Formation en Calcul Scientifique - LEM2I
–
Mise en pratique

Loïc Gouarin [1], Violaine Louvet [2]

[1] Laboratoire de Mathématique d'Orsay - CNRS
[2] Institut Camille Jordan - CNRS

12-14/12/2011

Problème étudié

On souhaite résoudre l'équation aux dérivées partielles suivante

$$\begin{cases} -\Delta u = f \quad \text{dans} \quad \Omega = [a,b] \times [c,d] \\ u = g \quad \text{sur} \quad \partial\Omega \end{cases}$$

Discrétisation de l'intervalle $[a,b]$ suivant l'axe des x

$$h_x = \frac{b-a}{n_x+1} \quad (n_x + 2 \text{ points})$$

Discrétisation de l'intervalle $[c,d]$ suivant l'axe des y

$$h_y = \frac{d-c}{n_y+1} \quad (n_y + 2 \text{ points})$$

Problème étudié

On souhaite résoudre l'équation aux dérivées partielles suivante

$$\begin{cases} -\Delta u = f \quad \text{dans} \quad \Omega = [a,b] \times [c,d] \\ u = g \quad \text{sur} \quad \partial\Omega \end{cases}$$

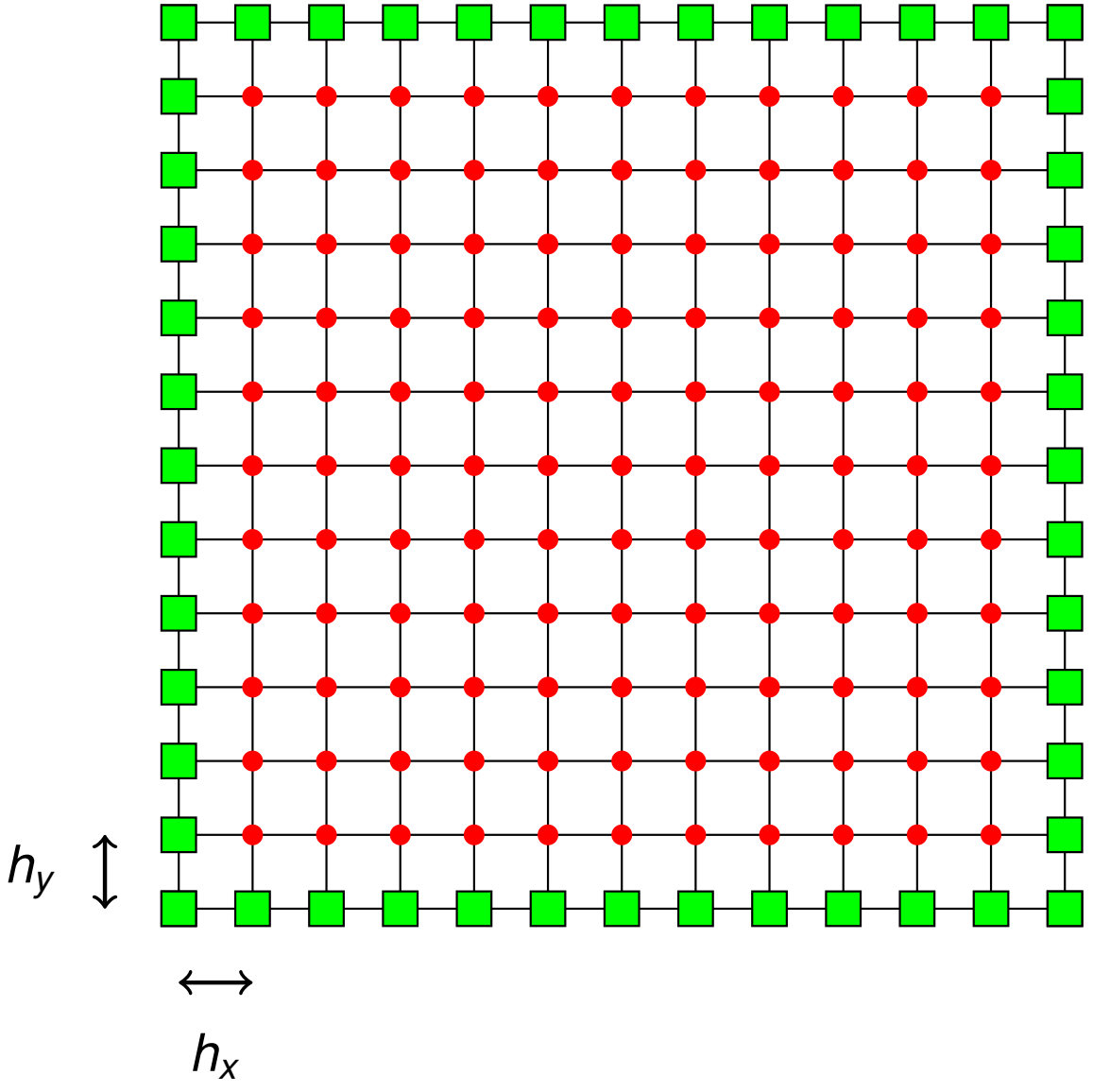
Discrétisation de l'intervalle $[a,b]$ suivant l'axe des x

