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

Fiche de TP 3 : Méthodes itératives pour matrices creuses

On s'intéresse à la résolution par méthodes itératives, 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 ou Csparse pour le stockage des matrices et vecteurs.*
- *Gnuplot pour la représentation graphique des matrices.*

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

Thème - 1 *Stockage optimal des matrices pour méthodes itératives*

Note 1.

Les matrices, dans les méthodes itératives, n'interviennent que dans les produits matrices-vecteurs. Un gain considérable en espace mémoire est donc possible si on ne stocke que les éléments non nuls de la matrice. Il faut alors décrire comment réaliser le produit matrice-vecteur avec le format de stockage adopté.

Q-1 : Stockage CSR (en anglais Compressed Sparse Row) ou stockage Morse.

Dans ce format on utilise trois tableaux (**`IA, JA, VA`**) pour stocker la matrice A. On ne stocke que les éléments non nuls de la matrice dans un vecteur **`VA`**, en parcourant la matrice ligne par ligne de la première à la dernière. Pour le k-ème élément non nul stocké en **`VA(k)`**, son indice de colonne est donné par **`JA(k)`**. Le vecteur **`IA`** est de taille **`n+1`** (où n est le nombre de lignes de la matrice) et est défini de la manière suivante : **`IA(i)`** est la position dans **`VA`** du premier élément non nul de la ligne **`i`**. Une autre interprétation qui justifie la taille **`n+1`** de **`IA`** est la suivante : On a toujours **`IA(1)= 0`** et **`IA(i+1) - IA(i)`** est le nombre d'éléments non nuls de la ligne **`i`** ; Ainsi **`IA(n) = nnz`**, où **`nnz`** est le nombre d'éléments non nuls de la matrice **`A`**, c'est à dire la taille de **`VA`**.

Représenter en langage **`C`**, une matrice creuse au format CSR en compléter la structure de données suivante. On créera les fichiers **`MatriceCSR.h, MatriceCSR.c`** pour l'occasion.

Listing 1 – Structure pour stockage CSR d'une matrice

```
typedef
struct MatriceCSR{
 int n;   // taille de la matrice
 int nnz; // nombre d'elements non nuls (juste pour acces rapide car nnz = IA[n])
 int*    IA; // pointeur des debuts de lignes dans VA
 int*    JA; // indices de colonne
 double* VA; // valeurs non nulles
}MatriceCSR;

//Construction de la matrice creuse a  partir d'une matrice pleine
//On ne retient que les entrees A(i,j)  telles que  |A(i,j)| > epsi
MatriceCSR* MatriceCSR_cree(const gsl_matrix* A, double epsi)

//Liberation des ressources
MatriceCSR_libere(void * sm);

//Produit matrice vecteur :  y = A * x
MatriceCSR_matvect(const MatriceCSR* A, const gsl_vector* x, gsl_vector* y);
```

Thème - 2 *Méthode du gradient à pas fixe (ou de Richardson)*

Exercice-1 **:** **Brèves notions théoriques**

On considère la fonction f définie de $\mathbb{R}^n$ dans $\mathbb{R}$ par $f(x) = \frac{1}{2}(Ax, x) - (b, x)$.

Q-1 **:** Montrer que x_0 est solution de $Ax = b$ si et seulement si x_0 minimise la fonction f.

Q-2 **:** On considère la méthode itérative : $\|x_0$ étant un vecteur initial donné, $x_{k+1} = x_k - \alpha\nabla f(x_k),\ k = 1, \ldots$.

Q-2-1 **:** Donner la matrice d'itération B de cette méthode et conclure que cette méthode converge si et seulement si $0 < \alpha < \frac{2}{\lambda_n}$ où $0 < \lambda_1 \leq \ldots \leq \lambda_n$ sont les valeurs propres de la matrice symétrique définie positive A.

Q-2-2 **:** Montrer que le meilleur choix de α est : $\alpha_{opt} = \frac{2}{\lambda_n+\lambda_1}$ et qu'alors $\varrho(B) = \frac{\lambda_n-\lambda_1}{\lambda_n+\lambda_1} \equiv \frac{cond(A)-1}{cond(A)+1}$.

Exercice-2 **:** **Algorithme et Implémentation**

Q-1 **:** Déduire l'algorithme, donné ci-dessous, du gradient à pas constant, faisant intervenir le résidu dans les itérations. (*On rappelle que le résidu à l'itération k est défini par $r_k = b - Ax_k$*).

