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

L'utilitaire GSL

Thème - 1 *Motivations*

La librairie GSl accessible à l'adresse `http/www.gnu.org/software/gsl` est une librairie suffisamment fournie, touchant à peu près tous les domaines de l'analyse numérique, comme le témoigne sa documentation abondante, accessible en ligne.

Cette librairie nous permettra, dans ce cours, de travailler en toute sérénité lorsque nous aurons à manipuler les matrices pleines. Elle nous offrira en effet des structures adéquates en langage C pour l'algèbre linéaire. On pourra ainsi disposer entre autres :

- d'une structure de vecteur avec toutes les opérations associées (addition, multiplication par un scalaire, norme, . . . , bref tout ce qui est reconnu dans la communauté sous la normalisation *BLAS de niveau 1 (BLAS 1)*,
- d'une structure de matrice pleine avec toutes les opérations requises (*BLAS 2 et BLAS 3*). Elle nous offrira ainsi la possibilité de réaliser des opérations comme l'inversion et la factorisation des matrices pleines, le calcul de leur déterminant, le calcul des valeurs et vecteurs propres et bien d'autres.

Elle ne nous offrira malheureusement pas une structure pour matrices creuses. Nous pourrons néanmoins nous servir des structures offertes pour construire sans difficulté majeure nos propres structures pour matrices creuses.

Thème - 2 *Description*

GSL est une bliothèque assez dense touchant à peu près tous les domaines de l'analyse numérique. La consultation sa documentation accessible en ligne permettra de s'en convaincre.

Pour le présent cours, nous nous limiterons à son module d'algèbre linéaire.

La documentation en ligne qu'elle offre est suffisamment fournie et propose de nombreux exemples.

Il nous semble beaucoup plutôt judicieux de vous y diriger.

Thème - 3 *Illustrations*

En guise d'illustration nous allons déterminer une solution approchée par différences finies du problème du Laplacien en dimension 1.

Listing 1 – gsl_illustration.c

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

```c
#include <math.h>
//#include <gsl/gsl_vector.h> // pas necessaire
#include <gsl/gsl_linalg.h>
/*************************************************/
typedef struct
{
  double a0, a1;
  int n;
  int is_param;
  double h;
  gsl_vector * x ;
} Mesh, *pMesh;
/*-------------------------------------------------*/
void mesh_alloc(pMesh m)
{
  if(!m->is_param)
    return;
  m->x =  gsl_vector_alloc (m->n);
  m->h = (m->a1 - m->a0) / (double)(m->n - 1);
  for(int i = 0; i < m->n; ++i)
    gsl_vector_set(m->x,i, m->a0 + i * m->h);
}
/*-------------------------------------------------*/
void mesh_free(pMesh m)
{
  if(m->x)
  gsl_vector_free (m->x);
  m->x = NULL;
}
/*-------------------------------------------------*/
void mesh_print(pMesh m, const gsl_vector *b)
{
  int n = m->n;
  int i;
  for(i = 0; i < n; ++i)
    printf("%f \t %f\n ", gsl_vector_get(m->x,i), gsl_vector_get(b,i));
}
/*************************************************/

typedef double (*pF) (double);

/*************************************************/
typedef struct
{
  Mesh m;
  gsl_matrix* A ;
  gsl_vector* b;
  pF f;
  pF g;
} Problem, *pProblem;
/*-------------------------------------------------*/
void problem_alloc(pProblem p)
{
  mesh_alloc(&(p->m));
  int n = p->m.n ;
  p->A = gsl_matrix_alloc(n,n);
  p->b = gsl_vector_alloc(n);
}
/*-------------------------------------------------*/
void problem_free(pProblem p)
{
  gsl_matrix_free(p->A);
  mesh_free(&p->m);
  gsl_vector_free (p->b);
}
/*-------------------------------------------------*/
void problem_print_sol(pProblem p)
{
  mesh_print(&p->m, p->b);
}
/*-------------------------------------------------*/
void problem_build_lapalce(pProblem p)
{
  int n = p->m.n;
```

```
  double h = p->m.h;
  double ih = 1. / (h * h);

  for(int i = 1; i < n - 1; ++i)
  {
    gsl_matrix_set(p->A, i, i, 2. * ih);
    gsl_matrix_set(p->A, i, i + 1, -ih);
    gsl_matrix_set(p->A, i, i - 1, -ih);
    gsl_vector_set(p->b, i, p->f( gsl_vector_get(p->m.x,i) ));
  }
  gsl_matrix_set(p->A, 0, 0, 2 * ih);
  gsl_vector_set(p->b, 0, p->g( gsl_vector_get( p->m.x,0)) * 2. * ih);
  gsl_matrix_set(p->A, n - 1, n - 1, 2. * ih);
  gsl_vector_set(p->b, n - 1, p->g( gsl_vector_get(p->m.x,n - 1)) * 2. * ih);
}
/*--------------------------------------------------*/

void problem_solve(pProblem p)
{
  int s;
  int n = p->m.n ;
  gsl_permutation * per = gsl_permutation_alloc (n);
  gsl_linalg_LU_decomp (p->A, per, &s);
  gsl_linalg_LU_solve (p->A, per, p->b, p->b);
}
/**************************************************/
void test_mesh()
{
  Mesh m = { 0, 1, 20, 1, 0, 0 };
  mesh_alloc(&m);
  mesh_print(&m, m.x);
}

/*--------------------------------------------------*/
double f(double x)
{
  return -2;
}
/*--------------------------------------------------*/
double g(double x)
{
  return x * x;
}
/*--------------------------------------------------*/

//gcc -std=c99 -I/cheminvers/gsl/include gsl_illustration1.c -o gsl_illustration -L/cheminvers↩
    /gsl/lib -lgsl -lblas -lm

int main(int argc, char **argv)
{
  int n = pow(2,7) - 1;
  Problem pb = { (Mesh) {0, 1, n, 1, 0, 0}, 0, 0 };
  problem_alloc(&pb);
  pb.f = f;
  pb.g = g;
  problem_build_lapalce(&pb);
  problem_solve(&pb);
  problem_print_sol(&pb);
  problem_free(&pb);
  return 0;
}
```

Remark 0.0.1 (Extension en deux dimensions).

L' extension de l'exemple précédent en deux dimensions d'espace est possible et ne demandera pas beaucoup plus d'effort. Une telle extension est faite dans la présentation de l'utilitaire Csparse.