$$x(i) = x_i = a + ih_x, \quad i = 0, \cdots, n_x + 1$$

Discrétisation de l'intervalle $[c,d]$ suivant l'axe des y

$$y(j) = y_j = c + jh_y, \quad j = 0, \cdots, n_y + 1$$

Domaine discrétisé



- points intérieurs
- points extérieurs

Méthode des différences finies

Cette méthode consiste à approximer les dérivées partielles d'une équation au moyen des développements de Taylor.

Pour h_x suffisamment petit, on peut écrire les développements de Taylor au voisinage du point (x_i, y_j) de la solution u

$$u(x_{i+1}, y_j) = u(x_i, y_j) + h_x \frac{\partial u}{\partial x}(x_i, y_j) + \frac{h_x^2}{2} \frac{\partial^2 u}{\partial x^2}(x_i, y_j) + \frac{h_x^3}{6} \frac{\partial^3 u}{\partial x^3}(x_i, y_j) + O(h_x^4)$$

$$u(x_{i-1}, y_j) = u(x_i, y_j) - h_x \frac{\partial u}{\partial x}(x_i, y_j) + \frac{h_x^2}{2} \frac{\partial^2 u}{\partial x^2}(x_i, y_j) - \frac{h_x^3}{6} \frac{\partial^3 u}{\partial x^3}(x_i, y_j) + O(h_x^4)$$

Méthode des différences finies

$$\frac{\partial^2 u}{\partial x^2}(x_i, y_j) = \frac{u(x_{i+1}, y_j) - 2u(x_i, y_j) + u(x_{i-1}, y_j)}{h_x^2} + O(h_x^2)$$

Méthode des différences finies

$$\frac{\partial^2 u}{\partial x^2}(x_i, y_j) = \frac{u(x_{i+1}, y_j) - 2u(x_i, y_j) + u(x_{i-1}, y_j)}{h_x^2} + O(h_x^2)$$

$$\frac{\partial^2 u}{\partial y^2}(x_i, y_j) = \frac{u(x_i, y_{j+1}) - 2u(x_i, y_j) + u(x_i, y_{j-1})}{h_y^2} + O(h_y^2)$$

Méthode des différences finies

$$\frac{\partial^2 u}{\partial x^2}(x_i, y_j) = \frac{u(x_{i+1}, y_j) - 2u(x_i, y_j) + u(x_{i-1}, y_j)}{h_x^2} + O(h_x^2)$$

$$\frac{\partial^2 u}{\partial y^2}(x_i, y_j) = \frac{u(x_i, y_{j+1}) - 2u(x_i, y_j) + u(x_i, y_{j-1})}{h_y^2} + O(h_y^2)$$

$$-\Delta u(x_i, y_j) = -\frac{\partial^2 u}{\partial x^2}(x_i, y_j) - \frac{\partial^2 u}{\partial y^2}(x_i, y_j)$$

