Calcul scientifique en Fortran 90
TP Résolution de systèmes linéaires

Préambule

Dans ce TP, on va programmer des méthodes directes pour résoudre des systèmes linéaires de la forme :

$$Ax = b,$$

où $A \in \mathcal{M}(\mathbb{R})$ est une matrice à coefficients réels inversible, x et b sont des vecteurs de $\mathbb{R}^n$.

Première partie - Factorisations

1. Dans un module `mod_algebre`, programmer une procédure `lu` qui réalise la décomposition LU d'une matrice A. Cette procédure doit donc avoir une variable d'entrée (une matrice) et une variable de sortie (la matrice M contenant les deux matrices de la factorisation LU de la précédente).
   L'algorithme de la décomposition LU est rappelé ci-dessous. À la fin de

Algorithme 1 Algorithme de la décomposition LU d'une matrice A de taille n

$M = A$
Pour $k = 1$ à $n - 1$ **faire**
  $M(k+1:n, k) = M(k+1:n, k)/M(k, k)$
  $M(k+1:n, k+1:n) = M(k+1, k+1:n) - M(k+1:n, k)M(k, k+1:n)$
Fin pour

   cet algorithme, les termes non nuls de L sont dans la partie triangulaire inférieure de M (sauf les 1 sur la diagonale) et ceux de U dans la partie triangulaire supérieure.
2. Écrire un programme `SysLin`, qui construit une matrice A au profil dynamique (remplie en se basant sur un exemple du cours ou du TD d'analyse

numérique pour le moment), utilise le module `mod_algebre` pour faire la décomposition LU de A, récupère le résultat et construit L et U à partir de M, puis vérifie le résultat.

3. Rajouter dans le module `mod_algebre` une fonction `chol` qui effectue la factorisation de Cholesky de A. Cette fonction doit donc avoir une variable d'entrée (une matrice) et une variable de sortie (la matrice L de la décomposition de Cholesky.
   On rappelle l'algorithme de la décomposition de Cholesky :

Algorithme 2 Algorithme de la décomposition de Cholesky d'une matrice A de taille n

Pour i=1 à n **faire**

$$L_{i,i} = \sqrt{A_{i,i} - \sum_{k=1}^{i-1} L_{i,k}^2}$$

Pour j=i+1 à n **faire**

$$L_{i,j} = \left(A_{i,j} - \sum_{k=1}^{i-1} L_{i,k} L_{j,k}\right) / L_{i,i}$$

Fin pour

$L_{i,i+1:n} = 0.$

Fin pour

4. Compléter le programme `SysLin` pour qu'il effectue également la décomposition de Cholesky de A et vérifie le résultat.
   Indication : On pourra utiliser l'instruction `Transpose`.
5. Améliorer la fonction `chol` pour qu'elle renvoie un message d'erreur au lieu de planter si A n'est pas sdp.

Deuxième partie - résolution

1. Programmer dans le module `mod_algebre` une fonction qui résout le système $Ax = b$ une fois la factorisation LU effectuée. L'appel devra donc être de la forme suivante : `x=reslu(M,b)`, où `M` est le résultat de la procédure `lu`.
2. Programmer également dans ce même module une fonction qui résout le système $Ax = b$ une fois la factorisation de Cholesky effectuée. L'appel devra donc être de la forme suivante : `x=reschol(L,b)`, où `L` est le résultat de la fonction `chol`.
3. Tester ces fonctions sur des exemples.

Troisième partie : temps de calcul

1. Écrire une fonction interne au programme `SysLin` qui génère une matrice de taille n de l'une des deux formes suivantes :

$$A = I_n + \lambda B B^\top, \text{ (premier choix)}$$

$$A = \begin{pmatrix} 2 & -1 & 0 & \dots & 0 \\ -1 & 2 & -1 & \dots & 0 \\ 0 & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & -1 & 2 \end{pmatrix}, \text{ (second choix)}$$

   où B est une matrice contenant des nombres aléatoires entre 0 et 1 et $\lambda < 1$. Pour générer B on pourra utiliser la procédure `random_number(B)`.
2. Faire les décompositions LU et Cholesky de ces matrices dans le programme principal.
3. Obtenir le temps CPU de chacune de ces décompositions. Une manière de procéder est d'utiliser la procédure `Cpu_time` :

```
Call Cpu_time(t1)
!Instructions
Call Cpu_time(t2)
tcpu = t2-t1
```

4. Modifier le programme `SysLin` pour qu'il écrive dans des fichiers le temps CPU nécessaire aux factorisations LU et Cholesky pour chacun des deux types de matrices et pour $n = 2, 5, 10, 20, 50, 100$ et 200.
5. Tracer les courbes sous Gnuplot. Ces résultats sont-ils conformes à ceux obtenus dans le cours d'analyse numérique ?

Quatrième partie : temps de calcul de la méthode "naïve" ⋆⋆

Dans cette partie, on va vérifier que la méthode consistant à calculer l'inverse de la matrice grâce à la formule $A^{-1} = \dfrac{1}{det(A)} Com(A)^\top$ a un coût astronomique.

1. Écrire dans le module `mod_algebre` une fonction *récursive* `det` qui calcule le déterminant d'une matrice en faisant un développement par rapport à la première ligne.
   Pour rappel, la formule est :

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

où $A_{1,j}$ est le mineur du terme $a_{1,j}$.
Précision : une fonction ou subroutine `recursive` s'appelle elle-même comme son nom l'indique. On la déclare en rajoutant le mot-clé `recursive` avant `function` ou `subroutine`. Par exemple, voici le calcul de factorielles :

```
Recursive Function Factoriel(n)Result(fact)

  Integer, Intent(in) :: n
  Integer             :: fact

  If (n==0) Then
    fact = 1
  Else
    fact = n*Factoriel(n-1)
  End If

End Function Factoriel
```

Attention : il faut s'assurer que la boucle de récursivité s'arrête !

2. Vérifier cette fonction, en la comparant aux résultats donnés par les décompositions déjà programmées.
3. Écrire une fonction `InvMat` qui calcule l'inverse d'une matrice en utilisant la formule avec la comatrice et la fonction récursive `det`. Vérifier cette fonction sur des exemples de taille modeste.
4. Faire en sorte que le programme principal rende le temps CPU passé à calculer l'inverse d'une matrice et constater ce temps sur les exemples de la troisième partie. **!!!!Attention à ne pas prendre n trop grand!!!!**. Si c'est le cas, on peut interrompre un exécutable en tapant `Ctrl - C`.