Analyse Numérique

Travaux Pratiques - Série 6

Exercice 1 : Le pivot de Gauss. On aimerait utiliser l'algorithme d'élimination de Gauss avec recherche partielle de pivot pour résoudre le système linéaire

$$Ax = b, \tag{1}$$

où A est une matrice carrée.

1. D'abord, il faut écrire des "solvers" pour résoudre des systèmes triangulaires. Pour un système $Ux = b$, où U est une matrice triangulaire supérieure, utiliser l'en-tête suivante :

```
function y = usol(U,b)
% USOL solves the system Ux = b for upper triangular U
%   y = usol(U,b) solves the system Ux = b, where U is a
%   upper triangular matrix.
```

   Voici l'en-tête pour $Lx = b$, où L est une matrice triangulaire inférieure :

```
function y = lsol(L,b)
% LSOL solves the system Lx = b for lower triangular L
%   y = lsol(L,b) solves the system Lx = b, where L is an
%   lower triangular matrix.
```

   Pour vérifier vos programmes, créer d'abord des matrices triangulaires aléatoires L et U. (Les commandes `rand`, `tril` et `triu` peuvent être utiles.) Ensuite, créer la solution exacte `x_exact` et calculer `b = L*x_exact` et `c = U*x_exact`. Quelle est la différence entre `x_exact` et les résultats de `lsol`/`usol` ?

2. Ensuite, écrire un programme qui calcule la factorisation LU sans recherche partielle de pivot :

```
function [L,U] = LUNoPivot(A)
% LUNOPIVOT computes LU factorization of A without pivoting
%   [L,U] = LUNoPivot(A) computes the LU factorization of A using
%   Gaussian elimination without pivoting. L is a unit lower triangular
%   matrix and U is an upper triangular matrix.
```

   Appliquer à la matrice

$$A = \begin{bmatrix} 4 & -3 & 1 \\ 0.5 & 3 & -1 \\ 0 & 1 & 0 \end{bmatrix},$$

   puis comparer avec les matrices L et U données par la commande Matlab `lu(A)`. Pour la matrice A proposée, la matrice de permutation P est simplement l'identité (Pourquoi ?). Ensuite, utiliser `lsol` et `usol` pour résoudre $Ax = b$ avec un vecteur b aléatoire.

3. Rajouter la recherche partielle de pivot en modifiant le programme ci-dessus (voir section 5.5). Utiliser l'en-tête suivante :

```
function [L,U,p] = LUPivot(A)
% LUPIVOT computes LU factorization of A with partial pivoting
%   [L,U,p] = LUPivot(A) computes the LU factorization of A with partial
%   pivoting. L is a unit lower triangular matrix, U is an upper triangular
%   matrix, and p is a vector of permutations where p(i) is the row of A
%   from which the i-th pivot is selected. In other words, p is computed such
%   that solving A*x = b is equivalent to solving L*U*x = b(p).
```

Pour ce faire, il faut faire très attention lors des échanges des équations ! Les facteurs L et U, ainsi que le vecteur p, doivent être modifiés avec chaque échange. Comment peut-on utiliser `lsol`/`usol` pour résoudre $Ax = b$ dans ce cas général ?

4. Vérifier le programme `LUPivot` sur les exemples ci-dessous :

$$\text{(a)} \quad A = \begin{pmatrix} 1 & 5 & 2 \\ 0 & 0 & 8 \\ 2 & 4 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 9 \\ 8 \\ 9 \end{pmatrix}$$

$$\text{(b)} \quad A = \left(\left(\frac{j}{n} \right)^{i-1} \right)_{i,j=1}^{n} \quad b = \begin{pmatrix} 1 \\ 1/2 \\ \vdots \\ 1/n \end{pmatrix}.$$

Pour $n = 8, 10, 12$, à quelle précision numérique du résultat peut-on s'attendre ?

5. Comparer les résultats obtenus en résolvant $Ax = b$ en utilisant d'abord `LUPivot` puis la commande matlab " \". Qu'observez-vous ? Choisissez n'importe quels matrices A et b.

6. Vérifier que le pivot trouvé par `LUPivot` est le même que celui utilisé par l'algorithme `LUNoPivot` pour la matrice suivante de taille $n \times n$ (matrice impliquée dans le calcul de splines). Comment expliquer cela ?

$$A = \frac{1}{h} \begin{bmatrix} 4 & 1 & & & \\ 1 & 4 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & 4 & 1 \\ & & & 1 & 4 \end{bmatrix}$$

où $h = \frac{1}{n}$, pour certains n grand.