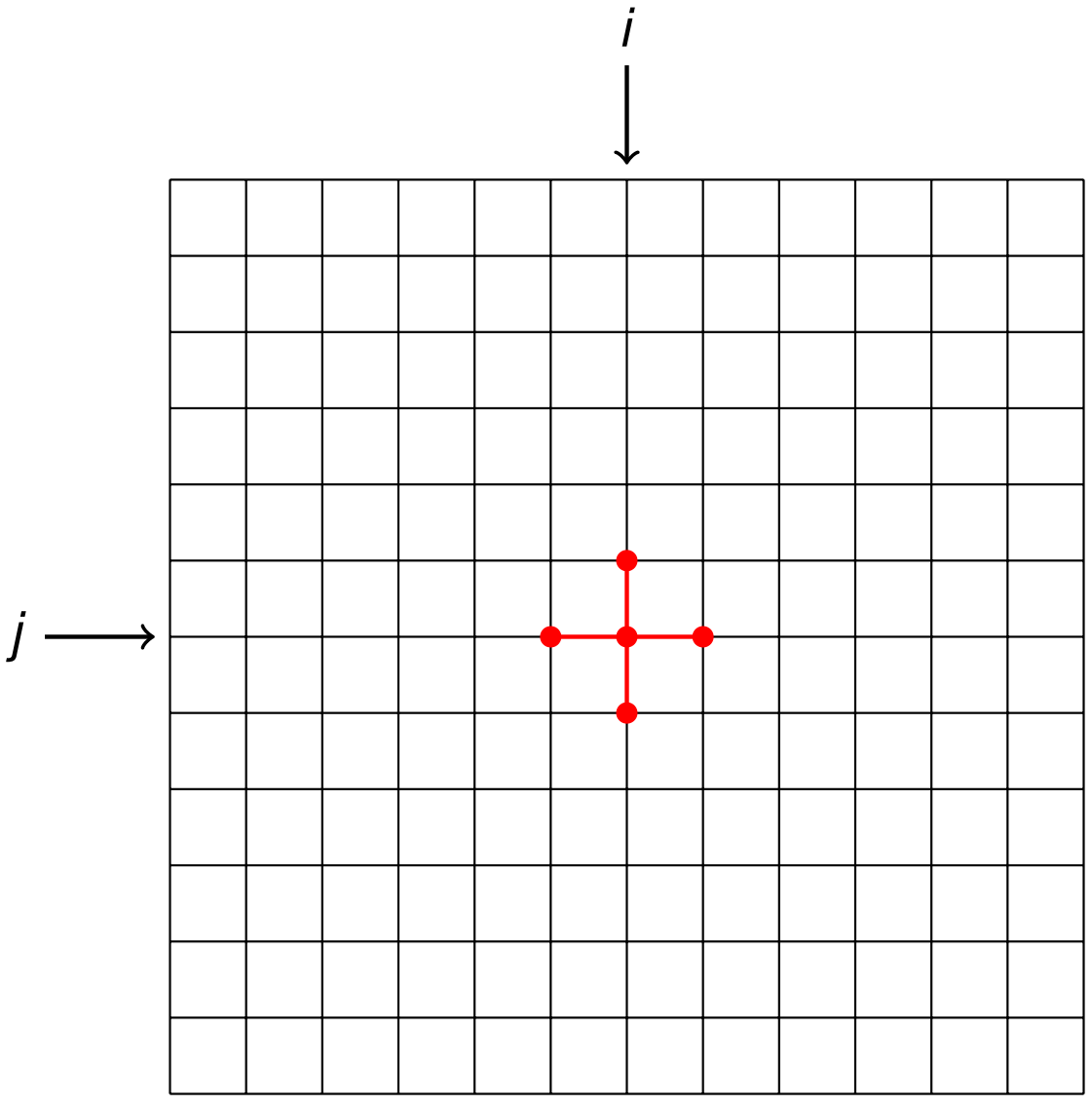
Méthode des différences finies

$$\frac{\partial^2 u}{\partial x^2}(x_i, y_j) = \frac{u(x_{i+1}, y_j) - 2u(x_i, y_j) + u(x_{i-1}, y_j)}{h_x^2} + O(h_x^2)$$

