4M054 : Mise en oeuvre des éléments finis
X.Claeys, G.Migliorati, B.Thierry & P.-H.Tournier

Projet

Il est possible de réaliser le projet en binôme. Celui-ci devra être envoyé par email à l'adresse claeys@ann.jussieu.fr au plus tard le dimanche 12 mai 2019 à minuit

Le projet sera envoyé sous forme d'archive .zip contenant les éléments suivant :
— les codes sources du projet,
— un fichier readme résumant le contenu des différents fichiers, et la manière de s'en servir,
— un fichier makefile permettant de compiler le projet.

Vous êtes invités à réutiliser le travail déjà effectué au cours du semestre en TP. Par ailleurs, on attend que vous mettiez en place des tests de votre propre initiative pour vous assurer du bon fonctionnement de votre code et de sa conformité au cahier des charges imposé. Vous serez interrogés sur les tests mis en place.

1 Structure matrice bloc

Dans tout ce projet, on considèrera uniquement des matrices à coefficients réels. Dans la suite on notera $[\![1,n]\!] := \{1,\dots n\}$, et on définit un ensemble de vecteurs d'entiers $\mathcal{I}_n := \{\ (j_1,\dots,j_l)\ \mid\ 1 \leq j_1 < j_2 < \dots < j_l \leq n,\ l \geq 0\}$. Étant donné un élément $\mathrm{I} = (i_1,\dots,i_l) \in \mathcal{I}_n$, on notera $|\mathrm{I}| := l$ sa longueur, et on définira

$$\mathrm{R}_{n,\mathrm{I}} = (r_{j,k}) \in \mathbb{R}^{n\times|\mathrm{I}|} \quad r_{j,k} := \delta_{j,i_k} = \begin{cases} 1 & \text{si } j = i_k \\ 0 & \text{si } j \neq i_k \end{cases} \tag{1}$$

Dans cette première partie du projet, on cherche à modéliser les matrices $\mathrm{A} \in \mathbb{R}^{n\times m}$ prenant la forme par blocs suivante

$$\mathrm{A} = \sum_{q=1\dots Q} \mathrm{R}_{n,\mathrm{I}_q}\mathrm{B}_q\mathrm{R}_{m,\mathrm{J}_q}^\top \tag{2}$$

où $\mathrm{I}_q \in \mathcal{I}_n$, $\mathrm{J}_q \in \mathcal{I}_m$ et $\mathrm{B}_q \in \mathbb{R}^{|\mathrm{I}_q|\times|\mathrm{J}_q|}$ pour chaque $q = 1\dots Q$. Dans la forme de matrice (2), on n'exclut pas le fait que deux vecteurs d'entiers I_q admettent des éléments en commun : par exemple $\mathrm{I}_1 = (1,2,3)$ et $\mathrm{I}_2 = (2,3,4)$ admettraient les indices $2,3$ en commun.

Question 1) Écrire une classe `DenseMatrix` modélisant les matrices denses. Cette classe comportera les éléments suivants

données membre
- `nr` et `nc` : deux `int` représentant le nombre de lignes/colonnes
- `val` : tableau de `double` de taille `nr`×`nc` stockant les coefficients de la matrice.

fonctions membre
- un constructeur prenant en arguments deux `int` pour initialiser `nr` et `nc`, et qui met les coefficients de la matrice à 0.
- un constructeur par recopie
- une surcharge de l'opérateur = par recopie
- une surcharge de l'opérateur `( , )` prenant en argument d'entrée deux `int` et permettant un accès en lecture/écriture aux coefficients de la matrice
- une surcharge de l'opérateur `<<` permettant un affichage de la matrice dans le terminal

- une fonction membre MvProd prenant en argument un const vector<double>& noté x, et un vector<double>& noté b, et renvoyant void. Cette fonction membre réalisera le produit matrice-vecteur avec x et stockera le résultat dans b.
- une fonction membre LUSolve prenant en argument un vector<double>& noté x, et un const vector<double>& noté b, et renvoyant void. Cette fonction résoudra l'équation $\mathrm{A}\boldsymbol{x} = \boldsymbol{b}$ au moyen d'une factorisation LU et stockera le résultat dans le vecteur x.
- une fonction membre cg prenant en argument un vector<double>& noté x, et un const vector<double>& noté b, et renvoyant void. Cette fonction résoudra l'équation $\mathrm{A}\boldsymbol{x} = \boldsymbol{b}$ par l'algorithme du gradient conjugué.

