TP Fortran – Résolution de systèmes triangulaires

Février 2017

1 Introduction

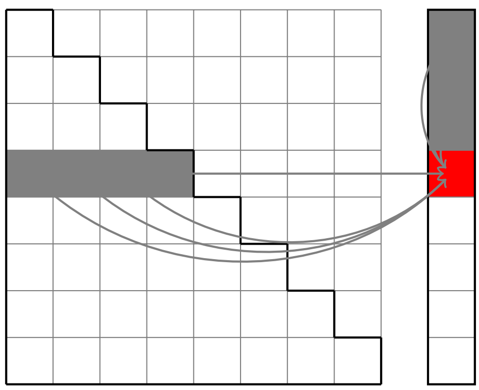
Le but de ce TP est de se familiariser avec la syntaxe de base de Fortran (déclarations de variables et de procédures, boucles, etc.) et d'illustrer l'intérêt d'implanter un algorithme qui suive le schéma de stockage imposé par le langage utilisé.

Il s'agit d'implanter deux versions différentes d'un algorithme de résolution de systèmes triangulaires inférieurs : la résolution triangulaire *sans reports* et la résolution triangulaire *avec reports*. 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 reports

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

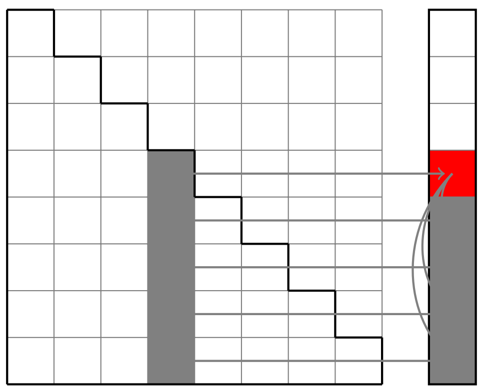
$x = b$
pour $j = 1$ **à** n **faire**
  pour $i = 1$ **à** $j-1$ **faire**
    $x_j = x_j - l_{ji}x_i$
  fin pour
  $x_j = \dfrac{x_j}{l_{jj}}$
fin pour



Algorithme 2
Résolution triangulaire avec reports

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.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.F90` en rajoutant deux procédures `left_looking_solve` (résolution sans reports) et `right_looking_solve` (résolution avec reports), qui doivent avoir l'interface suivante :

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

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

Entrées :
- `L`, matrice de taille $\mathtt{n}\times\mathtt{n}$ 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.
- $\mathtt{n} > 0$.

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

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

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

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

Entrées :
- `L`, matrice de taille $\mathtt{n}\times\mathtt{n}$ 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 : $\mathtt{n} > 0$.

Post-conditions : $\emptyset$.

Pour compiler, utilisez `make`. Pour lancer le code principal, lancez `test_solve`.

3 Performances

Utilisez la fonction `cpu_time` afin de mesurer le temps passé dans les procédures `left_looking_solve` et `right_looking_solve`. Faites des tests sur des matrices de tailles raisonnables ($n \leqslant 20000$) et expliquez les différences de performances entre les deux algorithmes.