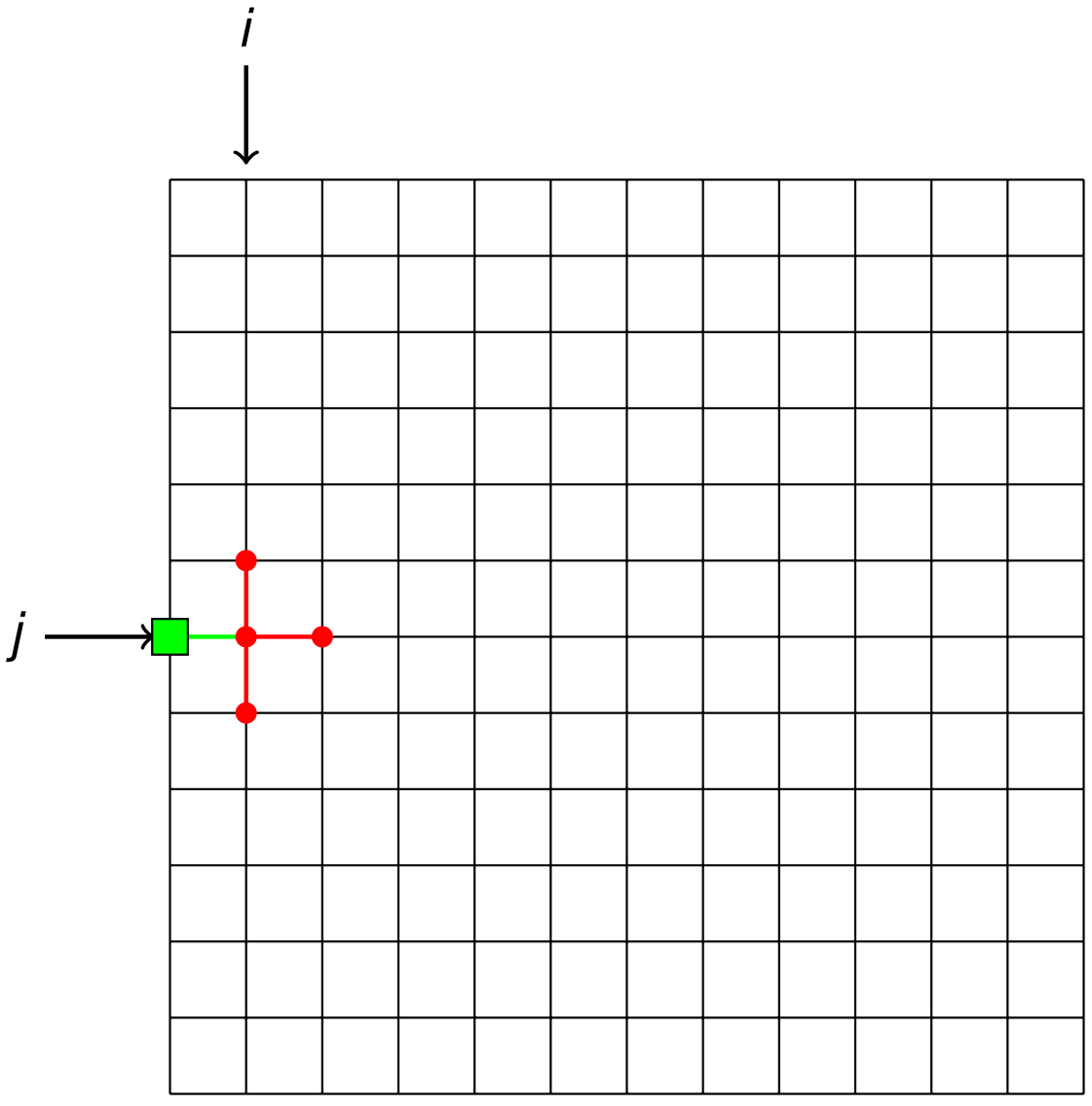
$$\frac{\partial^2 u}{\partial y^2}(x_i, y_j) = \frac{u(x_i, y_{j+1}) - 2u(x_i, y_j) + u(x_i, y_{j-1})}{h_y^2} + O(h_y^2)$$