Question 2) Écrire une classe Block modélisant les matrices de la forme $\mathrm{R}_{n,\mathrm{I}}\mathrm{B}\mathrm{R}_{m,\mathrm{J}}^{\top}$ où $\mathrm{I} \in \mathcal{I}_n$, $\mathrm{J} \in \mathcal{I}_m$ et $\mathrm{B} \in \mathbb{R}^{|\mathrm{I}|\times|\mathrm{J}|}$. Cette classe comportera les éléments suivants

données membre

- nr et nc : deux int représentant les nombres n et m,
- Ir et Ic : deux vector<int> représentant I et J,
- mat : une DenseMatrix représentant B.

fonctions membre

- un constructeur prenant en arguments deux int et deux vector<int> pour initialiser nr, nc, Ir et Ic, et qui initialise mat en mettant ses coefficients à 0.
- un constructeur par recopie
- une surcharge de l'opérateur = par recopie
- une fonction membre MvProd prenant en argument un const vector<double>& noté x, et un vector<double>& noté b. Cette fonction membre réalisera le produit matrice-vecteur de la matrice $\mathrm{R}_{n,\mathrm{I}}\mathrm{B}\mathrm{R}_{m,\mathrm{J}}^{\top}$ avec x et stockera le résultat dans b.

Question 3) Écrire une classe BlockMatrix modélisant les matrices $\mathrm{A} \in \mathbb{R}^{n\times m}$ de la forme (2). Cette classe comportera les éléments suivants

données membre

- nr et nc : deux int représentant les nombres n et m,
- val : un list<Block> stockant les suites de matrice $\mathrm{R}_{n,\mathrm{I}_q}\mathrm{B}_q\mathrm{R}_{m,\mathrm{J}_q}^{\top}$.

fonctions membre

- un constructeur prenant en arguments deux int pour initialiser nr, nc, et initialisant val à la liste vide,
- un constructeur par recopie,
- une surcharge de l'opérateur = par recopie,
- une surcharge de l'opérateur += avec, comme opérande de droite, une variable Block et qui ajoute cette variable à la liste val,
- une fonction membre MvProd prenant en argument un const vector<double>& noté x, et un vector<double>& noté b. Cette fonction membre réalisera le produit matrice-vecteur de la matrice A avec x et stockera le résultat dans b.
- une fonction membre MinRes prenant en argument un const vector<double>& noté x, et un vector<double>& noté b. Cette fonction résoudra l'équation $\mathrm{A}\boldsymbol{x} = \boldsymbol{b}$ au moyen de l'algorithme MinRes.

2 Préconditionneur de Schwarz additif

On se place maintenant dans la situation d'une matrice $\mathrm{A} \in \mathbb{R}^{n\times n}$ a priori dense pour laquelle on souhaite assembler un préconditionneur c'est-à-dire une matrice $\mathrm{P} \in \mathbb{R}^{n\times n}$ telle que $\mathrm{P}\cdot\mathrm{A}$ admet un meilleur conditionnement que A. Pour construire un tel préconditionneur, la méthode de Schwarz additive (ASM) consiste à se donner une famille $\mathrm{I}_q \in \mathcal{I}_n, q = 1\dots Q$ telle que $[\![1,n]\!] = \cup_{q=1}^{Q}\mathrm{I}_q$ (i.e.

