Origanisation de l'UE M325

Table des matières

1 Contenu

L'unité d'enseignement M325 (Calcul scientifique II) s'intitule "*Initiation aux schémas numériques pour les EDPs (Laplacien) et méthodes avancées pour le système A U = f*". Le langage de programmation choisis pour ce cours est le langage C.

Le module M325 vient vient les modules

- M311 : Analyse matricielle et applications
- M312 : Algorithmique et programmation en C
- M315 : Calcul scientifique I : *Méthodes numériques pour l'équation de transport et schémas numériques pour les équations différentielles ordinaires.*

Et se déroule simultanément avec le cours

- M326 : *Projet de calcul scientifique*. Un projet en relation avec une partie du présent cours y est proposé sous le titre **"Méthode de réduction cyclique pour systèmes tridiagonaux : récursion et liste doublement chaînée en langage C"**.

Il est donc nécessaire de le positionner par rapport à ces modules dont il s'inscrit dans la continuité.

Ainsi, pour nous inscrire respectivement dans la continuité

1. **de M312 :** nous proposons d'approfondir les connaissances qui y on été acquises. Pour cela nous faisons le choix de reposer les séances de TPs sous le langage de programmation C. Et pour être dans la continuité de M312, nous envisageons d'explorer les points suivants, *structuration physique et logique dans la programmation en C −− prise en main de bibliothèques écrites en langage C pour le calcul scientifique −− Compilation automatique, et normes de programmations.*
2. **de M315 :** les schémas numériques pour les équations de transport y étant abordés, nous proposons d'aborder les schémas numériques pour problèmes elliptiques stationnaires et non stationnaires. Nous nous limiterons cependant à l'étude des schémas de type différences finies. Rappelons à titre indicatif qu'il existe des méthodes numériques beaucoup plus adaptés pour ces types d'équations. Citons en exemples : la méthode des éléments finis (continus et discontinus), la méthode spectrale, la méthode des différences finies mimétiques, la méthode des volumes finis spectrales etc. Nous n'aborderons pas ces méthodes dont l'étude nécessite des outils mathématiques pour l'instant hors de notre portée (L3). Un projet est proposé en M326 sous le titre **"Calcul scientifique aux services d'une équation aux dérivées partielles"** et permettant de découvrir entre autres la méthode de type volumes finis et comprendre comment ces méthodes initialement prévues pour les problèmes hyperboliques (*i.e du type transport*) peuvent être adaptées pour des problèmes elliptiques (*i.e. du type diffusion*).
3. **de M311 :** les méthodes directes pour matrices pleines et les méthodes itératives de base y étant abordées, nous envisageons ici, d'explorer **les méthodes directes pour matrices creuses −− les méthodes itératives de type projection et leur préconditionnement**. En l'occurrence nous définirons et analyserons les **méthodes de type Krylov**, dans les cas suivants : *la matrice est symétrique définie positive, la matrice est symétrique mais pas définie positive, la matrice est non-symétrique mais de partie positive définie positive, la matrice est non-symétrique et non définie positive.*

   Ces méthodes reposant uniquement sur le calcul du produit matrice-vecteur sont bien adaptées pour les matrices creuses. Le préconditionnement de ces méthodes fera essentiellement usage des méthodes itératives de bases vues en M311 : (jacobi, sor, ssor, approximations tridiagonales, factorisations LU/Cholesky incomplètes etc.)

   La simple disposition du produit matrice vecteur est suffisante pour développer, dans le contexte des matrices creuses, des méthodes efficaces de **calcul des valeurs/vecteurs propres**. Nous explorerons quelques unes d'entre-elles.

   Le préconditionnement peut être vu comme un moyen de rendre la résolution du système linéaire indépendante de sa taille et donc rapide lorsque la taille du système devient grande.

   Dans cette direction, d'autres méthodes existent, dont certaines implicitent le préconditionnement, comme les méthodes multi-grilles (que nous verrons) et les méthodes de décomposition de domaines

(que nous ne verrons pas). D'autres par contre s'en affranchissent, comme les méthodes de réduction cyclique et de transformation de Fourrier discrètes que nous verrons.

