© *Jean-Baptiste APOUNG KAMGA <jean-baptiste.apoung@math.u-psud.fr>*

L'utilitaire CSparse

Thème - 1 *Motivations*

La librairie CSparse accessible à l'adresse `http://www.cise.ufl.edu/research/sparse/CSparse/CSparse.tar.gz` est un modèle de librairie *légère*. Elle est écrite en langage C par Timothy A. Davis et a servi de base pour la librairie CXSPARSE et par la suite la librairie UMFPACK. Elle est à l'origine du *backslash* `"\"` de `Matlab` , dont l'efficacité n'est plus à démontrer.

Pour le présent cours, nous utilisons la librairie CSparse pour les raisons suivantes :

- **Elle est écrite en C** : Présentée sous forme de bibliothèque elle rentre bien dans le cadre de notre cours, dont l'un des objectifs est de s'inscrire dans la continuité de l'UE Math 312 :Älgorithmique et programmation en C∴ Elle nous permettra ainsi de découvrir à une moindre échelle une structuration potentielle d'une bibliothèque C.
- **Elle est légère en anglais lightweigth** : Elle fournit un seul fichier entête : **`cs.h`** contenant les structures de données et les prototypes des fonctions utilitaires et un ensemble de fichier au format **`.c`** dont la compilation permettra de générer la librairie (fichier **`.a`**).
- **Elle est efficace et pédagogique** : elle présente l'essentiel de ce qui se fait dans les *gros* codes de méthodes directes pour systèmes linéaires creux (renumérotation, factorisation symbolique et numérique, graphe d'élimination . . .). Et ceci d'un point de vue mise en oeuvre informatique qui permet de déceler les difficultés qu'un pseudo-code passe de manière involontaire sous silence.
- **Il est crédible et simple d'utilisation** : il est écrit par celui qui a pensé et mis en oeuvre le fameux **backslash** de Matlab et a été l'une des première mouture de UMFPACK dont on ne saurait faire l'éloge.

Par contre comme le dit l'auteur lui-même, et c'est ce que nous faisons, CSparse est fourni seulement à titre pédagogique, et ne doit pas être utiliser pour les codes de production.

Thème - 2 *Description*

Dans le présent document nous décrivons brièvement la partie dont nous aurons besoin et illustrons dans la section suivante un exemple d'utilisation en soulignant bien-sûre les guides de programmation en C (qui apparaîtrons de manière implicite) dont nous souhaitons pour ce cours.

CSparse fournit pour l'utilisateur essentiellement une seule structure de données **`cs_sparse`** qui représente une matrice creuse au format coordonnées (COO) encore appelé en anglais **"triplet format"** ou compressé suivant les colonnes (en anglais **Compressed Sparse Column (CSC)**)

Listing 1 – Structure de données pour la matrice creuse

```
/* --- primary CSparse routines and data structures ------------------------- */
typedef struct cs_sparse    /* matrix in compressed-column or triplet form */
{
    csi nzmax ;     /* maximum number of entries */
    csi m ;         /* number of rows */
    csi n ;         /* number of columns */
    csi *p ;        /* column pointers (size n+1) or col indices (size nzmax) */
    csi *i ;        /* row indices, size nzmax */
    double *x ;     /* numerical values, size nzmax */
    csi nz ;        /* # of entries in triplet matrix, -1 for compressed-col */
} cs ;
```

Les principales fonctions permettant de manipuler et d'utiliser cette structure sont données dans les Listing2 ci-dessous. Nous renvoyons à la section suivante pour une illustration, plus particulièrement les Remark0.0.1 et Remark0.0.2.

