Résolution de Systèmes Linéaires issues des EDP

ModIA

TP3 Méthodes Multigrilles

Année scolaire 2024–2025

Carola Kruse Ronan Guivarch

1 Introduction

Dans ce TP, nous allons implémenter une méthode multigrilles en 1D dans Matlab.

2 Multigrilles 1D (Matlab)

Nous allons développer un solveur multigrilles pour résoudre le problème de Poisson avec des conditions aux limites de Dirichlet homogènes dans une dimension

$$\begin{aligned} -u'' &= f, && \text{in } \Omega = [0,1] \\ u &= 0, && \text{on } \Gamma \end{aligned}$$

Dans la suite, nous supposons que la solution exacte de ce problème de Poisson est donné par

$$u(x) = x(1-x). \tag{1}$$

Il est facile de vérifier que cette solution respecte les conditions aux limites de Dirichlet (comment ?). Nous utilisons cette solution pour trouver le membre de droite

$$f = -u'' = 2. \tag{2}$$

En choisissant une solution exacte, il est possible de calculer les erreurs introduites par la discrétisaton par différences finies. Elle peut ainsi être utilisée pour valider l'implémentation de la discrétisation.

2.1 Maillage

Nous nous concentrons sur un solveur à deux niveaux, ça veut dire que nous avons une hiérarchie de seulement deux maillages. Pour cela, soit N le nombre d'éléments dans l'intervalle $[0,1]$. Nous allons donc définir les deux domaines suivants avec $h = \frac{1}{N}$

$$\begin{aligned} &\Omega_h, && x_i = ih, && i = 0,..,N && \text{(fine grid)}, \\ &\Omega_{2h}, && x_i = 2ih, && i = 0,..,N/2 && \text{(coarse grid)}. \end{aligned}$$

En règle générale, nous devrions construire ce maillage et utiliser les informations pour obtenir la matrice et les opérateurs de transfert. Pour cet exemple simple , ce n'est par contre pas nécessaire et tous les calculs peuvent être faits à la main.

2.2 Discrétisation par différences finies

Nous utilisons l'approche *différence centrée*

$$u_i'' \approx \frac{-u_{i-1} + 2u_i - u_{i+1}}{h^2} = f_i, \quad i = 1, .., N-1 \tag{3}$$

Avec cette discrétisation, nous obtenons, sur la grille fine, le système linéaire

$$A\mathbf{u} = \frac{1}{h^2} \begin{pmatrix} 2 & -1 & 0 & \dots & 0 \\ -1 & 2 & -1 & \ddots & \vdots \\ 0 & -1 & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & 2 & -1 \\ 0 & \dots & 0 & -1 & 2 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_{N-1} \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_{N-1} \end{pmatrix} \tag{4}$$

Comme nous avons des conditions aux limites de Dirichlet homogènes, ce n'est pas nécessaire d'inclure les deux degrés de libertés u_0 et u_N dans la matrice. La matrice du problème, sur la grille fine, est donc de taille $N-1$.

Travail demandé :

- Écrivez une fonction Matlab avec N en entrée et $A \in \mathbb{R}^{N-1 \times N-1}$ en sortie

```
function A = getMatrixA(N)
% Your code here
```

Astuce: Vous pouvez utiliser la commande `diag(vec,m)` de Matlab, où `vec` correspond au vecteur des valeurs sur la diagonale, et $m = 0$ est la diagonale, $m = 1$ et $m = -1$ sont la super- et sous-diagonale (`help diag` pour plus d'information).

Pour obtenir le membre de droite dans Matlab, créez un vecteur de longueur $N-1$ et remplissez-le avec la valeur 2.

```
rhsf = ?
```

Nous sommes maintenant en mesure de calculer la solution (jusqu'à précision de machine) du système linéaire (4), que nous allons utiliser comme référence pour la solution obtenue par la méthode de multigrilles. Pour cela, nous faisons

```
sol_ref = A \ rhsf;
```

2.3 Lisseur

Pendant le cours, nous avons vu deux lisseurs, la méthode de Jacobi pondérée et de Gauss-Seidel pondérée. Ici, nous utilisons la méthode de Jacobi pondérée

$$\mathbf{u}^{(m+1)} = (I - \omega D^{-1} A)\mathbf{u}^m + \omega D^{-1}\mathbf{f}, \tag{5}$$

avec $0 \leq \omega \leq \frac{2}{\lambda_{max}(D^{-1}A)}$

Travail demandé : Implémentez une fonction qui applique m itérations de Jacobi pondérée à une fonction $u^{(m)}$ avec $u^{(m+1)}$ en sortie

```
function ump1 = weighted_jacobi(A,um,f,omega,m)
% Your code here
```