Exercice 2 : La règle de Cramer, une méthode peu efficace numériquement. On souhaite maintenant résoudre le système linéaire (1) en utilisant la règle de Cramer

$$x_i = \frac{\det(A_i)}{\det(A)}, \quad i = 1 \dots n,$$

où A_i est la matrice obtenue en remplaçant la $i$ème colonne de A par le vecteur b. La fonction déterminant det sera calculée récursivement en utilisant le développement de Laplace :

$$\det(A) = \sum_{j=1}^{n} (-1)^{1+j} a_{1,j} \det M_{1,j}$$

où $M_{1,j}$ est la matrice obtenue à partir de la matrice A en supprimant la ligne i et la colonne j (par convention, le déterminant de la matrice vide est égal à 1).

1. *Coût des calculs.*
   (a) Écrivez un programme pour calculer le déterminant par la formule de Laplace et testez-le en utilisant des matrices aléatoires (`rand` en MATLAB) de dimension $n = 3, 5, 7, 9$.
   (b) Écrivez un programme pour calculer $\det(A)$ pour une matrice A raisonnablement grande, par exemple $n \geq 100$, sans utiliser la formule de Laplace.
   (c) Estimer le coût des calculs pour résoudre un système linéaire en utilisant la règle de Cramer.
2. *Stabilité numérique.* Pour une matrice donnée A, vous pouvez suivre les instructions ci-dessous pour tester la stabilité numérique de la résolution de systèmes linéaires.

   (1) Générer une solution "exacte" $x = \texttt{randn}(n, 1)$, et le membre de droite correspondant $b := Ax$.

(2) On appelle $\widehat{x}$ la solution numérique de $Ax = b$ par la règle de Cramer. Calculer l'erreur forward et les normes des résidus :

$$\texttt{err} := \frac{\|\widehat{x} - x\|_2}{\|x\|_2}, \qquad \texttt{res} := \frac{\|A\widehat{x} - b\|_2}{\|A\|_2\|x\|_2 + \|b\|_2}.$$

Pour simplicité, on considère le système linéaire 2×2 $Ax = b$, pour lequel la règle de Cramer devient

$$x_1 = (b_1 a_{22} - b_2 a_{12})/d, \qquad x_2 = (b_2 a_{11} - b_1 a_{21})/d$$

où $d = a_{11}a_{22} - a_{21}a_{12}$.
Utiliser la fonction `matgen` (disponible sur Chamilo) pour générer des matrices A de dimension $n = 2$, et condition $\kappa = 10^2, 10^6, 10^{10}, 10^{18}$, respectivement. Observer les erreurs `err` et `res` pour la règle de Cramer, et comparer le résultat avec l'algorithme de l'élimination de Gauss de l'Exercice 1, ainsi qu'avec la fonction de MATLAB backslash \.

Exercice 3 : Inversion-et-multiplication. Dans cet exercice on examine l'idée de résolution d'un système linéaire $Ax = b$ par inversion directe : d'abord calculer la matrice inverse $M = A^{-1}$, puis appliquer la multiplication matrice-vecteur $x = Mb$.

1. Écrire un programme pour résoudre efficacement $x^{(1)}, x^{(2)}, \ldots, x^{(k)}$ d'un ensemble de k systèmes linéaires
$$Ax^{(i)} = b^{(i)} \qquad \text{for} \qquad i = 1, \ldots, k,$$
où $b^{(i)} \in \mathbb{R}^n$ sont des membres de droite donnés.
Indice : Comme A est la même pour tout les systèmes linéaires, on peut réutiliser sa factorisation LU.
2. Modifier votre programme pour calculer la matrice inverse d'une matrice A.
Indice : A^{-1} est la solution de $AX = I$, où I est une matrice identité.
3. Étudier la stabilité de l'algorithme inversion-et-multiplication comme dans l'Exercice 2.2, en utilisant des matrices de dimension $n = 100$ et condition $\kappa = 10^2, 10^6, 10^{10}, 10^{18}$.

Exercice 4 : Équation de Poisson en dimension 1 Prenons une fine barre métallique homogène de longueur L et de section constante. On suppose que la barre est thermiquement isolée, de telle manière que la chaleur se propage uniquement suivant la barre. On applique une source thermique extérieure $H(x)$ tout au long de la barre. On pose $u(x,t)$ la température au point x de la barre et au temps t. Alors en appliquant le principe de conservation de l'énergie, on obtient l'équation suivante, dite de la chaleur 1D :

$$\frac{\partial u(x,t)}{\partial t} = \alpha^2 \frac{\partial^2 u(x,t)}{\partial^2 x} + H(x), \qquad x \in [0,L], \quad t > 0,$$