Algorithme du gradient a pas fixe

ALGORITHME DU GRADIENT A PAS FIXE	COMMENTAIRES
Donnees: A,b, x0, iterMax, eps	
Resultat x approximation de A^ -1 b	
Initialisation:	
Choisir : x = x0	
Calculer : r = b - A * x	
iter = 0	
Iterations :	
Tant que (|| r || > eps * ||b||) et (iter < iterMax)	
x = x + alpha * r	
r = b - A * x	r = r - alpha* (A*r)
iter = iter + 1	
Fin Tant que	

(On peut stocker les normes des residus (|| r ||) dans un vecteur de taille iterMax pour un post-traitement)

Q-2 **:** Programmer cet algorithme à travers une fonction de prototype :

```
int GradientPasFixe(const MatriceCSR * A, gsl_vector* x, const gsl_vector* b,
                    double alpha, double tol, int iterMax,  gsl_vector* Resi)
```

- **A** est la matrice du système et **b** le vecteur second membre,
- **x** est le vecteur initial en entrée, et contiendra la solution approchée en sortie,
- **alpha** désigne le paramètre α (le pas fixe),
- **tol** est la valeur ε du test d'arrêt et **iterMax** le nombre maximal d'itérations,
- **Resi** est un vecteur (de taille iterMax) stockant la valeur de la norme du résidu à chaque itération : Resi$[k] = \|r_k\|, k = 1, \ldots,$.

Exercice-3 **:** **Validation**

On considère la matrice et second membre du problème de Laplace 2D générée, soit par une de vos fonctions (TP1) soit par la fonction du Listing3 (**A = matrice_Laplace2D(10,10), b = smb_Laplace2D(10,10)**).
On fixe **iterMax = 10000, tol = 1E-6**. Faire varier α entre **1E-3** et **3 E-3** par pas de **1E-4**. Représenter (ou afficher sur le terminal) le nombre d'itérations en fonction du paramètre α. Déterminer numériquement la valeur α conduisant à un nombre minimal d'itérations. Comparer avec la valeur donnée par la théorie.

Thème - 3 *Méthode du gradient à pas variable*

Exercice-1 **:** **Brèves notions théoriques**

Q-1 **:** On suppose construite une suite $(p_0, p_1, \dots p_k, \dots)$ de vecteurs linéairement indépendants. Et on considère la méthode itérative suivante :

x_0 vecteur initial donné,
$x_{k+1} = x_k + \alpha_k p_k$.
Où α_k est choisi tel qu'il réalise le minimum de $f(x_k + \alpha p_k)$.

Q-1-1 **:** Montrer que $\alpha_k = \frac{(r_k, p_k)}{(Ap_k, p_k)}$, où $r_k = b - Ax_k$, et conclure que $(r_{k+1}, p_k) = 0, \forall k \geq 0$.

Q-1-2 **:** On pose $E(x_k) = (A(x_k - x), (x_k - x))$ où x est la solution de $Ax = b$.
Montrer que pour la valeur de α_k ci-dessus, on a $E(x_{k+1}) = E(x_k) - \frac{(r_k, p_k)^2}{(Ap_k, p_k)}$.

Q-1-3 **:** En remarquant que $E(x_k) = (A^{-1}r_k, r_k)$, montrer que $E(x_{k+1}) = E(x_k)\left(1 - \frac{(r_k, p_k)^2}{(A^{-1}r_k, r_k)(Ap_k, p_k)}\right)$.

Q-1-4 **:** Déduire de la question précédente que $E(x_{k+1}) \leq E(x_k)\left[1 - \frac{1}{cond(A)}\left(\frac{r_k}{\|r_k\|}, \frac{p_k}{\|p_k\|}\right)^2\right]$.
Où $cond(A) = \frac{\lambda_n}{\lambda_1}$ désigne le conditionnement de la matrice A.
on utilisera le fait que $(Ay, y) \leq \lambda_n(y, y), \quad (A^{-1}y, y) \leq \frac{1}{\lambda_1}(y, y) \forall y$.
Conclure qu'une condition suffisante de convergence est de choisir les p_k tels que
$\forall k \geq 0 \quad \left(\frac{r_k}{\|r_k\|}, \frac{p_k}{\|p_k\|}\right) \geq \mu > 0$, où μ est une constante indépendante de k.