Ensuite, nous illustrons l'effet du lisseur sur un vecteur. D'abord, nous créons un vecteur correspondant au k-ème vecteur propre de la matrice A sur un maillage avec 64 éléments. Après, nous appliquons m itérations du Jacobi. Pour cela, ajoutez le k-ième vecteur propre dans le code suivant.

```
clear all;
N = 64;
h = 1/N;
A = getMatrixA(N);
omega = 2/3;
rhsf = zeros(N-1,1);
j = 1:N-1;

k = 6;
% kieme vecteur propre
um = ?

m = 10;
ump1 = weighted_jacobi(A,um',rhsf,omega,m);

x=h:h:1-h;
plot(x,um,x, ump1);
legend('um','ump1');
title('Damping effect of weighted Jacobi method')
```

Travail demandé : Variez

- le nombre d'itérations de Jacobi,
- le paramètre ω dans la méthode de Jacobi.
- le vecteur propre en utilisant $k \in \{3, 12, 48\}$,

Qu'est-ce que vous observez ?

2.4 Opérateurs de transfert

Pour transférer les informations d'un maillage à l'autre, il nous faut des opérateurs de transfert inter-grilles. Dans ce simple exemple, nous utilisons l'interpolation linéaire pour la prolongation

$I_{2h}^h : \mathbb{R}^{\frac{N}{2}-1} \to \mathbb{R}^{N-1}$ avec

$$v_{2j}^h = v_j^{2h},$$
$$v_{2j+1}^h = \frac{1}{2}(v_j^{2h} + v_{j+1}^{2h})$$

et *full weighting* pour la restriction $I_h^{2h} : \mathbb{R}^{N-1} \to \mathbb{R}^{\frac{N}{2}-1}$ avec

$$v_j^{2h} = \frac{1}{4}(v_{2j-1}^h + 2v_{2j}^h + v_{2j+1}^h)$$

Travail demandé : Implémentez l'opérateur d'interpolation I_{2h}^h et l'opérateur de restriction I_h^{2h} dans une forme matricielle comme vu en cours. Quel est l'avantage de choisir le couple interpolation et full weighting ?

```
% Interpolation
function I2hh = interpol(N)
% Your code here
```

```
% Restriction

Ih2h = ?
```

2.5 Schéma de multigrilles à deux niveaux

Nous pouvons maintenant définir la méthode de multigrilles sur deux niveaux. Remplissez le code suivant. Nous commençons avec le setup des éléments que nous avons besoin :

```
clear all;
% Setup maillage
N = 64;
h = 1/N;

% Setup Jacobi
omega = 2/3;

% Setup of the fine and coarse grid matrix
% and the right-hand side
Ah = getMatrixA(N);
A2h = getMatrixA(N/2);

rhsf = 2*ones(N-1,1);

% Compute direct solution of linear system
sol_ref = ?

% Setup interpolation matrix
I2hh = interpol(N);
Ih2h = ?
```

Maintenant, nous définissons une méthode de multigrilles en 1D. Completez le code suivant :

```
% Initial vector
v(1:N-1,1) = 0;

% Vector to store the error at each iteration.
% We do 10 iterations.
err(1:10) = 0;

% Multigrid iterations with 2 pre-smoothing steps

for i=1:10
  v = weighted_jacobi(Ah,v,rhsf,omega,2);

  % residual on fine grid
  res_h = ?

  % Restriction of residual to coarser grid
  res_2h = ?

  % Solve the coarse grid error equation
  e_2h = ?

  % Interpolate the coarse grid error to the fine grid
  e_h = ?

  % Update the approximate fine grid solution
  v = v + ?

  % Compute the error with respect to the direct
  % solution of the linear system
  err(i) = norm(sol_ref-v);
end
```

- Que constatez-vous pour l'erreur calculée ?

2.6 Adaptation du critère d'arrêt

Veuillez noter que la solution directe du système linéaire n'est généralement pas disponible. Au lieu de cela, nous pouvons calculer le résidu à chaque étape du processus itératif pour constater une réduction de l'erreur et définir un critère d'arrêt

$$q^{(m)} = \frac{\|res_h^{(m)}\|_2}{\|res_h^{(0)}\|_2} = \frac{\|b - Ax^{(m)}\|_2}{\|b - Ax^{(0)}\|_2}$$

Dans l'exemple, nous avons utilisé l'erreur algébrique à des fins d'illustration.

- Reprenez le code en remplaçant la boucle *for* par une boucle *while* permettant de stopper la boucle quand l'erreur algébrique est plus petite qu'un ε donné ou qu'un nombre maximum d'itérations est atteint.
- Vous ferez varier la taille du problème. Que constatez-vous pour le nombre d'itérations pour un ε donné ?