Résolutions de systèmes triangulaires

Mars 2021

1 Introduction

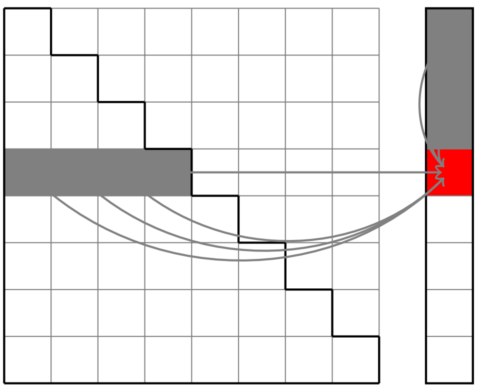
Le but de ce TP est de se familiariser avec la syntaxe de base de Fortran (déclarations de variables et de procédures, boucles, etc.) et d'illustrer l'intérêt d'implanter un algorithme qui suive le schéma de stockage imposé par le langage utilisé.

Il s'agit d'implanter deux versions différentes d'un algorithme de résolution de systèmes triangulaires inférieurs : la résolution triangulaire *sans report* et la résolution triangulaire *avec report*. Les deux algorithmes sont rappelés et illustrés ci-dessous. Notez qu'ils effectuent exactement les mêmes calculs, seul l'ordre change.

Algorithme 1
Résolution triangulaire sans report

Entrées : matrice triangulaire L
second membre b
Sortie : solution $x = L^{-1}b$

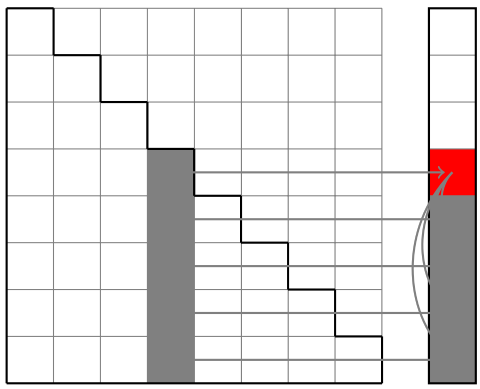
$x = b$
pour $j = 1$ **à** n **faire**
  pour $i = 1$ **à** $j-1$ **faire**
    $x_j = x_j - l_{ji}x_i$
  fin pour
  $x_j = \dfrac{x_j}{l_{jj}}$
fin pour



Algorithme 2
Résolution triangulaire avec report

Entrées : matrice triangulaire L
second membre b
Sortie : solution $x = L^{-1}b$

$x = b$
pour $j = 1$ **à** n **faire**
  $x_j = \dfrac{x_j}{l_{jj}}$
  pour $i = j+1$ **à** n **faire**
    $x_i = x_i - l_{ij}x_j$
  fin pour
fin pour



2 Implantation

Le fichier `test_solve_inf.F90` contient un squelette de programme principal qui initialise la matrice et le second membre ; la matrice est stockée dans un tableau carré dont la partie triangulaire supérieure ne doit pas être accédée. Complétez `test_solve_inf.F90` en rajoutant deux procédures `left_looking_solve` (résolution sans report) et `right_looking_solve` (résolution avec report), qui doivent avoir l'interface suivante :

`[left/right]_looking_solve(L,x,b,n)`

Sémantique : effectue la résolution sans/avec report du système triangulaire $\texttt{Lx} = \texttt{b}$.

Entrées :
- `L`, matrice de taille $\texttt{n}\times\texttt{n}$ de nombres réels double précision.
- `b`, second membre, vecteur de taille `n` de nombres réels double précision.
- `n`, entier.

Sortie : `x`, vecteur de taille `n`.

Pré-conditions :
- `L` est initialisée et aucun terme de sa diagonale n'est nul.
- $\texttt{n} > 0$.

Post-conditions : `x` contient la solution de $\texttt{Lx} = \texttt{b}$.

Ajoutez également une <u>fonction</u> de calcul de l'erreur inverse avec l'interface suivante :

`backward_error(L,x,b,n)`

Sémantique : calcule l'erreur inverse $\frac{||\texttt{Lx-b}||_2}{||\texttt{b}||_2}$.

Entrées :
- `L`, matrice de taille $\texttt{n}\times\texttt{n}$ de nombres réels double précision.
- `x`, solution calculée, vecteur de taille `n` de nombres réels double précision.
- `b`, second membre, vecteur de taille `n` de nombres réels double précision.
- `n`, entier.

Retour : un nombre réel double précision.

Pré-conditions : $\texttt{n} > 0$.

Post-conditions : $\emptyset$.

Pour compiler, utilisez `make`. Pour exécuter le code principal, tapez `./test_solve_inf`.

3 Performances

Utilisez la fonction `cpu_time` afin de mesurer le temps passé dans les procédures `left_looking_solve` et `right_looking_solve`.

question de compréhension

Faites des tests sur des matrices de tailles raisonnables ($n \leqslant 20000$) et expliquez les différences de performance entre les deux algorithmes (insérez la réponse en fin du fichier *test_ solve_ inf.F90* sous forme de commentaires).

À rendre

Déposer sous moodle une archive de nom **FORTRAN_VOTRE_NOM.{tar, zip, rar}** contenant les fichiers *test_ solve_ inf.F90* et *Makefile*.