Q-1-5 **:** Déduire de la question précédente que $p_k = r_k, \forall k \geq 0$ est un choix possible assurant la convergence.

Q-2 **:** **On prend dans cette question** $p_k = r_k \ \ \forall k \geq 0$.

Q-2-1 **:** Écrire l'algorithme ainsi obtenu.

Q-2-2 **:** Que devient $E(x_{k+1})$ de la question 1-c ci-dessus ?

Q-2-3 **:** *On admet l'inégalité de Kantorovich suivante :* $\frac{(Ay, y)(A^{-1}y, y)}{(y, y)^2} \leq \frac{(\lambda_n + \lambda_1)^2}{4\lambda_n \lambda_1} \ \forall y \neq 0$..
Montrer qu'on a alors $E(x_{k+1}) = E(x_k)\left(\frac{cond(A)-1}{cond(A)+1}\right)^2$.

Q-2-4 **:** Conclure que $\|x_k - x\| \leq \sqrt{\frac{E(x_0)}{\lambda_1}}\left(\frac{cond(A)-1}{cond(A)+1}\right)^k$.

Exercice-2 : Algorithme et implémentation

Q-1 : Montrer que l'algorithme du gradient à pas variable peut s'écrire comme ci-dessous (compartiment de droite).

Algorithme du gradient a pas variable

```
     ALGORITHME DU GRADIENT A PAS VARIABLE            |   NOTATIONS ET COMMENTAIRES
------------------------------------------------------|-------------------------------------------
Donnees: A,b, x0, iterMax, eps                        |- (a,b) designe le produit scalaire de a et b
Resultat x approximation de A^-1 b                    |- On peut optimiser les calculs
------------------------------------------------------|-------------------------------------------
 Initialisation :
    x  = x0                                           | x = x0
    r  = b - A * x                                    | r = b - A * x
  iter = 0                                            | gamma = (r,r)
                                                      | gamm0 = eps * eps * (b,b)
                                                      | iter = 0
 Iterations :
Tant que (|| r || > eps * ||b||) et (iter < iterMax)  | Tant que (gamma > gamm0) et (iter < iterMax)
    alpha = (r,r)/(A * r, r)                          |   y = A * r
    x = x + alpha * r                                 |   alpha = gamma/(y,r)
    r = b - A * x                                     |   x = x + alpha * r
    iter = iter + 1                                   |   r = r - alpha * y
                                                      |   gamma = (r,r)
                                                      |   iter = iter + 1
Fin Tant que                                          |Fin Tant que
-------------------------------------------------------------------------------------
( On peut stocker les érsidus(|| r ||) dans un vecteur de taille iterMax  pour un post-traitement)
```

Q-2 : Programmer cet algorithme à travers une fonction de prototype :

```
int GradientPasVariable(const MatriceCSR * A, gsl_vector* x, const gsl_vector* b,
                                        double tol, int iterMax, gsl_vector* Resi)
```

(*Les arguments sont définis comme à la méthode du gradient à pas fixe.*)

Exercice-3 : Validation

Exécuter l'algorithme avec les données de l'exercice 3 du Thème précédent. Afficher les valeurs du pas variable α_k, et comparer les avec la valeur optimale du pas dans la méthode du gradient à pas fixe pour le même problème. Que constatez-vous ?

Thème - 4 *Méthode du gradient conjugué*

Exercice-1 **:** **Brèves notions théoriques**

Une amélioration de la méthode du gradient à pas variable consiste à choisir les directions p_k telles que x_{k+1} réalise le minimum de f sur $x_0+[p_0, p_1, \dots p_k]$. La particularité de la méthode réside dans le fait que ce problème de minimisation globale, doit se réduire au problème de minimisation locale : $\min_{x=x_0+\bar{x}+y, y\in[p_k]} f(x)$, dans lequel $\bar{x} \in [p_0, p_1, \dots p_{k-1}]$ est connu et issu d'une précédente minimisation. Les deux premières questions ci-dessous expliquent pourquoi on doit choisir les p_k A-conjugués.

Q-1 **:** Monter que $f(x_0 + \bar{x} + \alpha p_k) = f(x_0 + \bar{x}) + \alpha(p_k, A\bar{x}) + \frac{\alpha^2}{2}(p_k, Ap_k) - \alpha(p_k, r_0)$.