chaque $j \in [\![1, n]\!]$ appartient à au moins un I_q) et à choisir

$$\mathrm{P} = \sum_{q=1}^{Q} \mathrm{R}_{n,\mathrm{I}_q} \mathrm{A}_{\mathrm{I}_q,\mathrm{I}_q}^{-1} \mathrm{R}_{n,\mathrm{I}_q}^{\top}. \tag{3}$$

Dans cette expression $\mathrm{A}_{\mathrm{I},\mathrm{I}} \in \mathbb{R}^{|\mathrm{I}|\times|\mathrm{I}|}$ est une matrice extrait de A avec les lignes et colonnes associées aux indices contenus dans I c'est-à-dire, si $\mathrm{I} = (i_1, \ldots, i_l)$ et $\mathrm{A}_{\mathrm{I},\mathrm{I}} = (b_{j,k})_{j,k=1\ldots l}$, alors $b_{j,k} = \mathrm{A}_{i_j,i_k}$ pour tout $j, k = 1 \ldots l$. On propose d'utiliser la classe `BlockMatrix` pour modéliser les préconditionneurs de Schwarz additifs.

Question 4) On s'intéresse d'abord à générer la famille de vecteurs d'indices $\mathrm{I}_q, q = 1 \ldots Q$. On propose de générer une telle famille de la manière suivante. Étant donnés trois paramètres entiers $l, r, n \in \mathbb{N}$, écrire une routine qui génère une liste de vecteurs d'entiers $\mathrm{I}_q \in \mathcal{I}_n$ vérifiant

$$\begin{aligned}
&\mathrm{I}_1 = (1, 2, \ldots, l + r)\\
&\mathrm{I}_q = (l(q-1) - r, (q-1)l - r + 1, \ldots, lq + r) \quad \text{pour } \; 1 < q \leq q_\star := E(n/l) - 1\\
&\mathrm{I}_{q_\star+1} = (lq_\star, \ldots, n)
\end{aligned} \tag{4}$$

Question 5) Ajouter dans la classe `DenseMatrix` une surcharge de l'opérateur `( , )` prenant en argument d'entrée deux `vector<int>` notés `Ir` et `Ic` et renvoyant en sortie une `DenseMatrix` formée par le bloc $\mathrm{A}_{\mathrm{I}r,\mathrm{I}c}$ extrait de la matrice A.

Question 6) Ajouter une routine `Extract` dans la classe `BlockMatrix` prenant en argument une `DenseMatrix` notée A et un `vector<vector<int>>` noté I et représentant une partition $\{\mathrm{I_q}\}_{q=1}^{Q}$ avec recouvrement de $\{1, \ldots, n\}$ où n est la taille de A. Cette routine aura l'effet suivant. Si `P` est une variable de type `BlockMatrix`, alors l'appel `P.Extract(I,A)` aura pour effet de remplir le tableau `val` de `P` de manière à ce que `P` se comporte comme la matrice

$$\sum_{q=1}^{Q} \mathrm{R}_{n,\mathrm{I}_q} \mathrm{A}_{\mathrm{I}_q,\mathrm{I}_q} \mathrm{R}_{n,\mathrm{I}_q}^{\top}$$

Ce sont bien les matrices $\mathrm{A}_{\mathrm{I}_q,\mathrm{I}_q}$ que l'on demande de stocker dans `val` et non pas les inverses $\mathrm{A}_{\mathrm{I}_q,\mathrm{I}_q}^{-1}$ car on ne souhaite avoir à inverser des blocs matriciels explicitement.

Question 7) Dans la classe `Block`, ajouter une fonction membre `MvProdInv` prenant en argument un `const vector<double>&` noté $\boldsymbol{x}$ et un `vector<double>&` noté $\boldsymbol{b}$. Cette fonction membre aura l'effet suivant. Si `C` représente le bloc $\mathrm{R_I B R_J^{\top}}$, alors l'appel `C.MvProdInv(x,b)` calculera le vecteur $\boldsymbol{y} = \mathrm{R}_\mathrm{J}^{\top}\boldsymbol{x}$, résoudra l'équation $\mathrm{B}\boldsymbol{z} = \boldsymbol{y}$ et stockera le résultat dans $\boldsymbol{z}$, puis réalisera l'affectation $\boldsymbol{b} = \mathrm{R_I}\boldsymbol{z}$. Cette opération correspond donc à réaliser le produit matrice-vecteur par la matrice $\mathrm{R_I B^{-1} R_J^{\top}}$.

