© *J.-B. A. K. <jean-baptiste.apoung@math.u-psud.fr>*

Fiche de TP 2 : Méthodes directes pour matrices creuses

On s'intéresse à la résolution par une méthode directe, des systèmes linéaires $Ax = b$, lorsque la matrice A est creuse. Afin de mener à bien la résolution des exercices proposés, on fera usage des utilitaires :

- *GSL pour le stockage des matrices et vecteurs.*
- *Gnuplot pour la représentation graphique des matrices.*
- `safemira_sparse` *pour la décomposition de certaines matrices creuses*

Une présentation succincte de ces utilitaires est accessible sur Dokeos dans le dossier dédié au présent cours.

Thème - 1 *Existence de factorisation LU de matrice*

Note 1.

Dans cette partie on travaillera avec les matrices pleines. On utilisera l'utilitaire GSL.

Algorithme de résolution de systèmes triangulaires inférieures (à gauche) et supérieures (à droite)

```
-------------------------------------------------------------------------------------
     ALGORITHME DE DESCENTE                 |        ALGORITHME DE REMONTEE
-------------------------------------------------------------------------------------
Donnees: A,b. Resultat:b solution de A x= b| Donnees: A,b. Resultat:b solution de A x=b
-----------------------------------------|-------------------------------------
pour i= 0 à n-1                          | pour i= n-1 à 0
     s= 0                                |      s = 0
     pour j = 0 à i-1                    |        pour j = i+1 à  n-1
       s = s + A(i,j) * b(j)             |             s = s + A(i,j) * b(j)
     fin j                               |        fin
  b(i) = (b(i) - s) /A(i,i)              |    b(i) = (b(i) - s)/A(i,i)
fin                                      |fin
( On remarquera  que le second membre b est  ecrase par la solution)
```

Q-1 : Ecrire une une fonction **`void descenteRemonteeLU (gsl_matrix* A, gsl_vector* b)`**, qui résout un système linéaire $Ax = b$, où la matrice A contient sa factorisation LU : U occupant la diagonale et la partie triangulaire supérieure et L (sans sa diagonale) occupant la partie triangulaire inférieure. On rappelle que la diagonale de L est unité (i.e $L_{i,i} = 1$) et n'est donc pas stockée.

Tester la fonction en générant aléatoirement des matrices triangulaires inférieures et supérieures (voir TP1), de tailles n et en évaluant l'erreur commise dans la résolution.

Q-2 : Ecrire une fonction **`void decompLU (gsl_matrix* A)`** qui effectue la décomposition LU d'une matrice carrée A et l'écrase par les facteurs L et U de sa décomposition, sans stocker la partie diagonale de L. L'algorithme est fourni ci-dessous. La factorisation devra s'interrompre si l'on rencontre un pivot nul.

Algorithme de décomposition A = LU

```
Donnee: A.        Resultat: A contenant L et U
U occupe la partie triangulaire superieure de A et L la partie triangulaire inferieure
L etant à diagonale unite,  sa partie diagonale n'est  pas stockee
-------------------------------------------  | ---------------------------------------------
pour k = 0 à n-1                             |% : introduit un commentaire
  pour i = k+1 à n-1                         |
          A(i,k)= A(i,k)/ A(k,k)             | On devra arreter si A(k,k) est nul
          pour j = k+1 à n-1                 |
             A(i,j)=A(i,j)-A(i,k)*A(k,j)     |
          fin                                |
  fin                                        |
fin                                          |
```

Q-2-1 : Evaluer cette fonction sur la matrice $A = \begin{bmatrix} 2 & 4 & -4 & 1 \\ 3 & 6 & 1 & -2 \\ -1 & 1 & 2 & 3 \\ 1 & 1 & 4 & 1 \end{bmatrix}$.
Conclure en calculant tous les mineurs principaux d'odre $k, k = 1 \ldots 3$ de A.

Q-2-2 : On considère la matrice de permutation $P = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$.
Vérifier que le produit $B = PA$ admet une factorisation LU, en évaluant **`decompLU(B)`**.
Expliquer alors comment résoudre un système linéaire avec la matrice A ci-dessus.

Q-3 : La question précédente met en évidence la nécessité du pivotage (permutation de lignes) dans la factorisation LU de certaines matrices.