Listing 2 – Fonctions principales de CSparse

```
 /*!
  * \brief  C = alpha*A + beta*B
  * \param A
  * \param B
  * \param alpha
  * \param beta
  * \return C
  */
cs *cs_add (const cs *A, const cs *B, double alpha, double beta) ;

/*!
  * \brief  x=A\b where A is symmetric positive definite; b overwritten with solution
  *
  * Resout le systeme lineaire en utilisant la factorisation Cholesky
  * Afin de reduire le remplissage de la matrice pendant la factorisation,
  * les lignes et colonnes sont prealablement permutees; on parle de renumerotation
  *
  * \param order  type de la renumerotation:
  *               2 pour la facorisation LU 1 pour Cholesky et 3 pour la factorisation QR
  * \param A
  * \param b
  * \return
  */
csi cs_cholsol (csi order, const cs *A, double *b) ;
/*!
  * \brief C = compressed-column form of a triplet matrix T
  * Les factorisations n'ont lieu que si le format est de type csc
  * Aini, si une matrice est au formet triplet ou COO, il faut appeler cette fonction.
  * \param T
  * \return
  */
cs *cs_compress (const cs *T) ;
/*!
  * \brief remove duplicate entries from A
  * \param A
  * \return
  */
csi cs_dupl (cs *A) ;
/*!
  * \brief add an entry to a triplet matrix
  * \param T
  * \param i
  * \param j
  * \param x
  * \return 1 if ok, 0 otherwise
  */
csi cs_entry (cs *T, csi i, csi j, double x) ;
/*!
  * \brief y = A*x+y
  * \param A
  * \param x
  * \param y
```

```
 * \return
 */
csi cs_gaxpy (const cs *A, const double *x, double *y) ;
/*!
 * \brief load a triplet matrix from a file
 * \param f
 * \return
 */
cs *cs_load (FILE *f) ;
/*!
 * \brief x=A\b where A is unsymmetric; b overwritten with solution
 *
 * Resout le systeme lineaire en utilisant une factorisation LU
 * Afin de reduire le remplissage de la matrice pendant la factorisation,
 * les lignes et colonnes sont prealablement permutees on parle de renumerotation
 *
 * \param order  type de la renumerotation:
 *               2 pour la facorisation LU 1 pour Cholesky et 3 pour la factorisation QR
 * \param A
 * \param b
 * \param tol
 * \return
 */
csi cs_lusol (csi order, const cs *A, double *b, double tol) ;
/*!
 * \brief C = A*B
 * \param A
 * \param B
 * \return A*B
 */
cs *cs_multiply (const cs *A, const cs *B) ;
/*!
 * \brief 1-norm of a sparse matrix = max (sum (abs (A))), largest column sum
 * \param A
 * \return norme de A
 */
double cs_norm (const cs *A) ;
/*!
 * \brief     print a sparse matrix
 *
 *
 * \param A
 * \param brief
 * \return
 */
csi cs_print (const cs *A, csi brief) ;
/*!
 * \brief x=A\b where A can be rectangular; b overwritten with solution
 * \param order   type de la renumerotation
 *               2 pour la facorisation LU 1 pour Cholesky et 3 pour la factorisation QR
 * \param A
 * \param b
 * \return
 */
csi cs_qrsol (csi order, const cs *A, double *b) ;
/*!
 * \brief  transpose la matrice A:  C = A'
 * \param A
 * \param values
 * \return C
 */
cs *cs_transpose (const cs *A, csi values) ;
/* utilities */
void *cs_calloc (csi n, size_t size) ;
/*!
 * \brief cs_free : wrapper for free
 * \param p
 * \return  NULL to simplify the use of cs_free
 */
void *cs_free (void *p) ;
/*!
 * \brief cs_realloc : wrapper for realloc
 * \param p
 * \param n
```

```
 * \param size
 * \param ok
 * \return  return original p if failure
 */
void *cs_realloc (void *p, csi n, size_t size, csi *ok) ;
/*!
 * \brief cs_spalloc : allocate a sparse matrix (triplet form or compressed-column form)
 * \param m
 * \param n
 * \param nzmax
 * \param values
 * \param triplet  1 pour format triplet et 0 pour format csc
 * \return A avec
 *  A->nz = 0 pour triplet A->nz = -1 pour csc
 */
cs *cs_spalloc (csi m, csi n, csi nzmax, csi values, csi triplet) ;
/*!
 * \brief cs_spfree : free a sparse matrix
 * \param A
 * \return
 */
cs *cs_spfree (cs *A) ;
/*!
 * \brief cs_sprealloc : change the max # of entries sparse matrix
 * \param A
 * \param nzmax
 * \return
 */
csi cs_sprealloc (cs *A, csi nzmax) ;
/*!
 * \brief cs_malloc : wrapper for calloc
 *  Un tableau de taille max(n,1) et cree
 * \param n
 * \param size
 * \return
 */
void *cs_malloc (csi n, size_t size) ;
```