$$-\Delta u(x_i, y_j) \approx \frac{-u(x_{i+1}, y_j) + 2u(x_i, y_j) - u(x_{i-1}, y_j)}{h_x^2} + \frac{-u(x_i, y_{j+1}) + 2u(x_i, y_j) - u(x_i, y_{j-1})}{h_y^2}$$

Méthode des différences finies



Méthode des différences finies



Système linéaire Ax = b

$$A = \begin{bmatrix} A_1 & A_2 & 0 & \cdots & 0 \\ A_2 & A_1 & A_2 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & A_2 & A_1 & A_2 \\ 0 & \cdots & 0 & A_2 & A_1 \end{bmatrix}$$

avec

$$A_1 = \begin{bmatrix} \frac{2}{h_x} + \frac{2}{h_y} & -\frac{1}{h_x} & 0 & \cdots & 0 \\ -\frac{1}{h_x} & \frac{2}{h_x} + \frac{2}{h_y} & -\frac{1}{h_x} & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & -\frac{1}{h_x} & \frac{2}{h_x} + \frac{2}{h_y} & -\frac{1}{h_x} \\ 0 & \cdots & 0 & -\frac{1}{h_x} & \frac{2}{h_x} + \frac{2}{h_y} \end{bmatrix}$$

Système linéaire Ax = b

$$A = \begin{bmatrix} A_1 & A_2 & 0 & \cdots & 0 \\ A_2 & A_1 & A_2 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & A_2 & A_1 & A_2 \\ 0 & \cdots & 0 & A_2 & A_1 \end{bmatrix}$$

avec

$$A_2 = \begin{bmatrix} -\frac{1}{h_x} & 0 & 0 & \cdots & 0 \\ 0 & -\frac{1}{h_x} & 0 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & 0 & -\frac{1}{h_x} & 0 \\ 0 & \cdots & 0 & 0 & -\frac{1}{h_x} \end{bmatrix}$$

Système linéaire Ax = b

$$b = \begin{bmatrix} f(x_1, y_1) + \dfrac{g(x_0, y_1)}{h_x} + \dfrac{g(x_1, y_0)}{h_y} \\ f(x_2, y_1) + \dfrac{g(x_1, y_0)}{h_y} \\ \vdots \\ f(x_i, y_j) \\ \vdots \\ f(x_{n_x}, y_{n_y}) + \dfrac{g(x_{n_x+1}, y_{n_y})}{h_x} + \dfrac{g(x_{n_x}, y_{n_y+1})}{h_y} \end{bmatrix}$$

Stockage de la matrice A

On utilisera un stockage creux appelé CSR (compressed sparse row). Sa structure de données peut être représentée par

- val : tableau représentant les valeurs non nulles de la matrice.
- col_ind : tableau représentant les indices des colonnes où on trouve les valeurs non nulles.
- row_ptr : liste des indices où commence chacune des lignes .

Exemple CSR

$$A = \begin{pmatrix} 10 & 0 & 0 & 0 \\ 3 & 9 & 0 & 0 \\ 0 & 7 & 8 & 7 \end{pmatrix}$$

$$val = [10, 3, 9, 7, 8, 7]$$

$$col\_ind = [0, 0, 1, 1, 2, 3]$$

$$row\_ptr = [0, 1, 3, 6]$$

Résolution du système

- méthode directe : décomposition LU
- méthode itérative : gradient conjugué

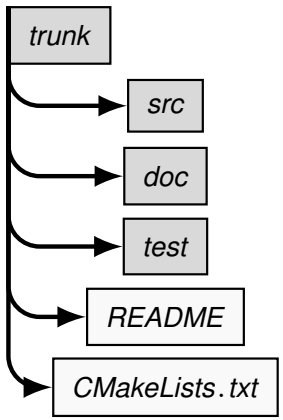
Prise en main des sources

- *laplacian.f90* : module effectuant la création de la matrice du Laplacien avec la prise en compte des conditions aux limites.
- *CSR.f90* : module définissant le type dérivé fortran pour les matrices creuses à stockage CSR et réalisant différentes opérations sur ces matrices.
- *linalg.f90* : module réalisant l'algorithme du gradient conjugué.
- *test_cg.f90* : programme principal pour la résolution de l'équation de la chaleur par le gradient conjugué