La suite du document décrit l'organisation envisagée des séances de cours et de TPs, les modalités d'évaluation et fournit les références bibliographiques et des liens vers les utilitaires pour ce cours. Une description succincte de ces utilitaires est fournie dans des documents annexes.

2 Organisation

2.1 Contenu et répartition

L'unité d'enseignement M325 s'étale sur 6 semaines. Avec 4 h 30 de cours intégré et 4 h de Tp par semaine.

Les TPs dans leurs contenus essaieront de suivre ceux du cours des semaines concernées. Chaque séance de Tp se déroulera en deux phases : dans une première on essaiera d'implémenter les algorithmes ou méthodes vus en cours et dans une second phase on fera usage de certains utilitaires (voir Section 3.2) pour des validations ou des comparaisons.

Les semaines et les contenus sont organisés comme suit

- Semaine 1 : du 03/03/15 au 06/03/15 **attention pas de TP le 03 /03/15**.
  - Différences finies pour le laplacien (1D-2D) et pour l'équation de la chaleur.
  - TPs (production des systèmes linéaires), laplacien 2D par différences finies. Calculs d'erreurs et ordre de la méthode. Structuration physique et compilation automatique. Utilisations de **`GSL`**, **`CSparse`**, **`Gnuplot`** (*sans rentrer dans les détails des méthodes d'inversion des systèmes linéaires inclus*).
- Semaine 2 : du 16/03/15 au 20/03/15
  - Cours : Méthodes directe pour matrices creuses
  - TP : avec les utilitaires **`GSL`** (factorisation des matrices pleines, mise en évidence du remplissage), renumérotation avec **`CSparse`** et **`METIS`**, factorisation avec **`safemira_sparse`**. Applications et comparaisons.
- Semaine 3 et 4 : 23/03/15 au 03/04/15
  - Cours : Méthodes itératives adaptées aux matrices creuses : Méthodes de type gradients. Méthode de projection sur les espaces de Krylov (GC, GMRES), Préconditionnement.
  - TP : Programmer les algorithmes (explorer d'autres QMR, TFQMR, CGS, BiCGStab). Preconditionnement : Jacobi, Gauss-Seidel, ILU(0), ILUT. Sur un problème modèle de convection-diffusion-réaction, choisir la méthode et le préconditionneur les plus adaptés. Utilitaires engagés : **`GSL`** ou **`CSparse`**.
- Semaine 5 : 06/04/15 au 10/04/15 (**pas de cours le lundi 06/04**)
  - Cours : Méthodes rapides pour matrices de structure particulières : FFT pour le laplacien 2D, réduction cyclique pour systèmes tridiagonaux 1D, introduction aux Multi-grilles géométriques.)
  - TP : mise en oeuvre FFT (schémas 3 pts 1D et 5 points 2D), Multi-grilles (Schéma 9 pts 2D) Réduction cyclique schémas 3 points 1D. Comparaisons. Utilitaires engagés : **`GSL`** peut-être **`FFTW`**.
- Semaine 6 : 13/04/15 au 17/04/15
  - Cours : Calcul de valeurs et vecteurs propres pour matrices creuses : Lanczos
  - TP : avec **`GSL`** ou **`CSparse`**(mise en oeuvre des algorithmes). Construction de **Wrapper** en C autour de la bibliothèque Fortran **`Arpack`**. Applications.

Note 1.

- Semaine du 20/04/15 au 24/04/15 : TP mais pas cours le lundi. Cours mais pas TP le jeudi.
- Semaine du 27/04/15 : cours et TP seulement le lundi.

2.2 Dates et modalités dévaluation

Un examen final sur machine avec une partie théorie aura lieu la semaine du 14/05/2015 et comptera pour $3/4$ de la note finale.

Deux tests en TPs auront lieu respectivement début 4-ième semaine et fin 6-ième semaine ; Leur moyenne comptera pour $1/4$ de la note finale. En résumé on a schématiquement

Note 2 (Modalités d'évaluation)**.**

Session 1 : $F = \frac{3}{4}E + \frac{1}{4}CC$.
Avec $CC = \frac{1}{2}CC1 + \frac{1}{2}CC2$.
Les $CCi, i = 1, 2$ sont les notes des tests effectués en Tps.