où α^2 est le coefficient de diffusion. On suppose aussi que les extrémités de la barre sont maintenues à une température nulle,

$$u(0,t) = u(L,t) = 0.$$

Si la température initiale est donnée par $g(x)$, alors on a

$$u(x,0) = g(x) \qquad \text{with} \qquad g(0) = g(L) = 0.$$

À présent, on s'intéresse aux solutions stationnaires de cette équation, c'est à dire les solutions vérifiant $\frac{\partial u(x,t)}{\partial t} = 0$. L'équation de la chaleur stationnaire ainsi obtenue est appelée équation de Poisson. À présent, la température ne dépend plus du temps t. Si l'on appelle la température $\phi(x)$ et la source de chaleur extérieure $F(x)$, l'équation de Poisson est

$$\begin{aligned} -\frac{\partial^2 \phi(x)}{\partial^2 x} &= F(x), \qquad x \in [0,L], \\ \phi(0) &= \phi(L) = 0. \end{aligned} \tag{2}$$

Pour résoudre numériquement (2), on divise l'intervalle $[0, L]$ en n sous-intervalles de longueur $h = L/n$ avec $0 = x_0 < x_1 < ... < x_n < x_n = L$. On utilise alors un schéma de différences finies pour approcher $\phi_{xx} := \frac{\partial^2\phi(x)}{\partial^2 x}$, et on obtient finalement

$$\frac{1}{h^2}\begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & -1 & \ddots & \ddots & \\ & & \ddots & \ddots & -1 \\ & & & -1 & 2 \end{pmatrix}\begin{pmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_{n-2} \\ \phi_{n-1} \end{pmatrix} = \begin{pmatrix} F_1 \\ F_2 \\ \vdots \\ F_{n-2} \\ F_{n-1} \end{pmatrix},$$

où $F_i = F(x_i)$.

1. Résoudre le système d'équations tridiagonal précédent en utilisant l'opération Matlab ”\" pour $F(x) = -2$, $L = 1$ et $n = 10, 20, 50, 100$. Comparer avec la solution exacte.
2. Utiliser votre fonction `LUNoPivot` pour observer la structure des matrices de la décomposition LU de la matrice de Poisson vue ci-dessus. Modifier votre fonction `LUNoPivot.m` pour le rendre plus rapide pour ce système d'équations tridiagonal.

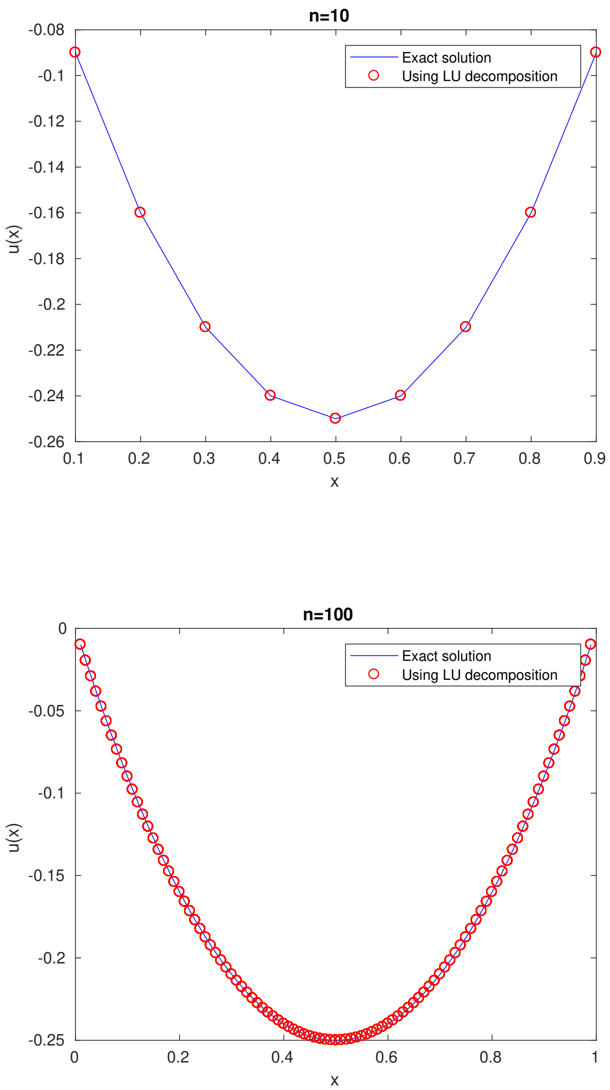

FIGURE 1 – Solutions de l'équation de Poisson avec $F(x) = -2$ pour $n = 10$ et 100.