Structuration

Structurer le répertoire de travail pour :

- Dissocier les différents éléments
- Initier un référentiel subversion propre
- Faciliter le développement à plusieurs



A vous

Créer la structure ci-dessus et copier les sources des modules du code dans le répertoire *src* et le programme principal dans le répertoire *test*

Etape 1 : Compilation

On va utiliser :

- *CMake* pour automatiser la compilation du code

A vous

- Créer un fichier *CMakeLists.txt* dans le répertoire *src* pour créer une bibliothèque à partir des fichiers fortran.
- Créer un fichier *CMakeLists.txt* dans le répertoire *test* pour créer l'exécutable associé au programme *test_cg.f90*
- Créer un fichier *CMakeLists.txt* pour le projet dans le répertoire *trunk*
- Compiler (out-of-sources)

Etape 2 : Debugger

A vous

- Corriger le programme pour finir la compilation
- Utiliser un outil de debuggage pour trouver l'erreur d'exécution
  - Par défaut, le build de CMake est en mode Debug
  - On peut le préciser directement :
    cmake -DCMAKE_BUILD_TYPE=Debug ..
- Utiliser *valgrind* pour vérifier l'intégrité de l'usage de la mémoire.

On a un code qui tourne !!

Etape 3 : Intégrer la documentation

A vous

- Documenter les différentes routines du code
- Construire le fichier de configuration pour doxygen
- Intégrer la génération automatique de documentation dans les *CMakeLists.txt*
- Générer la documentation

Etape 4 : Ajouter les sorties graphiques

- *vtkTools.cxx* : fichier C++ qui permet la gestion des sorties du code au format *VTK*.

A vous

- Ajouter ce fichier dans les sources du code
- Ajouter un appel à la fonction *save_vtk* dans le programme principal
- Modifier les différents fichiers *CMakeLists.txt* pour prendre en compte ce nouveau fichier source :
  - Dépendances de l'exécutable avec la bibliothèque *vtkRendering* (*test/CMakeLists.txt*)
  - Compilation d'un fichier C++ (*src/CMakeLists.txt*)
  - Localisation de la bibliothèque *VTK* et inclusion de ses dépendances (*CMakeLists.txt*)

Etape 5 : Profiler le code

Objectifs :

- Identifier les parties du code les plus coûteuses en temps de calcul
- Evaluer l'utilisation des caches

A vous

- Utiliser *gprof* pour profiler le code
  - Définir les options pour le profiling
  - *cmake -DCMAKE_BUILD_TYPE=Profile ..*
- Utiliser *cachegrind* pour identifier les éventuels problèmes de cache

Quelles sont les parties à optimiser ?

Etape 6 : Optimiser le code

Faire porter l'effort d'optimisation sur les parties les plus coûteuses en temps de calcul.

A vous

- Utiliser les techniques vues en cours pour améliorer le programme

Peut-on faire mieux ?

Etape 7 : Utiliser une résolution directe

- L'algorithme de gradient a besoin de beaucoup d'itérations pour converger.
- Amélioration possible : changer la méthode numérique de résolution

A vous

- Etudier l'interface Fortran pour l'utilisation de *SuperLU*
- Intégrer le script *CMake* de recherche de package pour *SuperLU* et modifier le *CMakeLists.txt* principal en conséquence.
- Modifier le *src/CMakeLists.txt*.
- Ecrire un deuxième programme test en utilisant *SuperLU* et modifier le *test/CMakeLists.txt*.
- Compiler et tester le nouveau code, le comparer avec la résolution par gradient conjugué.