

Initiation à la librairie GSL

Rémy Malgouyres

Laboratoire d'Algorithmique et d'Image de Clermont-Ferrand, IUT,
département info

B.P. 86

63172 AUBIERE cedex

<http://laic.u-clermont1.fr/~mr>

Table des matières

1	Les vecteurs et le type <code>gsl_vector</code>	2
2	Les matrices et le type <code>gsl_matrix</code>	3
3	Opérations sur les vecteurs et les matrices	5
3.1	Multiplication de deux matrices	5
3.2	Multiplication d'une matrice par la transposée d'une matrice	5
3.3	Multiplication d'une matrice par un vecteur	6
3.4	Résolution d'un système linéaire	7
3.5	Inversion d'une matrice	8
3.6	Calcul de vecteurs propres d'une matrice symétrique	9

La librairie *GSL* ou *GNU scientific library* est disponible gratuitement et permet de faire des calculs d'algèbre linéaire (multiplications et inversion de matrices, calcul de vecteurs propres, etc...). Dans cette partie, nous voyons brièvement les types et fonctions de la *GSL* qui nous permettront de calibrer une caméra par la méthode des moindres carrés comme expliqué ci-dessus.

Pour utiliser la librairie *GSL* dans un programme *C* ou *C++*, il faut inclure le fichier `<gsl/gsl_linalg.h>`.

1 Les vecteurs et le type `gsl_vector`

Un vecteur est donné par un pointeur sur `gsl_vector`. Avant d'utiliser un vecteur, il faut l'allouer avec `gsl_vector_alloc` qui prend en paramètre le nombre de coordonnées du vecteur. On peut affecter les coordonnées d'un vecteur avec `gsl_vector_set`, et on peut accéder aux coordonnées d'un vecteurs par `gsl_vector_get`. Le vecteur doit être libéré avec `gsl_vector_free`.

```

1  /* la fonction retourne le vecteur et fait un passage par adresse */
2  gsl_vector* SaisieVecteur(int *n)
3  {
4      gsl_vector *v; /* déclaration d'un vecteur */
5      int i;
6      double valeur;
7
8      puts("Veuillez entrer le nombre de coordonnées du vecteur");
9      scanf("%d", n);
10
11     v = gsl_vector_alloc(*n); /* allocation du vecteur */
12     for (i=0 ; i<*n ; i++)
13         {
14             scanf("%lf", &valeur);
15             /* initialisation de la coordonnée v[i] : */
16             gsl_vector_set(v, i, valeur);
17         }
18     return v;
19 }

```

```
1 void AfficheVecteur(gsl_vector* v, int n)
2 {
3     int i;
4     double valeur;
5     for (i=0 ; i<n ; i++)
6     {
7         /* r cup ration de la coordonn e v[i] : */
8         valeur = gsl_vector_get(v, i);
9         printf("%.3f\n", valeur);
10    }
11 }
```

```
1 int main(void)
2 {
3     gsl_vector *v; /* d claration d'un vecteur */
4     int n; /* nombre de coordonn es du vecteur */
5     v = SaisieVecteur(&n)
6     AfficheVecteur(v, n);
7     gsl_vector_free(v); /* lib ration du vecteur */
8     return 0;
9 }
```