Q-3-1 : Montrer que pour toute matrice inversible $A \in \mathcal{M}_n(\mathbb{R})$, il existe une matrice de permutation P telle que la matrice PA admette une factorisation LU. Comparée à la factorisation LU, on a allégé certaines hypothèses sur la matrice A pour que la factorisation $PA = LU$ soit possible. Lesquelles ?

Q-3-2 : Ecrire une fonction, **`void decompLUP(gsl_matrix* A, gsl_vector* p)`**, qui effectue la factorisation LU d'une matrice régulière A selon une méthode de Gauss avec pivotage partiel ; c'est à dire avec une permutation de lignes. Le vecteur p stockera les permutations effectuées sur les lignes. (*L'algorithme est donné ci-dessous*).
On rappelle la relation suivante entre le vecteur p et la matrice P : $P(i,j) = 1$ si $p(j) = i$ et 0 sinon. On rappelle aussi que $P^{-1} = P^T$.

Algorithme de décomposition PA = LU

```
Donnee: A. Resultats: A contenant L,et U   |     Explications des notations et indications
et P vecteur de permutation des lignes     |
-------------------------------------------| ---------------------------------------------
pour i =0 à n-1  P(i) = i fin              |% : introduit un commentaire
pour k = 0 à n -1                          |
  recherche la ligne ipivot du pivot:      |ipivot est telle que |A(ipivot,k)|=max|A(i,k)| k<= i <= n-1
  permuter les lignes P(ipivot) et P(k)    | notation MATLAB pour permuter les lignes P(ipivot) et P(k) :|
  pour i = k+1 à n-1                       |ip=P(ipivot); P(ipivot)=P(k);P(k)= ip;
          A(i,k)= A(i,k)/A(k,k)            |v=A(k,:); A(k,:) = A(ipivot,:); A(ipivot,:)= v;
          pour j = k+1 à n-1               |
          A(i,j) = A(i,j)-A(i,k) * A(k,j)  | GSL dispose d'une fonction qui permute les lignes d'une matrice :
          fin                              | gsl_matrix_swap_rows (A, ipivot, k).
  fin                                      |
```

Q-3-3 : Vérifier le bon fonctionnement de cette fonction en considérant par exemple la matrice A ci-dessus. On pourra résoudre des systèmes linéaires $Ax = b$ avec des b bien choisis (par exemple $b = Ax_0$ avec x_0 donné).
(*On rappelle que pour tout* $v \in \mathbb{R}^n$, $(P^{-1}v)(i) = v(p(i)), i = 0, \ldots, n-1$).

Thème - 2 *Phénomène de remplissage et illustration de la réduction du remplissage*

Note 2.

Dans cette partie on travaillera avec les matrices pleines. On utilisera l'utilitaire GSL. On souhaite ici se convaincre qu'il peut y avoir remplissage (apparition des valeurs non nulles à des positions où il y avait des zeros) dans la factorisation LU d'une matrice creuse. Ce phénomène, assez pénalisant, est moins prononcé suivant la structure de la matrice. Un moyen pour l'éviter est la permutation des lignes et des colonnes (renumérotation) et vise à orienter une matrice donnée vers ces textures moins favorables au remplissage.

Exercice-1 **:** **Evidence du remplissage**

Q-1 **:** Ecrire une fonction qui génère la matrice *"flèche"* suivante : A est de taille n donnée et ses seuls éléments non nuls sont sur sa première ligne, sa première colonne et sa diagonale et valent tous 1 sauf $A(0,0) = n$.

Q-2 **:** Pour diverses valeurs de n, afficher la matrice A (voir Listing3). Afficher aussi la matrice de la décomposition LU de A . Qu'observe-t-on ?

(*On pourra aussi compter et afficher le nombre d'éléments non nuls de la matrice A avant et après la factorisation.*). Commenter.

Exercice-2 **:** **Prise en compte d'une renumérotation et illustration**

On suppose que l'on dispose d'un matrice A et d'un vecteur de permutations $iperm$ telle que la matrice B obtenue après permutation des lignes et colonnes de A est définie par : $B(iperm(i), iperm(j)) = A(i,j), 0 \leq i,j \leq n-1$. Le vecteur $iperm$ est appelé *vecteur de la permutation inverse* ; Il est associé à la matrice P telle que $PAP^T = B$, par la relation $P(i,j) = \delta_{i,iperm(j)}$ où $\delta_{i,j}$ est le symbole de Kronecker : valant 1 si $i = j$ et 0 sinon.