Ajouter ensuite dans la classe `BlockMatrix` une fonction membre `MvProdInv` qui réalisera de manière analogue le produit matrice vecteur par la matrice $\sum_{q=1}^{Q} \mathrm{R}_{n,\mathrm{I}_q} \mathrm{A}_{\mathrm{I}_q,\mathrm{I}_q}^{-1} \mathrm{R}_{n,\mathrm{I}_q}^{\top}$. Bien évidemment, il conviendra d'utiliser la fonction membre `MvProdInv` de la classe `Block`.

Question 8) L'algorithlme du gradient conjugué préconditionné (PCG) est une variante du gradient conjugué permettant de résoudre l'équation $\mathrm{PA}\boldsymbol{x} = \mathrm{P}\boldsymbol{b}$ en supposant que A et P sont SDP. Il prend la forme de l'algorithme 1 à la page suivante.

Ajouter une fonction membre `pcg` dans la classe `DenseMatrix` qui prend en argument d'entrée un `vector<double>&` noté $\boldsymbol{x}$, un `const vector<double>&` noté $\boldsymbol{b}$ et un `double` noté ϵ, et renvoyant `void`. Cette fonction membre a vocation à résoudre l'équation $\mathrm{A}\boldsymbol{x} = \boldsymbol{b}$ avec l'algorithme 1 en prenant le préconditionneur P donné (3) i.e. par la méthode de Schwarz additive.

Algorithm 1

```
function PCG(A,P,b,ε_TOL)
    ε = ε_TOL |b|_2
    x = 0
    r = b
    z = Pr
    p = z
    while |r|_2 ≥ ε do
        γ = r^⊤ z
        α = γ/|p|_A^2
        x = x + αp
        r = r − αAp
        z = Pr
        β = r^⊤ z/γ
        p = z + βp
    end while
    return (x)
end function
```

Question 9) On s'intéresse maintenant à résoudre le système $\mathrm{A}\boldsymbol{x} = \boldsymbol{b}$ au moyen de l'algorithme du gradient conjugué avec ou sans préconditionneur. Pour effectuer cette comparaison, on pourra utiliser les matrices fournies dans l'archive `matrices2` de l'énoncé de projet (i.e. fichiers `matrixa.txt` avec `a`=07,08,...15) et tirer un vecteur $\boldsymbol{b}$ aléatoirement. On représentera sur la même figure, l'historique de convergence (la norme du résidu au fil des itérations) des deux méthodes itératives : `cg` d'une part, et `pcg` d'autre part. La norme du résidu sera représentée en échelle logarithmique.

Le format dans lequel sont stockées les matrices fournies dans l'archive du projet est le suivant. Si la matrice stockées est $\mathrm{A}_{j,k} \in \mathbb{R}^{n\times m}$, et si on note $a = (a_p)_{p=1,\ldots,n\times m}$ la suite de valeurs $a_{j*m+k} = \mathrm{A}_{j,k}$, alors le fichier aura la structure suivante :

```
#SIZE
n    m

#DATA
```

$$\begin{array}{llll} a_1 & a_2 & \cdots\cdots & a_{20} \\ a_{21} & a_{22} & \cdots\cdots & a_{40} \\ \vdots & & & \vdots \end{array}$$

---

3 Question subsidiaire

Les questions précédentes ont porté sur le cas où la matrice A est dense. Reprenez tout le projet en traitant cette fois le cas où la matrice A est creuse et représentée par une structure de données creuse.