2 Les matrices et le type `gsl_matrix`

Une matrice est donn e par un pointeur sur `gsl_matrix`. Avant d'utiliser une matrice, il faut l'allouer avec `gsl_matrix_alloc` qui prend en param tre le nombre de lignes et le nombre de colonnes de la matrice. On peut affecter les coefficients d'une matrice avec `gsl_matrix_set`, et on peut acc der aux coefficients d'une matrice par `gsl_matrix_get`. La matrice doit  tre lib r e avec `gsl_matrix_free`.

```
1  /* la fonction retourne la matrice et fait des passages par adresse */
2  gsl_matrix* SaisieMatrice(int *n, inte *m)
3  {
4      gsl_matrix *A; /* d claration d'une matrice */
5      int i, j;
6      double valeur;
7
8      puts("Veuillez entrer le nombre de lignes et de colonnes");
9      scanf("%d %d", n, m);
10     A = gsl_matrix_alloc(*n, *m); /* allocation du vecteur */
11     for (i=0 ; i<*n ; i++)
12         for (j=0 ; j<*m ; j++)
13             {
14                 scanf("%lf", &valeur);
15                 /* initialisation du coefficient A[i,j] : */
16                 gsl_matrix_set(A, i, j, valeur);
17             }
18     return A;
19 }
```

```
1  void AfficheMatrice(gsl_matrix* A, int n, int m)
2  {
3      int i, j;
4      double valeur;
5      for (i=0 ; i<n ; i++)
6          {
7              for (j=0 ; j<m ; j++)
8                  {
9                      /* r cup ration du coefficient A[i,j] : */
10                     valeur = gsl_matrix_get(A, i, j);
11                     printf("%.3f ", valeur);
12                 }
13             printf("\n");
14         }
15 }
```

```
1  int main(void)
2  {
3      gsl_matrix *A; /* d claration d'une matrice */
4      int n, m; /* nombre de lignes et de colonnes */
5      A = SaisieMatrice(&n, &m)
6      AfficheMatrice(A, n, m);
7      gsl_matrix_free(A); /* lib ration de la matrice */
8      return 0;
9  }
```

3 Opérations sur les vecteurs et les matrices

Pour utiliser les opérations sur les matrices de la librairie *cblas* de la *GSL* dans un programme *C* ou *C++*, il faut inclure le fichier `<gsl/gsl_blas.h>`.

3.1 Multiplication de deux matrices

Pour multiplier une matrice *A* par une matrice *B*, on utilise la fonction `gsl_blas_dgemm`.

```
1 void MutliplieMatriceMatrice(void)
2 {
3     gsl_matrix *A, *B, *C;
4     int na, ma, nb, mb;
5
6     A = SaisieMatrice(&na, &ma);
7     B = SaisieMatrice(&nb, &mb);
8
9     if (ma != nb)
10    {
11        puts("Erreur, dimensions incompatibles");
12        return;
13    }
14
15    /* calcul de C = A*B : */
16    C = gsl_matrix_alloc(na, mb); /* allocation aux bonnes dimensions */
17    gsl_blas_dgemm(CblasNoTrans, CblasNoTrans, 1.0, A, B, 0.0, C);
18
19    AfficheMatrice(C); /* affichage du resultat */
20
21    gsl_matrix_free(A); /* libération des matrices */
22    gsl_matrix_free(B);
23    gsl_matrix_free(C);
24 }
```

3.2 Multiplication d'une matrice par la transposée d'une matrice

Pour multiplier la transposée A^T d'une matrice *A* par une matrice *B*, on utilise aussi la fonction `gsl_blas_dgemm`. La différence est dans l'option `CblasTrans`.

```
1 void MutliplieTranspose(void)
2 {
3     gsl_matrix *A, *B, *C;
4     int na, ma, nb, mb;
5
6     A = SaisieMatrice(&na, &ma);
7     B = SaisieMatrice(&nb, &mb);
8
9     if (na != nb)
10    {
11        puts("Erreur, dimensions incompatibles");
12        return;
13    }
14
15    /* calcul de  $C = A^T * B$  : */
16    C = gsl_matrix_alloc(ma, mb); /* allocation aux bonnes dimensions */
17    gsl_blas_dgemm(CblasTrans, CblasNoTrans, 1.0, A, B, 0.0, C);
18
19    AfficheMatrice(C); /* affichage du r sultat */
20
21    gsl_matrix_free(A); /* lib ration des matrices */
22    gsl_matrix_free(B);
23    gsl_matrix_free(C);
24 }
```