3 Quelques références bibliographie

3.1 Les supports livres

Les supports que nous proposons ici ne sont pas les seuls dans le domaine. N'hésitez pas à consulter et comparer avec d'autres références que vous trouverez.

1. Pour le langage C :
   — R. Malgouyres, R. Zrour, F. Feschet : *Initiation à l'algorithme et à la programmation en C. Bon pour les structures de données en C.*
   — J-L Imber : *Algorithmes fondamentaux en langage C. Regarder annexes A1 et A2 pour les makefiles.*
   — J-M Lery : *Algorithmique Application en C. Bon pour les structures de données, table de hachage etc..*
2. Pour la discrétisation des équations aux dérivées partielles, nous conseillons :
   — *Equations aux dérivées partielles et leurs applications*. Brigite Lucquin. *Pour les différences finies , voir à partir de la page 152*
   — *Analyse numérique des équations aux dérivées partielles*. Laurent Di Menza.
3. Pour la résolution des systèmes linéaires
   — G. H Golub, G. A. Meurant : *Résolution numérique des grands systèmes linéaires*.
   — C. Brezinski, M. Redivo-Zaglia : *méthodes numériques itératives*.
   — P.G. Ciarlet : *Introduction à l'analyse numérique matricielle et à l'optimisation*.
   — G. Allaire, S. M. Kaber : *Algèbre linéaire numérique.*
   — Y. Saad : *Iterative Methods for Sparse Linear Systems*.
   — K. Chen : *Matrix Preconditioning Technique and Applications*.
   — T. A. Davis : *Direct methods for Sparse linear systems*.
4. Pour les méthodes rapides pour systèmes linéaires
   — W. HackBusch : *Multi-grid methods and Applications*.
   — Certaines références déjà mentionnées ci-dessus (K.Chen).

3.2 Les utilitaires

Les séances de Travaux pratiques se dérouleront dans le langage de programmation C. Les bases de ce langage sont supposées acquises.

Cependant nous n'aurons pas les mêmes facilités que sous `Matlab` où les *solveurs* intégrés et un bon outil graphique permettaient de valider assez rapidement la pertinence et la mise en oeuvre des algorithmes.

Afin de mener à bien nos séances de TPs, et nous assurer que lors de celles-ci, nous ne perdrons pas du temps dans la création des structures de données pour manipuler les vecteurs et les matrices ou pour représenter graphiquement les solutions calculées, il nous semble judicieux de disposer d'un certain nombres d'utilitaires, faciles à prendre en main, écrits en langage C, et utilisés comme outils de chevet par certains spécialistes du calcul scientifique. Ces utilitaires se doivent donc d'être légers et accessibles.

Pour le présent cours, nous ferons essentiellement usage des utilitaires :

- **GSL** `http/www.gnu.org/software/gsl`, pour tout ce qui concerne la manipulation des matrices pleines. Il sera en fait presque notre environnement de travail, de par la richesse des structures qu'il fournit.
- **CSparse** `http://www.cise.ufl.edu/research/sparse/CSparse/CSparse.tar.gz`, pour tout ce qui est méthodes directes pour matrices creuses. Nous n'emploierons pas la démarche qui a fait son fondement, puisque nous ne plongerons pas dans la théorie des graphes qui la sous-tende. Nous emploierons une démarche qui permettra de jumeler l'analyse symbolique et la factorisation numérique. Pour cette fin nous aurons recours à une structure dynamique, basée sur les listes chaînées. Nous avons pour cela développé une structure de matrice creuse que nous avons appelée **safemira_sparse**. Elle permettra de comprendre et c'est ce qui est essentiel, les phénomènes de remplissage dans les méthodes directe pour matrices creuses et la nécessité des méthodes et techniques de renumérotation qui précède l'analyse symbolique. Néanmoins nous tirerons partie des algorithmes de renumérotation présents dans la librairie CSparse. Dans cette direction, on verra l'apport des outils adaptés aux traitements des graphes comme **METIS** dans la renumérotation.
- **Gnuplot** `http://www.gnuplot.info`. Cet outils a été l'outil que nous avons retenu pour la représentation graphique.

Nous fournissons des notes décrivant succinctes ces utilitaires et illustrant par de brefs exemples l'utilisation que nous en ferons.