Q-2 **:** Conclure qu'un moyen de découpler le problème de minimisation globale en une succession de problèmes de minimisation locale consiste à prendre $(p_k, A\bar{x}) = 0$, ce qui équivaut à $(p_k, Ap_i) = 0,\ \forall 0 \leq i \leq k-1$. **On dit dans ce cas que les vecteurs** $p_i, i = 1 \dots k$ **sont A-conjugués**.

La méthode du gradient conjugué consiste alors en deux points :

- choisir les directions p_k A-conjuguées ; c'est-à-dire $(Ap_k, p_j) = 0\ \forall 0 \leq j \leq k-1$,
- initialiser $p_0 = r_0$, et prendre p_{k+1} dans le plan contenant r_{k+1} et p_k c'est-à-dire $p_{k+1} = r_{k+1} + \beta_{k+1}p_k$.

Q-3 **:** Montrer que les vecteurs p_{k+1} et p_k sont A-conjugués si et seulement si $\beta_{k+1} = -\frac{(r_{k+1}, Ap_k)}{(p_k, Ap_k)}$.

Q-4 **:** En remarquant que $(r_k, p_{k-1}) = 0\ \forall k$, montrer que $(r_k, p_k) = (r_k, r_k)\ \forall k$.
Simplifier alors l'expression de α_k.

Q-5 **:** En écrivant $r_k = p_k - \beta_k p_{k-1}$, montrer que $(r_{k+1}, r_k) = 0$. (*On utilisera l'expression de* β_k).

Q-6 **:** En écrivant $Ap_{k-1} = \frac{1}{\alpha_{k-1}}(r_{k-1} - r_k)$, montrer que

$$(Ap_{k-1}, r_k) = -\frac{1}{\alpha_{k-1}}(r_k, r_k) \quad \text{et} \quad (Ap_{k-1}, p_{k-1}) = \frac{1}{\alpha_{k-1}}(r_{k-1}, r_{k-1}).$$

Simplifier alors l'expression de β_{k+1}.

Q-7 **:** Montrer que $(r_k, r_j) = 0\ \forall\ 0 \leq j \leq k-1$.

En déduire qu'en arithmétique exacte, l'algorithme du gradient conjugué converge en au plus n itérations, où n est la taille du système.

(*On remarquera que si* $r_k \neq 0, \forall\ 0 \leq k \leq n-1$*, alors* $[r_0, \dots, r_{n-1}]$ *est une base de* $\mathbb{R}^n$).

Exercice-2 : Algorithme et implémentation

Q-1 : En déduire l'algorithme du gradient conjugué donné ci-dessous

Algorithme du gradient conjugué

```
Donnees: A,b, x0, iterMax, eps                    |- (a,b) designe le produit scalaire de a et b
Resultat x approximation de A^-1 b                |- On peut optimiser les calculs
```

Initialisation :

```
                                                  | k = 0
                                                  | x = x0
   r0  = b - A * x0                               | r = b - A * x
   d0  = r0                                       | p = r
                                                  | gamma = (r,r)
                                                  | gamm0 = eps * eps * (b,b)
```

Iterations :

```
Pour k = 0,...                                    | Tant que (gamma > gamm0) et ( k < iterMax)
                                                  |  y = A * p
    alpha_k = (r_k,r_k)/(A * p_k, p_k)            |  alpha = gamma/(y,p)
    x_{k+1} = x_k + alpha_k * p_k                 |  x = x + alpha * p
    r_{k+1} = b - A * x_{k+1}                     |  r = r - alpha * y
   beta_{k+1} = (r_{k+1},r_{k+1})/(r_k,r_k)       |  beta = (r,r)/gamma
                                                  |  gamma = (r,r)
   p_{k+1} = r_{k+1} + beta_{k+1} * p_k           |  p = r + beta * p
                                                  |  k  = k + 1
   STOP Si ||r_{k+1}|| < eps * ||b|| OU k = iterMax|
                                                  |
Fin Pour k                                        | Fin Tant que
```

(**On peut stocker les normes des residus (|| r ||) dans un vecteur pour un post-traitement**)

Q-2 : Programmer cet algorithme à travers une fonction de prototype :

```
int GradientConjugue(const MatriceCSR * A, gsl_vector* x, const gsl_vector* b,
                                    double tol, int iterMax, gsl_vector* Resi)
```

(*Les arguments sont définis comme à la méthode du gradient à pas variable.*)

Thème - 5 *Comparaison numérique des méthodes du gradient*

Exercice-1 : Comparaison numérique sur un problème du Laplacien en dimension 2

On considère A et b la matrice et le second membre obtenus par discrétisation par différences finies du laplacien sur $]0,1[\times]0,1[$ avec pour second membre $f = 1$ et des conditions aux limites de Dirichlet homogènes. La grille utilisée est définie par $h_x = \frac{1}{n+1}, h_y = \frac{1}{m+1}$.

