour la matrice $\mathbb{A}_1$?
Commençons par noter que la matrice $\mathbb{A}_1$ est *symétrique définie-positive* :

— Une matrice A de $\mathbb{R}^N \times \mathbb{R}^N$ est symétrique lorsque $A_{i,j} = A_{j,i}$, pour $1 \leq i, j \leq N$;
— Une matrice A de $\mathbb{R}^N \times \mathbb{R}^N$ est définie-positive lorsque $(Ax, x)_{\mathbb{R}^N} > 0$, pour tout x non-nul.

Par voie de conséquence, on peut réaliser la factorisation de Cholesky de la matrice A, c'est-à-dire construire une matrice *triangulaire inférieure* L ($L_{i,j} = 0$ si $j > i$) telle que $A = LL^\mathsf{T}$. En remplaçant A par sa factorisation, on peut résoudre le système linéaire

$$A\vec{u} = \vec{f}$$

en deux temps :

— une *descente* : résoudre $L\vec{y} = \vec{f}$;
— une *remontée* : résoudre $L^\mathsf{T}\vec{u} = \vec{y}$.

Ceci est très avantageux, car chacun des deux systèmes fait intervenir une matrice triangulaire, pour laquelle la résolution du système linéaire associé est élémentaire...

NB. Dans le cas général, on utilisera la factorisation de Gauss (fonction `lu`) ; on pourra également considérer des méthodes itératives de résolution (rubrique `iterative methods`).

Exercice 3 1. Ecrire une fonction Matlab pour calculer la solution de $A\vec{u} = \vec{f}$ à l'aide la méthode de Cholesky, en utilisant la fonction `chol`.

2. Comment procéder si on doit résoudre p systèmes linéaires, de seconds membres $f_1, \cdots, f_p$?

Corrigé 3 1. Dans le fichier `SolChol.m`, créér la fonction `SolChol` :

```
function u = SolChol(A,f)
% On suppose que la matrice A est SDP

% 1. Calcul de L triangulaire inferieure telle que A = L.L'
LT = chol(A);
L  = LT';
% 2. Resolution de L.L'u = f, par une descente-remontee
% 2.1 Descente : resolution de Ly = f
y = L\f;
% 2.2 Remontee : resolution de L'u = y
u = LT\y;
```

2. On peut par exemple, dans le fichier `SolCholmult.m`, créér la fonction `SolCholmult` :

```
function u = SolCholmult(A,f)
% On suppose que la matrice A est SDP
```

```
% 1. Calcul de L triangulaire inferieure telle que A = L.L'
LT = chol(A);
L  = LT';

% p Seconds membres stockes sous la forme d'un tableau
[N,p] = size(f);
u = zeros(N,p);
for k = 1:p
% 2. Resolution de L.L'u(k) = f(k), par descente-remontee
% 2.1 Descente : resolution de Ly = f(k)
  y = L\f(:,k);
% 2.2 Remontee : resolution de L'u(k) = y
  u(:,k) = LT\y;
end
```

Outre la résolution de systèmes linéaires, l'autre grande catégorie de problèmes liés aux matrices est le calcul de ses valeurs propres (et de ses vecteurs propres.) Précisons. Soit A une matrice de $\mathbb{C}^{N\times N}$: $\lambda \in \mathbb{C}$ est une valeur propre de A si, et seulement si, l'une des deux assertions ci-dessous est vérifiée :

— $det(A - \lambda\, I_N) = 0$;
— il existe un vecteur x non nul de $\mathbb{R}^N$ satisfaisant à $Ax = \lambda x$.

NB. Il est souvent intéressant de calculer les modes propres d'un système physique. Une fois discrétisé (et après linéarisation éventuelle), on aboutit typiquement au problème mentionné ci-dessus de la détermination des valeurs propres.

Exercice 4 1. En utilisant la fonction `eig`, calculer les valeurs propres de $\mathbb{A}_0$.

2. En utilisant la fonction `eig`, peut-on calculer les valeurs propres de $\mathbb{A}_1$, pour $N = 10.000$?
3. En utilisant la fonction `eigs`, calculer les 5 plus petites valeurs propres de $\mathbb{A}_1$, pour $N = 100, 300, 1.000, 3.000$, etc. Conclusion ?

Corrigé 4 1. Taper en ligne `eig(A0)`.

2. Taper en ligne `eig(fil(10000))` : la réponse est longue à venir... Le calcul de **toutes** les valeurs propres est globalement coûteux.
3. Taper en ligne `eigs(fil(100),5,'sm')`, etc. Peu ou pas d'évolution. Par voie de conséquence, ce sont des "constantes" (indépendantes de N !) du problème discrétisé issu du modèle du fil pesant. La signification physique est que ses nombres correspondent à des modes propres associés au modèle du fil pesant lui-même, c'est-à-dire les solutions de : *trouver* (u, μ) *tel que*

$$-u'' = \mu\, u \text{ sur }]0,1[, \quad u(0) = u(1) = 0.$$

NB. Au contraire, on constate une évolution rapide des plus grandes valeurs propres...