Thème - 3 *Illustrations*

En guise d'illustration nous allons déterminer une solution approchée par différences finies de l'équation aux dérivées partielles

$$\left\{ \begin{array}{rcll} -u'' & = & 1 & \text{dans }]0,1[\\ u(0) & = & 0 & \\ u(1) & = & 0 & \end{array} \right. \tag{1}$$

Sur un maillage uniforme de $[0,1]$ de pas $h = \frac{1}{N+1}$, on sait que le système linéaire obtenu $A\,U = F$ est tri-diagonal.

Mais afin d'illustrer les possibilités de la bibliothèque CSparse, nous allons plutôt résoudre le système pour $F = A\,U_0$ et comparer la solution U obtenue avec U_0. La résolution du système linéaire issu de la discrétisation de (1) se fera sans difficulté majeure.

Les Listing6 et Listing4 illustrent l'utilisation de cette bibliothèque. Le Listing4 n'est qu'une reprise du précédent (Listing6) avec une attention portée sur le type de structuration que nous espérons de la part des étudiants aux termes de ce cours.

Listing 3 – csparse_illustration1.c

```
#include <stdio.h>
#include <cs.h>
```

```c
// Pour la compilation
// gcc -std=c99  -I../CSparse/Include exemple1.c  -L../CSparse/Lib -lcsparse -o exemple1 -lm
/*!
On va resoudre le èsytme : A x  = b
On choisira b de sorte que la solution exacte soit x =[0,1,2,3]
[ 2 -1   0   0 ]    [ x1 ]    [ b1 ]
[-1  2  -1   0 ]    [ x2 ]    [ b2 ]
[ 0 -1   2  -1 ]    [ x3 ]  = [ b3 ]
[ 0  0  -1   2 ]    [ x4 ]    [ b4 ]
*/
void
test ()
{
// on va érsoudre A x = b;
  int n = 4;
//1- Allocation des ressources
  csi nzmax = 1;
  csi values = 1;
  csi triplet = 1;
  cs *T = cs_spalloc (n, n, nzmax, values, triplet);    // matrice creuse au format ↩
      écoordonnes (COO)
  double *x = cs_calloc (n, sizeof (double));
  double *b = cs_calloc (n, sizeof (double));
//2- Formation du èsystme
// diagonale
  for (int i = 0; i < n; ++i)
    {
      cs_entry (T, i, i, 2.0);
      x[i] = (double) i;        // x est la solution exacte
    }
//sous/sur diagonale
  for (int i = 0; i < n - 1; ++i)
    {
      cs_entry (T, i, i + 1, -1.0);
      cs_entry (T, i + 1, i, -1.0);
    }
// la matrice doit ^etre au format compress-sparse-column (CSC)
  cs *A = cs_compress (T);
  cs_spfree (T);
// second membre
  cs_gaxpy (A, x, b);           // b = A * x
//2- é Rsolution du èsystme
  csi renumerotation = 2;
  double tol = 1e-16;
  cs_lusol (renumerotation, A, b, tol);
//3-  Affichage et évrification de la solution
  for (int i = 0; i < n; ++i)
    printf ("%f \t %f \n", x[i], b[i]);
//4- é Libration des ressources
  cs_spfree (A);
  cs_free (x);
  cs_free (b);
}
/** ***************************************************
                PROGRAMME PRINCIPAL
*****************************************************/
int
main (int argc, char **argv)
{
  test ();
  return 0;
}
```