Note 3 (Attention)**.**

Les utilitaires mentionnés ci-dessus et utilisés pour ce cours sont déjà disponibles sur les machines des salles informatiques. Il n'est donc pas nécessaire de les télécharger. Ainsi les liens *web* vers ces utilitaires, fournis dans les différents supports du présent cours ne le sont qu'à titre indicatif.

3.3 Scripts partiels et leur bien fondé

Il arrivera durant les séances de TD sur machine, que des bouts de scripts en langage C vous soient transmis à travers Dokéos. Il en sera ainsi pour des raisons suivantes :

- Faciliter la saisie des bouts de programmes nécessaires mais dont l'importance est en marge de l'idée principale de la séance de TP en cours.
- Guider sur la structuration et les normes de programmation à mettre en place afin d'éviter toute dispersion et par conséquent focaliser la séance de TP en cours sur le thème porteur du cours de la semaine.

Nous fournissons ici deux exemples de scripts fournis

3.3.1 Quantifier les temps d'exécution

Listing 1 – cpu_timer.h : Intercace utilisateur du chronomètre

```
 /*
   M325 : Calcul Scientifique II
   L3 MINT  Univ. Paris Sud ORSAY

   Copyright (C) 2015 APOUNG KAMGA Jean-Baptiste

   This is part of S.A.F.E.M.I.R.A TOOLS

   This program is free software; you can redistribute it and/or modify
   it under the terms of the GNU General Public License as published by
   the Free Software Foundation; either version 2, or (at your option)
   any later version.

   This program is distributed in the hope that it will be useful,
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
   GNU General Public License for more details.

   You should have received a copy of the GNU General Public License
   along with this program; if not, write to the Free Software Foundation,
   Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
*/


#ifndef CPU_TIMERE_H
#define CPU_TIMERE_H

#define MAX_TIMER_SIZE   30

/*=====================================================================
//
//    Auteur: J.-B. APOUNG KAMGA
//    Date:  22 / 03 /2015
//    Time:  18:14
//    Modif: 08:50  22/03/2015 (ajout commentaires corection typos)
//=====================================================================*/
/*!
  Utilisation :

  \code
   int  id1 = cpu_timer_tic();

   charger_fichier();

   int  id2 = cpu_timer_tic();

   lire_matrice_au_format_coo();

   int  id3 = cpu_timer_tic();

   convertir_matrice_au_format_csr();

   double t1 =  cpu_timer_id_toc(id1);

   double t2 =  cpu_timer_id_toc(id2);

   double t3 =  cpu_timer_id_toc(id3);

   printf("Temps mis pour chargement de fichier, lecture et conversion de la matrice: %f ↩
       secondes  \n",t1);

   printf("Temps mis pour lecture et conversion de la matrice: %f secondes \n",t2);

   printf("Temps mis pour conversion de la matrice : %f secondes \n",t3);

  \endcode
  */


/*!
 * \brief é    Dclenche un èchronomtre
 * \return    id, l'unique entier identifiant cet instant de édclenchement
 * \Note :    On peut édclencher plusieurs èchronomtres, chaque édclenchement
 * é           tant éidentifi par son id éretourn. Le nombre maximum
```

```
 *            de édclenchements éautoriss est \a MAX_TIMER_SIZE  (ici 30)
 */
int cpu_timer_tic();

/*!
 * \brief êArrte le èchronomtre et retournr le temps éécoul depuis le dernier
 * é       dclenchement du èchronomtre.
 * \return Temps éécoul ;
 */
double cpu_timer_toc();

/*!
 * \brief êArrte le èchronomtre et retourne le temps éécoul depuis un certain édclenchement
 *        du èchronomtre.
 * \param id  Identifiant l'instant du édclenchement du èchronomtre.
 * \return Temps éécoul
 */
double cpu_timer_id_toc(int id);

#endif
```