Q-1 **:** Ecrire une fonction **`gsl_matrix * appliquer_inverse_permutation (gsl_matrix * A, int *iperm)`** qui prend en entrée une matrice pleine A et un vecteur de permutation $iperm$ et retourne la matrice B permutée de la matrice A.

Q-2 **:** Ecrire une fonction **`void lire_inverse_permutation (int *iperm, int n, const char *fichier)`** qui prend en arguments un tableau $iperm$ alloué de taille n, un entier n et un nom de fichier et qui lit la permutation inverse inscrite dans ce fichier et le stocke dans le tableau $iperm$. Le fichier contient n lignes et pour $i = 0, \ldots, n-1$, la ligne i contient la valeur de $iperm[i]$.

Q-3 **:** Représenter sur un même graphique la matrice A et sa factorisation LU. Commenter les observations. On utilisera la matrice du Laplacien 1D dont une fonction génératrice est donnée dans le Listing 2

Q-4 **:** Pour une matrice du Laplacien A,

- utiliser la fonction **`void ecrire_metis_graph (gsl_matrix * A, const char *filename)`** (voir Listing 4) pour générer le fichier "graph.txt" du graphe de la matrice A.
- En ligne de commande executer `ndmetis graph.txt` pour obtenir le fichier "graph.txt.iperm" contenant le vecteur de la permutation inverse de la matrice A.
- En utilisant le fichier "graph.txt.iperm", construire la matrice B permutée de A.
- Afficher sur la même figure la matrice B et sa factorisation LU.

Conclure que certaines matrices creuses, pas nécessairement sous forme bande, peuvent avoir une factorisation LU moins sujette au remplissage.

Note 3.

Le but de cette partie est d'exhiber des stockages de matrices favorables à l'utilisation des méthodes directes d'inversion des systèmes linéaires. Il est question de décrire des structures de données qui permettront de stocker les matrices creuses permutées pour une factorisation LU.

Exercice-1 : Matrices bandes (stockage et factorisation)

Q-1 : Pour stocker une matrice de largeur de bande connue, définir une structure de données comme suit :

Listing 1 – struct ture de données pour matrice bande

```
/* @c J.-B. A. K. Cours M325 Calcul Scientifique II 2015 */
typedef struct MatriceBande{
int n;  //  nombre de lignes: c'est une matrice carree
int lbg; // largeur de bande gauche
int lbd; // largeur de bande droite
gsl_vector* a; // tableau de taille (lbg + 1 + lbd) x n
/* Si A est la matrice pleine associee,
  l'element A(i,j) se trouve ici a la position
  a[ (lbg + lbd + 1)* i + lbg + j - i ] i.e. a[ (lbg + lbd) * i + lbg + j ]
  Les seuls elements de la ligne i ont pour numero de colonne j
  compris entre [max(i-lbg,0); min(i+lbd,n-1)]
*/
}MatriceBande;
// Ajouter une fonction d'allocation de resources et de liberation de resources
MatriceBande* matrice_bande_alloue(int n, int lbg,  int lbd);
void matrice_bande_libere(MatriceBande*);
```

Q-2 : Ecrire une fonction **`void pleineVersBande(MatriceBande* B, const gsl_matrix* A)`** qui calcule les demi-largeurs de bande inférieure **`lbg`** et supérieure **`lbd`** d'une matrice pleine A et stocke cette matrice sous format bande dans une matrice bande B.

Ecrire aussi une fonction **`void bandeVersPleine(gsl_matrix* A, const MatriceBande* B)`** qui transforme une matrice stockée sous format bande B en une matrice pleine A.

Q-2-1 : Ecrire une fonction **`void decompLUBande(MatriceBande* B)`**, qui effectue une factorisation LU d'une matrice stockée bande.

Faire varier l'entier n et générer la matrice du laplacien 1D. Récupérer le nombres d'opérations donnés par les factorisations **`decompLUBand`** et **`decompLU`**. Représenter sur un même graphique à l'échelle logarithmique ces nombres d'opérations en fonction de la taille n. Calculer les pentes des droites obtenues et conclure.