Listing 4 – csparse_illustration2.c

```c
#include <stdio.h>
#include <cs.h>
// Pour la compilation
// gcc -std=c99  -I../CSparse/Include  exemple2.c  -L../CSparse/Lib -lcsparse  -o exemple2 -lm

/*!
On va érsoudre le èsytme : A x  = b
On choisira b de sorte que la solution exacte soit x =[0,1,2,3]
```

```
[ 2 -1   0   0 ]    [ x1 ]    [ b1 ]
[-1  2  -1   0 ]    [ x2 ]    [ b2 ]
[ 0 -1   2  -1 ]    [ x3 ]  = [ b3 ]
[ 0  0  -1   2 ]    [ x4 ]    [ b4 ]
*/
/** ************************************************
                DECLARATIONS
**************************************************/
typedef struct
{
  csi n;
  cs *A;
  double *x;
  double *b;
  int is_init;
} Problem, *pProblem;

void Problem_init (pProblem pb);
void Problem_free (pProblem pb);
void Problem_build (pProblem pb);
void Problem_solve (pProblem pb);
void Problem_printSol (pProblem pb);

/*! Comment tester le code */
void
test ()
{
  int n = 4;
  Problem pb = { n, 0, 0, 0, 0 };
  Problem_init (&pb);
  Problem_build (&pb);
  Problem_solve (&pb);
  Problem_printSol (&pb);
  Problem_free (&pb);
}
/** ************************************************
                PROGRAMME PRINCIPAL
**************************************************/
int
main (int argc, char **argv)
{
  test ();
  return 0;
}
/** ************************************************
                DEFINITIONS
**************************************************/
void
Problem_init (pProblem pb)
{
  csi nzmax = 1;
  csi values = 1;
  csi triplet = 1;
  csi n = pb->n;
  pb->A = cs_spalloc (n, n, nzmax, values, triplet);
  pb->x = cs_calloc (n, sizeof (double));
  pb->b = cs_calloc (n, sizeof (double));
  pb->is_init = 1;
}
/** ------------------------------------------------*/
void
Problem_free (pProblem pb)
{
  if (pb->is_init)
    {
      cs_spfree (pb->A);
      cs_free (pb->x);
      cs_free (pb->b);
      pb->is_init = 0;
    }
}
/** ------------------------------------------------*/
void
```

```
Problem_build (pProblem pb)
{
  csi n = pb->n;
  cs *T = cs_spalloc (n, n, 1, 1, 1);
// diagonale
  for (int i = 0; i < n; ++i)
    {
      cs_entry (T, i, i, 2.0);
      pb->x[i] = (double) i;    // x stocke temporairement la solution exacte
    }
//sous/sur diagonale
  for (int i = 0; i < n - 1; ++i)
    {
      cs_entry (T, i, i + 1, -1.0);
      cs_entry (T, i + 1, i, -1.0);
    }

  cs_spfree (pb->A);
  pb->A = cs_compress (T);
  cs_spfree (T);
  cs_gaxpy (pb->A, pb->x, pb->b);       // b = A * x

  // x est ermis à  zero
  for (int i = 0; i < n; ++i)
    pb->x[i] = 0.0;
}
/** ------------------------------------------------------*/
void
Problem_solve (pProblem pb)
{
  csi n = pb->n;
  csi renumerotation = 2;
  double tol = 1e-16;
// on veut que  p->x contienne la solution du èproblme
  for (int i = 0; i < n; ++i)
    pb->x[i] = pb->b[i];

  cs_lusol (renumerotation, pb->A, pb->x, tol);
}
/** ------------------------------------------------------*/
void
Problem_printSol (pProblem pb)
{
  csi n = pb->n;
  for (int i = 0; i < n; ++i)
    printf (" x[%d] = %f \n", i, pb->x[i]);
}
/** ------------------------------------------------------*/
```

Remark 0.0.1 (Résolution d'un système avec plusieurs second membres)**.**

On peut être amené à résoudre un système linéaire avec plusieurs second membres, ou être confronté à un problème dont seul le second membre change au cours des itérations. Ce dernier survient notamment lors de la discrétisation des équations au dérivées partielles non stationnaires dont les coefficients sont indépendants du paramètre temps. Dans ces cas on peut factoriser une fois pour toute la matrice et n'utiliser que la factorisation pour déterminer les solutions pour divers second membres. On procéderait alors comme dans le Listing5.