Listing 2 – cpu_timer.c : Implémentation du chronomètre

```
/*
  M325 : Calcul Scientifique II
  L3 MINT  Univ. Paris Sud ORSAY

  Copyright (C) 2015 APOUNG KAMGA Jean-Baptiste

  This is part of S.A.F.E.M.I.R.A TOOLS

  This program is free software; you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation; either version 2, or (at your option)
  any later version.

  This program is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program; if not, write to the Free Software Foundation,
  Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
*/

#include "cpu_timer.h"
#include <time.h>
#include <stdio.h>


static
struct timer{
 int id;
 double snapshot_id[MAX_TIMER_SIZE];
}Timer = {-1,{0}};


double CPUtime()
{
#ifdef SYSTIMES
  struct tms buf;
  if(times(&buf) != -1)
    return ((double)buf.tms_utime +
            (double)buf.tms_stime) / (long)sysconf(_SC_CLK_TCK);
  else {;
  }
#endif
  return ((double)clock()) / (long double)CLOCKS_PER_SEC;
}

int cpu_timer_tic(){
  double t = CPUtime();
```

```
  Timer.id++;
  Timer.snapshot_id[Timer.id] = t;
  return  Timer.id;
}

double cpu_timer_toc()
{
  double elapse = CPUtime();
  if (Timer.id == -1){
    fprintf(stderr," You have never called tic !!!!!! \n");
    fprintf(stderr," We do not guarantee the accuracy of the output \n");
    Timer.snapshot_id[Timer.id = 0 ] =  elapse;
    return 0;
  }
   elapse -= Timer.snapshot_id[Timer.id];
  return elapse;
}

double cpu_timer_id_toc( int id )
{
  double elapse = CPUtime();
  int valid_id;
  if (Timer.id == -1){
    fprintf(stderr," You have never called tic !!!!!! \n");
    fprintf(stderr," We do not guarantee the accuracy of the output \n");
    Timer.snapshot_id[Timer.id = 0 ] =  elapse;
    return 0;
  }
  if (id < 0){
    fprintf(stderr,"Your have provided a bad id !!!!!!! \n");
    fprintf(stderr,"We are returning the elapse time from the last call of timer_tic() \n");
     valid_id = Timer.id;
  }else{
    valid_id = id;
  }
   elapse -= Timer.snapshot_id[valid_id];
  return elapse;
}
```

3.3.2 Implémenter une renumérotation (Cuthill-Mackey directe ou inverse) d'une matrice creuse, en utilisant une structure de graphe.

Ceci est un exemple de situation en marge, puisque non seulement la notion de graphe n'est pas vue, mais aussi l'utilité de la renumérotation pour la séance vise simplement à exhiber l'effet de la renumérotation sur le remplissage pendant la factorisation. Néanmoins on voudrait disposer de cette numérotation non seulement pour la suite du Tp, mais aussi dans une bibliothèque que nous souhaitons développer aux termes du cours. Dans les listings qui suivent il est donc question de bien comprendre la notion d'interface (c'est-à-dire ce que l'on met à disposition de l'utilisateur) et comment celle-ci diffère de l'implémentation où l'on peut être amené à définir des structures lourdes et pas forcément nécessaire pour l'utilisateur.

Listing 3 – rcm.h : Intercace utilisateur de l'algorithme RCM (Reverse Cuthill-Machkey)

```
/*
   M325 : Calcul Scientifique II
   L3 MINT  Univ. Paris Sud ORSAY

   Copyright (C) 2015 APOUNG KAMGA Jean-Baptiste

   This is part of S.A.F.E.M.I.R.A TOOLS

   This program is free software; you can redistribute it and/or modify
   it under the terms of the GNU General Public License as published by
   the Free Software Foundation; either version 2, or (at your option)
   any later version.

   This program is distributed in the hope that it will be useful,
```

```
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
   GNU General Public License for more details.

   You should have received a copy of the GNU General Public License
   along with this program; if not, write to the Free Software Foundation,
   Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
*/

#ifndef RCM_H
#define RCM_H

#include <stdio.h>
#include <stdlib.h>
#include <gsl/gsl_linalg.h>
/*=====================================================================
//
//    Auteur: J.-B. APOUNG KAMGA
//    Date:  22 / 03 /2015
//    Time:  18:14
//    Modif: 08:32  22/03/2015 (ajout commentaires correction typos)
//=====================================================================*/
/*!
 * \brief rcm  Construit la renumerotation de
 *  Cuthill-Mckee inverse d'une matrice
 *
 * \param pinv tableau de la renumerotation
 * \param A  la matrice  ici pleine
 * \param i  le numero de édpart. Non éutilis dans
 * l'éimplmentation qui suit puisqu'on prend pour
 * indice initial celui de plus petit édegr
 * (plus petit nombre de voisins)
 *
 * \details Calcule pour une matrice édonne \a A
 * la permutation inverse  \a pinv
 * telle que la matrice épermute \a B
 * édfinie par B(pinv(i),pinv(j)) = A(i,j), 0 <= i, j < n
 * èpossde une plus faible largeur de bande.
 */
void rcm (int *pinv, const gsl_matrix * A, int i);

#endif
```