Q-1 : Exécutez le script suivant, en vous assurant que les fonctions appelées sont bien accessibles.

Listing 2 – Script de test des méthodes du gradient

```
 /* @c J.-B. A. K. Cours M325 Calcul Scientifique II 2015 */

typedef enum{GRAD_FIX = 0, GRAD_VAR, GRAD_CONJ} TypeMethode;
void test_methode_gradient(TypeMethode method, const MatriceCSR* A, gsl_vector* b,
                           double alpha, double tol, int iterMax, const char* fichier)
{
  gsl_vector* Resi = gsl_vector_alloc(iterMax+1);
  gsl_vector* x = gsl_vector_calloc(b->size); // x = 0
  int nombre_iterations;
  int k ;
  switch(method)
  {
    default: break;
    case GRAD_FIX:
```

```
    {
      nombre_iterations = GradientPasFixe(A, x, b, alpha, tol, iterMax, Resi);
    }
    break;
    case GRAD_VAR:
    {
     nombre_iterations = GradientPasVariable(A, x, b, tol, iterMax, Resi);
    }
    break;
    case GRAD_CONJ:
    {
      nombre_iterations = GradientConjugue(A, x, b, tol, iterMax, Resi);
    }
    break;
  }
  FILE* os = fopen(fichier,"w");
  for (k = 0; k < nombre_iterations; k++)
  {
    fprintf(os,"%f  %g  \n", (double)k, log(gsl_vector_get(Resi,k)));
  }
  fclose(os);
  gsl_vector_free(Resi);
  gsl_vector_free(x);
};

void
test_comparaison ()
{
//éCration du èproblme du Laplacien
  int n = 10;
  gsl_matrix *gslA = matrice_Laplace2D (n, n);
  gsl_vector *b = smb_Laplace2D (n, n);

//Recherche de alpha optimal pour gradient a pas fixe
  double lambda[2];
  valeurs_propres_extremes (gslA, lambda);
  double alpha = 2.0 / (lambda[0] + lambda[1]);

// Stockage au format CSR
  MatriceCSR *A = MatriceCSR_cree (gslA, 1e-32);
  MatriceCSR_affiche (A);

// Test des methodes
// On stocke dans un fichier pour gnuplot,
// pour l'iteration k, on stocke sur la ligne k
// k   log(|| rk ||)
// de sorte a pouvoir representer la courbe k -> log(||rk||)

  double tol = 1e-9;
  int iterMax = 100000;
  test_methode_gradient (GRAD_FIX, A, b, alpha, tol, iterMax, "grad_fixe.txt");
  test_methode_gradient (GRAD_VAR, A, b, alpha, tol, iterMax, "grad_var.txt");
  test_methode_gradient (GRAD_CONJ, A, b, alpha, tol, iterMax, "grad_conj.txt");

//Liberation des ressources
  gsl_vector_free (b);
  gsl_matrix_free (gslA);
  MatriceCSR_libere (A);
}


int
main (int argc, char **argv)
{
 // test_stockage_csr();
 //  test_alpha_optimal ();
   test_comparaison();
  return 0;
}
```

Q-2 : Interpréter les résultats observés.