Listing 5 – csparse_illustration3.c

```
int main(int argc, char** argv)
{
/* Dans cet exemple, on stocke la factorisation LU */
    double *z *b;
    cs *A;
```

```
    // On suppose que la matrice A et le vecteur b sont construits.
    // On rappelle que b est le second membre et est ecrase par la solution
    css *S ;  /* stocke la factorisation symbolique */
    csn *N ; /* stocke la factorisation numerique */
    csi n, ok ;
    if (!CS_CSC (A) || !b) return (0) ;      /* verification des donnees*/
    n = A->n ;
    S = cs_sqr (renumerotation, A, 0) ;      /* renumerotation et analyse symbolique */
    N = cs_lu (A, S, tol) ;                  /* factorisation numerique  LU */
    z = cs_malloc (n, sizeof (double)) ;     /* get workspace */
    ok = S && N && z;
    if (ok)
    {
        cs_ipvec (N->pinv, b, z, n) ;        /* z = b(p) */
        cs_lsolve (N->L, z) ;                /* z = L\z */
        cs_usolve (N->U, z) ;                /* z = U\z */
        cs_ipvec (S->q, z, b, n) ;           /* b(q) = z */
    }
    // Si on modifie ulterieurement b,
    // on peut encore determiner la solution sans
    // factoriser une nouvelle fois la matrice :
        cs_ipvec (N->pinv, b, z, n) ;
        cs_lsolve (N->L, z) ;
        cs_usolve (N->U, z) ;
        cs_ipvec (S->q, z, b, n) ;

    cs_free (x) ;
    cs_sfree (S) ;
    cs_nfree (N) ;
    cs_spfree (A);
    cs_free (b);
return 0;
}
```

Remark 0.0.2.

Ainsi, si l'on souhaite résoudre une équation de la chaleur par un schéma aux différences finies en espace et un schéma implicite en temps, on peut directement stocker dans la structure `Problem` correspondante, les factorisations symbolique et numérique de la matrice principale (celle du premier membre du problème discrétisé en temps et en espace).

Pour conclure, montrons une utilisation de la librairie dans la résolution d'une équation aux dérivées partielles en dimension 2.