Listing 4 – rcm.c : Implémentation de l'algorithme RCM

```
 /*
   M325 : Calcul Scientifique II
   L3 MINT  Univ. Paris Sud ORSAY

   Copyright (C) 2015 APOUNG KAMGA Jean-Baptiste

   This is part of S.A.F.E.M.I.R.A TOOLS

   This program is free software; you can redistribute it and/or modify
   it under the terms of the GNU General Public License as published by
   the Free Software Foundation; either version 2, or (at your option)
   any later version.

   This program is distributed in the hope that it will be useful,
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
   GNU General Public License for more details.

   You should have received a copy of the GNU General Public License
   along with this program; if not, write to the Free Software Foundation,
   Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
*/

#include "rcm.h"

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
```

```c
#include <assert.h>
#include <string.h>
#include <gsl/gsl_linalg.h>

/*////////////////////////////////////////////////////////////
//
//              STRUCTURE DE DONNEES
//
////////////////////////////////////////////////////////////*/
typedef struct Sommet
{
  int num;
  int d;
  int est_visite;
  struct Arc *arcs;
} Sommet;
Sommet *sommet_creer (int num);
void sommet_libere (void *sommet);
void sommet_insere_arc (Sommet * ps, Sommet * cible);
//============================================================
typedef struct Arc
{
  Sommet *extremite;
  struct Arc *suivant;
} Arc;
Arc *arc_cree ();
void arc_libere (void *parc);
void arc_insere (Arc ** r, Arc * s);
void arc_insere_queue (Arc * r, Arc * s);
int  arc_size (Arc * a);
//============================================================
typedef struct Graphe
{
  int ns;
  Sommet **sommets;
  int est_initialise;
} Graphe;
void graphe_libere (void *pg);
void graphe_cree (Graphe * pg, const gsl_matrix * A);
int  graphe_rcm (Graphe * pg, int *p);
//============================================================
/*////////////////////////////////////////////////////////////
//
//              IMPLEMENTATION :  SOMMET
//
////////////////////////////////////////////////////////////*/
Sommet *
sommet_creer (int num)
{
  Sommet *s = (Sommet *) malloc (sizeof (Sommet));
  s->num = num;
  s->d = 0;
  s->est_visite = 0;
  s->arcs = NULL;
  return s;
}
/*------------------------------------------------------------*/
void
sommet_libere (void *ps)
{
  Sommet *p = (Sommet *) ps;
  if (p)
    {
      if (p->arcs)
        arc_libere (p->arcs);
      free (p);
      p = NULL;
    }
}
/*------------------------------------------------------------*/
void
sommet_insere_arc (Sommet * ps, Sommet * cible)
{
  Arc *a = arc_cree ();
```

```
  a->extremite = cible;
  arc_insere (&ps->arcs, a);
}
/*////////////////////////////////////////////////////////////////
//
// IMPLEMENTATION :    ARC
//
////////////////////////////////////////////////////////////////*/
Arc *
arc_cree ()
{
  Arc *res = (Arc *) malloc (sizeof (Arc));
  res->extremite = NULL;
  res->suivant = NULL;
  return res;
}
/*----------------------------------------------------------------*/
void
arc_libere (void *parc)
{
  Arc *p = (Arc *) parc;
  if (p)
    {
      arc_libere (p->suivant);
      free (p);
      p = NULL;
    }
}
/*----------------------------------------------------------------*/
void
arc_insere (Arc ** r, Arc * s)
{
  if (*r == NULL)
    {
      *r = s;
      return;
    }
  if ((*r)->extremite->d > s->extremite->d)
    {
      Arc *temp = *r;
      *r = s;
      (*r)->suivant = temp;
      return;