Q-2-2 : Ecrire une fonction **`void inverseLUBand(MatriceBande* B, gsl_vector* b)`**, qui résout le système linéaire $Ax = b$ pour lequel la matrice A factorisée LU est stockée sous forme bande. Le vecteur b sera écrasé par la solution.

Q-3 : Réduction de largeur de bande des matrices

Cette question a pour but de montrer qu'on peut transformer, via une renumerotation, une matrice donnée avec beaucoup de zéros, en une matrice de faible lageur de bande. Il est fourni, pour cette question, une fonction **`void rcm(int* iperm, const gsl_matrix* A, int i)`** à travers les fichiers **`rcm.h, rcm.c`**. Pour son appel, il faudra fournir une matrice A pleine de taille n et un indice $0 \leq i \leq n-1$ (on pourra prendre $i = 0$). On a en retour un vecteur $iperm$ de permutation inverse. Ainsi, la matrice permutée de A sera définie par : $B_{ipem(i)iperm(j)} = A_{ij}, 0 \leq i, j \leq n-1$.

Q-3-1 : Tester cette fonction en générant la matrice A du Laplacien 2D et en affichant sur un même graphique la matrice A et sa matrice permutée B. On pourra aussi afficher le nombre d'éléments non nuls des factorisations LU de A et de B. Commenter les observations.

Q-3-2 : Comparer les temps de résolution de systèmes linéaires avec la matrice A et la matrice renumerotée B. On utilisera les fichiers **`cpu_timer.h, cpu_timer.c`**

Listing 2 – Matrice du Laplacien 1D

```
 /* @c J.-B. A. K. Cours M325 Calcul Scientifique II 2015 */
gsl_matrix *
matrice_laplacien1d (int n, double a, double b)
{
  gsl_matrix *A = gsl_matrix_alloc (n, n);
  double h = (b - a) / (n + 1);
  double ih = 1. / (h * h);

  for (int i = 1; i < n - 1; ++i)
    {
      gsl_matrix_set (A, i, i, 2. * ih);
      gsl_matrix_set (A, i, i + 1, -ih);
      gsl_matrix_set (A, i, i - 1, -ih);
    }
  gsl_matrix_set (A, 0, 0, 2 * ih);            gsl_matrix_set (A, 0, 1, -ih);
  gsl_matrix_set (A, n - 1, n - 1, 2. * ih);   gsl_matrix_set (A, n - 1, n - 2, - ih);
  return A;
}
```

Listing 3 – Ecriture du squelette d'une matrice dans un fichier pour gnuplot

```
/* @c J.-B. A. K. Cours M325 Calcul Scientifique II 2015 */
/*!
Utilisation : spy_gnuplot_gsl_matrix (A, "A.txt");
Uisualisation sous gnuplot :
gnuplot> plot "A.txt" w lp
*/
void
spy_gnuplot_gsl_matrix (gsl_matrix * A, const char *fichier)
{
  int n, m, i, j;
  n = A->size1;
  m = A->size2;
  double eps = 1e-32;
  FILE *fid = fopen (fichier, "w");
  for (i = 0; i < n; i++)
    for (j = 0; j < m; j++)
      if (fabs (gsl_matrix_get (A, i, j)) > eps)
        fprintf (fid, "%f  %f\n\n", (float) j, (float) (n - i - 1.));
  fclose (fid);
}
```

Listing 4 – Ecriture du graphe d'une matrice dans un fichier pour metis

```
/* @c J.-B. A. K. Cours M325 Calcul Scientifique II 2015 */
void
ecrire_metis_graph (gsl_matrix * A, const char *filename)
{
  double eps = 1e-32;
  FILE *os = fopen (filename, "w");
  int n, m, i, j, nnz, nb_edges;
  n = A->size1;
  m = A->size2;
  assert (m == n);              //la matrice doit etre carree
  nnz = 0;
  for (i = 0; i < n; ++i)
    for (j = 0; j < m; ++j)
      if (fabs (gsl_matrix_get (A, i, j)) > eps)
        ++nnz;
  nb_edges = (nnz - n) / 2;
  fprintf (os, "%d  %d\n", n, nb_edges);
  for (i = 0; i < n; ++i)
    {
      for (j = 0; j < m; ++j)
        {
          if ((j != i) && fabs (gsl_matrix_get (A, i, j)) > eps)
            fprintf (os, " %d ", j + 1);
        }
      fprintf (os, "\n");
    }
  fclose (os);
}
```