Listing 6 – Illustration de la résolution d'un problème de Laplace en deux dimensions

```
#include <stdlib.h>
#include <stdio.h>
#include <assert.h>
#include <cs.h>
#ifdef DEBUG
#define CHECK(a)
printf ("file %s :  line %d  values : %f \n", __FILE__, __LINE__, (float) a)
#else
#define CHECK(a)
#endif
/*****************************************************
  Convertisseur:
  Permet une numerotation lexicographique des sommets
  de la grille.
  Il retoure le numero global du point se
  trouvant à la position (i,j)  d'une grille de taille  n x m
  --------
  Remarque: une approche plus generique sera faite en TP1
  --------
```

```c
 *****************************************************/
static int
C2I (int i, int j, int n, int m)
{
  return (j * n + i);
}

/*******************************************************
Type pour identifier la grille  cartesienne
*******************************************************/
typedef struct
{
  // En entree
  double a0, b0, a1, b1;         // le domaine [a0,a1]x[b0,b1]
  int nx;                        //nombre de points dans [a0,b0] a0 et a1 inclus
  int ny;                        //nombre de points dans [a0,b0] b0 et b1 inclus
  int is_param;                  // 1
  //En sortie
  double hx, hy;                 // pas dans les direction respective x, y
  double *x;                     // subdivision de [a0,a1], a0, a1 inclus
  double *y;                     // subdivision de [b0,b1], b0, b1 inclus
} Mesh, *pMesh;
//exemple d'initialisation Mesh m={0,0,1,1,20,20,1,0,0,0,0}
/*---------------------------------------------------*/
void
mesh_alloc (pMesh m)
{
  //if(!m->is_param)  return;
  m->x = (double *) cs_malloc (m->nx, sizeof (double));
  m->hx = (m->a1 - m->a0) / (double) (m->nx - 1);
  for (int i = 0; i < m->nx; ++i)
    m->x[i] = m->a0 + i * m->hx;

  m->y = (double *) cs_malloc (m->ny, sizeof (double));
  m->hy = (m->b1 - m->b0) / (double) (m->ny - 1);
  for (int i = 0; i < m->ny; ++i)
    m->y[i] = m->b0 + i * m->hy;
}

/*---------------------------------------------------*/
void
mesh_base_init (pMesh m,
                double a0, double b0, double a1, double b1, int nx, int ny)
{
  m->a0 = a0;
  m->b0 = b0;
  m->a1 = a1;
  m->b1 = b1;
  m->is_param = 1;
  m->nx = nx;
  m->ny = ny;
  mesh_alloc (m);
}

/*---------------------------------------------------*/
void
mesh_free (pMesh m)
{
  if (m->x)
    cs_free (m->x);
  if (m->y)
    cs_free (m->y);
}

/*---------------------------------------------------*/
void
mesh_print (pMesh m, double *b)
{
  int nx = m->nx;
  int ny = m->ny;
  int i, j;
  for (i = 0; i < nx - 1; ++i)
    {
      for (j = 0; j < ny - 1; ++j)
```

```
        {
          if (b != NULL)
            printf ("%f \t %f\t %f\n", m->x[i], m->y[j + 1],
                    b[C2I (i, j + 1, nx, ny)]);
          else
            printf ("%f \t %f\t %f\n", m->x[i], m->y[j + 1], 0);
        }
      printf ("\n");
    }
}

/*********************************************************************
   Type pour identifier le second membre et la condition de dirichlet
*********************************************************************/
typedef double (*pF) (double, double);
/*********************************************************************
   Type pour identifier le èproblme a resoudre
*********************************************************************/
typedef struct
{
  Mesh m;
  cs *A;
  double *b;
  cs *T;
  pF f;
  pF g;
} Problem, *pProblem;
/*-----------------------------------------------------*/
void
problem_alloc (pProblem p)
{
  CHECK (p->m.nx);
  CHECK (p->m.ny);
  csi n = p->m.nx * p->m.ny;
  csi nzmax = (p->m.nx - 1) * (p->m.ny - 1) * (1 + 4)
    + (p->m.nx - 1) * 2 + (p->m.ny - 1) * 3 + 4;
  csi values = 1;
  csi triplet = 1;
  p->T = (cs *) cs_spalloc (n, n, nzmax, values, triplet);
  p->b = (double *) cs_calloc (n, sizeof (double));
}

/*-----------------------------------------------------*/