    }
  return arc_insere (&(*r)->suivant, s);
}
/*----------------------------------------------------------------*/
void
arc_insere_queue (Arc * r, Arc * s)
{
  if (r->extremite->num == s->extremite->num)
    return;
  if (r->suivant)
    {
      arc_insere_queue (r->suivant, s);
    }
  else
    {
      r->suivant = arc_cree ();
      r->suivant->extremite = s->extremite;
    }
}
/*----------------------------------------------------------------*/
int
arc_size (Arc * a)
{
  return (a == NULL) ? 0 : (1 + arc_size (a->suivant));
}
/*////////////////////////////////////////////////
//
//        IMPLEMENTATION :  GRAPH
//
////////////////////////////////////////////////*/
void
```

```
graphe_libere (void *pg)
{
  int k;
  Graphe *p = (Graphe *) pg;
  if (p->sommets)
    {
      for (k = 0; k < p->ns; ++k)
        sommet_libere (p->sommets[k]);
      free (p->sommets);
      p->sommets = NULL;
    }
}
/*------------------------------------------------------------------*/
void
graphe_cree (Graphe * pg, const gsl_matrix * A)
{
  int i, j;
  int n = A->size1;
  int ny = A->size2;

  #warning VALEUR FIXE ICI

  double epsi = 1e-32;
  assert (n == ny);
  pg->ns = n;
  pg->sommets = (Sommet **) calloc (pg->ns, sizeof (Sommet *));
  pg->est_initialise = 1;

  for (i = 0; i < n; i++)
    {
      pg->sommets[i] = sommet_creer (i);
      pg->sommets[i]->est_visite = 0;

      for (j = 0; j < n; j++)
        {
          if ((j != i) && fabs (gsl_matrix_get (A, i, j)) > epsi)
            {
              pg->sommets[i]->d++;
            }
        }
    }
  for (i = 0; i < n; i++)
    {
      for (j = 0; j < n; j++)
        {
          if ((j != i) && fabs (gsl_matrix_get (A, i, j)) > epsi)
            {
              sommet_insere_arc (pg->sommets[i], pg->sommets[j]);
            }
        }
    }
}
/*------------------------------------------------------------------*/
int
graphe_rcm (Graphe * pg, int *p)
{
  int n = pg->ns;
  int i, j;
  Arc *a;
  int store = 0;
  int id = 0;
  int pid = pg->sommets[id]->d;
  //
  for (i = 1; i < n; ++i)
    {
      if (pg->sommets[i]->d < pid)
        {
          id = i;
          pid = pg->sommets[i]->d;
        }
    }
  a = arc_cree ();
  a->extremite = pg->sommets[id];
  while (a != NULL)
```

```
    {
      if (a->extremite->est_visite == 0)
        {
          p[store++] = a->extremite->num;
          a->extremite->est_visite = 1;
          Arc *c = a->extremite->arcs;
          for (; c != NULL; c = c->suivant)
            {
              if (c->extremite->est_visite == 0)
                {
                  arc_insere_queue (a, c);
                }
            }
          Arc *v = a;
      a = a->suivant;
          free (v);
      //
          if (a == NULL)
            {
              for (j = 0; j < n; j++)
                if (pg->sommets[j]->est_visite == 0)
                  {
                    a = arc_cree ();
                    a->extremite = pg->sommets[j];
                  }
            }
        }
    }
  //
  int *q = (int *) malloc (n * sizeof (int));
  for (i = 0; i < n; ++i)
    q[i] = p[n - 1 - i];
  for (i = 0; i < n; ++i)
    p[q[i]] = i;
  free (q);
  return *p;
}
/*//////////////////////////////////////////////////////////////////
//
//        IMPLEMENTATION :  RCM
//
//////////////////////////////////////////////////////////////////*/

/**================ FONCTION PRINCIPALE ==========================*/
void
rcm (int *pinv, const gsl_matrix * A, int i)
{
  Graphe g;
  graphe_cree (&g, A);
  graphe_rcm (&g, pinv);
  graphe_libere (&g);
}
```