Listing 3 – Matrice et second membre du Laplacien 2D

```
 /* @c J.-B. A. K. Cours M325 Calcul Scientifique II 2015 */
 /* discretisation de
   -u''(x,y) = 1          sur ]0,1[ x ]0,1[
    u(0,y) = u(1,y) = 0   sur [0,1]
    u(x,0) = u(x,1) = 0   sur [0,1]
    sur une subdivision de [0,1] x [0,1] generant
    n points internes suivant l'axe des x et
    m points internes suivant l'axe des y
 */
int ** creer_table_numerotation (int nx, int ny)
void liberer_table_numerotation(int **C2I, int nx, int ny);
/*********************  MATRICE  **********************************/
gsl_matrix *
matrice_Laplace2D (int n, int m)
{
  int i, j, I, Ig, Id, Ih, Ib;

  double hx = 1. / (double) (n + 1);
  double hy = 1. / (double) (m + 1);
  double ihx = 1. / (hx * hx);
  double ihy = 1. / (hy * hy);
  int nx = n + 2;
  int ny = m + 2;
  int ns = n * m;

  gsl_matrix *A = gsl_matrix_alloc (ns, ns);
  int **C2I = creer_table_numerotation (nx, ny);
  for (i = 1; i < nx - 1; i++)
    {
      for (j = 1; j < ny - 1; j++)
        {
          I = C2I[i][j];
          Ig = C2I[i - 1][j];
          Id = C2I[i + 1][j];
          Ib = C2I[i][j - 1];
          Ih = C2I[i][j + 1];
          gsl_matrix_set (A, I, I, 2. * ihx + 2. * ihy);
          if (Ig >= 0)
            gsl_matrix_set (A, I, Ig, -ihx);
          if (Id >= 0)
            gsl_matrix_set (A, I, Id, -ihx);
          if (Ih >= 0)
            gsl_matrix_set (A, I, Ih, -ihy);
          if (Ib >= 0)
            gsl_matrix_set (A, I, Ib, -ihy);
        }
    }
  liberer_table_numerotation (C2I, nx, ny);
  return A;
}
/********************* SECOND MEMBRE **********************************/
gsl_vector *
smb_Laplace2D (int n, int m)
{
  int i, j, I, Ig, Id, Ih, Ib;
  double hx = 1. / (double) (n + 1);
  double hy = 1. / (double) (m + 1);
  double ihx = 1. / (hx * hx);
  double ihy = 1. / (hy * hy);
  int nx = n + 2;
  int ny = m + 2;
  int ns = n * m;

  gsl_vector *b = gsl_vector_calloc (ns);
  int **C2I = creer_table_numerotation (nx, ny);

  for (i = 1; i < nx - 1; i++)
    {
      for (j = 1; j < ny - 1; j++)
        {
          I = C2I[i][j];
          gsl_vector_set (b, I, 1.0);
        }
    }
  liberer_table_numerotation (C2I, nx, ny);
  return b;
}
```

```
/*****************************************************
*             Fonctions internes
/*****************************************************/
int **
creer_table_numerotation (int nx, int ny)
{
  int i, j;
  int **C2I = (int **) malloc (nx * sizeof (int *));
  for (i = 0; i < nx; ++i)
    C2I[i] = (int *) calloc (ny, sizeof (int));
  int num = 0;
  for (j = 0; j < ny; j++)
    for (i = 0; i < nx; i++)
      {
        if ((i == 0) || (i == (nx - 1)) || (j == 0) || (j == (ny - 1)))
            C2I[i][j] = -1;
        else
            C2I[i][j] = num++;
      }
  return C2I;
}
/*-----------------------------------------------*/
void
liberer_table_numerotation (int **C2I, int nx, int ny)
{
  int i;
  if (C2I)
    {
      for (i = 0; i < nx; ++i)
        {
          if (C2I[i])
            free (C2I[i]);
          C2I[i] = NULL;
        }
      free (C2I);
      C2I = NULL;
    }
}
```

Listing 4 – Valeurs propres extrêmes

```
/* @c J.-B. A. K. Cours M325 Calcul Scientifique II 2015 */
/*
  Fonction retournant les valeurs propres extremes d'une matrice symetrique
  Necessaire pour determiner le pas optimal dans la methode du gradient a pas fixe
  alpha_optimal = 2.0/(lambda[0] + lambda[1]);
*/
void
valeurs_propres_extremes (const gsl_matrix * A, double lambda[2])
{
  int n = A->size1;
  int m = A->size2;
  gsl_matrix *Aw = gsl_matrix_alloc (n, m);

  gsl_matrix_memcpy (Aw, A);
  assert (n == m);

  gsl_vector *eval = gsl_vector_alloc (n);

  gsl_eigen_symm_workspace *w = gsl_eigen_symm_alloc (n);

  gsl_eigen_symm (Aw, eval, w);
  gsl_vector_minmax (eval, &lambda[0], &lambda[1]);
  gsl_eigen_symm_free (w);
  gsl_vector_free (eval);
  gsl_matrix_free (Aw);
}
```