void
problem_free (pProblem p)
{
  cs_spfree (p->A);
  mesh_free (&p->m);
  cs_free (p->b);
}

/*-----------------------------------------------------*/
//On affiche sur le terminal.
//Il faut rediriger la sortie vers un fichier text
//pour un traitement pas gnuplot
void
problem_print_sol (pProblem p)
{
  mesh_print (&p->m, p->b);
  //mesh_print(&p->m, NULL); // pour tester
}

/*-----------------------------------------------------*/
// Attention les conditions aux limites sont mise
// sous forme faible on Obtient pas la forme habituelle
// du éschma à  5 points
void
problem_build_laplace (pProblem p)
{
  double hx = p->m.hx, hy = p->m.hy;
  double ihx = 1. / (hx * hx);
  double ihy = 1. / (hy * hy);
  int nx = p->m.nx;
  int ny = p->m.ny;
```

```
  CHECK (nx);
  CHECK (ny);
  CHECK (ihx);
  CHECK (ihy);
  int i, j, I, Ig, Id, Ih, Ib;
  // parcours des points internes
  for (i = 1; i < nx - 1; ++i)
    {
      for (j = 1; j < ny - 1; ++j)
        {

          I = C2I (i, j, nx, ny);
          Ig = C2I (i - 1, j, nx, ny);
          Id = C2I (i + 1, j, nx, ny);
          Ih = C2I (i, j + 1, nx, ny);
          Ib = C2I (i, j - 1, nx, ny);

          assert (cs_entry (p->T, I, I, 2. * ihx + 2. * ihy));
          assert (cs_entry (p->T, I, Ig, -ihx));
          assert (cs_entry (p->T, I, Id, -ihx));
          assert (cs_entry (p->T, I, Ih, -ihy));
          assert (cs_entry (p->T, I, Ib, -ihy));
          p->b[I] = p->f (p->m.x[i], p->m.y[j]);
        }
    }
  // traitement des points frontieres
  for (i = 1; i < nx - 1; ++i)
    {
      I = C2I (i, 0, nx, ny);
      CHECK (I);
      assert (cs_entry (p->T, I, I, 1));
      p->b[I] = p->g (p->m.x[i], p->m.y[0]);
      I = C2I (i, ny - 1, nx, ny);
      assert (cs_entry (p->T, I, I, 1));
      p->b[I] = p->g (p->m.x[i], p->m.y[ny - 1]);
    }

  for (j = 1; j < ny - 1; ++j)
    {
      I = C2I (0, j, nx, ny);
      assert (cs_entry (p->T, I, I, 1));
      p->b[I] = p->g (p->m.x[0], p->m.y[j]);
      I = C2I (nx - 1, j, nx, ny);
      assert (cs_entry (p->T, I, I, 1));
      p->b[I] = p->g (p->m.x[nx - 1], p->m.y[j]);
    }

  I = C2I (0, 0, nx, ny);
  assert (cs_entry (p->T, I, I, 1));
  p->b[I] = p->g (p->m.x[0], p->m.y[0]);

  I = C2I (nx - 1, 0, nx, ny);
  assert (cs_entry (p->T, I, I, 1));
  p->b[I] = p->g (p->m.x[nx - 1], p->m.y[0]);

  I = C2I (nx - 1, ny - 1, nx, ny);
  assert (cs_entry (p->T, I, I, 1));
  p->b[I] = p->g (p->m.x[nx - 1], p->m.y[ny - 1]);

  I = C2I (0, ny - 1, nx, ny);
  assert (cs_entry (p->T, I, I, 1));
  p->b[I] = p->g (p->m.x[0], p->m.y[ny - 1]);

  p->A = cs_compress (p->T);
  assert (p->A);
  CHECK (p->A->nz);
  cs_spfree (p->T);
}

/*-----------------------------------------------------*/
void
problem_solve (pProblem p)
{
  /* p = amd(A+A') if symmetric is true, or amd(A'A) otherwise */
```

```c
  /* order 0:natural, 1:Chol, 2:LU, 3:QR */
  int order = 2;                //
  double tol = 1e-36;
  assert (cs_lusol (order, p->A, p->b, tol));
}

/*****************************************************
 *    Ici on teste le maillage
 * ***************************************************/
void
test_mesh ()
{
  Mesh m;
  mesh_base_init (&m, 0.0, 0.0, 1.0, 1.0, 30, 30);
  CHECK (m.nx);
  CHECK (m.ny);
  mesh_free (&m);
}

/*****************************************************
 *  Donnees pour le probleme de Diriclet
 * ***************************************************/
double
f (double x, double y)
{
  return -4;
}

double
g (double x, double y)
{
  return (x * x + y * y);
}

/*****************************************************
 *  Le programme principal
 * ***************************************************/
int
main (int argc, char **argv)
{
  Problem pb;
  mesh_base_init (&pb.m, 0, 0, 1, 1, 25, 25);
  problem_alloc (&pb);
  pb.f = f;
  pb.g = g;
  problem_build_laplace (&pb);
  problem_solve (&pb);
  problem_print_sol (&pb);
  problem_free (&pb);
  return 0;
}
```