3.3 Multiplication d'une matrice par un vecteur

Pour multiplier une matrice par un vecteur, on utilise la fonction `gsl_blas_dgemv`.

```
1 void MultiplieMatriceVecteur(void)
2 {
3     gsl_vector *v, *res;
4     gsl_matrix *A;
5     int na, ma, nv;
6
7     v = SaisieVecteur(&nv);
8     A = SaisieMatrice(&na, &ma);
9     if (nv != ma)
10    {
11        puts ("Erreur, dimensions incompatibles");
12        return;
13    }
14
15    /* calcul de res = A*v : */
16    res = gsl_vector_alloc(nv),
17    gsl_blas_dgemv(CblasNoTrans, 1.0, A, v, 0.0, res);
18
19    AfficheVecteur(res);
20
21    gsl_matrix_free(A); /* lib ration des matrices */
22    gsl_vector_free(v); /* lib ration des vecteurs */
23    gsl_vector_free(res);
24 }
```

3.4 R solution d'un syst me lin aire

Pour r soudre un syst me lin aire $Mx = b$, on utilise les fonctions `gsl_linalg_LU_decomp` et `gsl_linalg_LU_solve`. Ce proc d  d truit la matrice, et il faut donc en faire une copie si l'on souhaite la r utiliser. Il faut inclure `gsl/gsl_linalg.h` et `gsl/gsl_cblas.h`.

```
1  gsl_vector* ResoudSysteme(gsl_matrix *M, gsl_vector* b)
2  {
3
4      gsl_matrix *copieM, *Mmoins1, *Identite;
5      gsl_vector *b;
      gsl_vector *x;
      gsl_vector *y;
      gsl_permutation *p;
      if (M->size1 != M->size2)
          return NULL;
      n = M->size1;
      copieM = gsl_matrix_alloc(n,n);
      Mmoins1 = gsl_matrix_alloc(n,n);
      b = gsl_vector_alloc(n);
      x = gsl_vector_alloc(n);
      y = gsl_vector_alloc(n);
      p = gsl_permutation_alloc(n);

      //calcul de la solution de Mx = b
      gsl_matrix_memcpy(copieM, M);
      gsl_linalg_LU_decomp(copieM, p, &s);
      gsl_linalg_LU_solve(copieM, p,b, x);

      gsl_matrix_free(copieM);
      gsl_permutation_free(p);
      return x;
  }
```

3.5 Inversion d'une matrice

Pour inverser une matrice, on utilise les fonctions `gsl_linalg_LU_decomp` et `gsl_linalg_LU_invert`. Ce procédé détruit la matrice, et il faut donc en faire une copie si l'on souhaite la réutiliser. Il faut inclure `gsl/gsl_linalg.h` et `gsl/gsl_cblas.h`.

```
1  gsl_matrix* Inverse(gsl_matrix *M)
2
3  gsl_matrix *copieM, *Mmoins1;
4  gsl_permutation *p;
5  if (M->size1 != M->size2)
    return NULL;
    n = M->size1;
    copieM = gsl_matrix_alloc(n,n);
    Mmoins1 = gsl_matrix_alloc(n,n);
    p = gsl_permutation_alloc(n);

    // Inversion de la matric M
    gsl_matrix_memcpy(copieM, M);
    gsl_linalg_LU_decomp(copieM, p, &s);
    gsl_linalg_LU_invert(copieM, p, Mmoins1);

    gsl_matrix_free(copieM);
    gsl_permutation_free(p);
    return Mmoins1;
```

3.6 Calcul de vecteurs propres d'une matrice sym trique

Pour calculer des valeurs propres et vecteurs propre de la *GSL* dans un programme *C* ou *C++*, il faut inclure le fichier `<gsl/gsl_eigen.h>`.

Pour calculer les vecteurs propres d'une matrice sym trique, on utilise la fonction `gsl_eigen_symmv`. Il faut pour cela allouer un espace de travail avec la fonction `gsl_eigen_symmv_alloc`. On r cup re les valeurs propres dans un vecteur `eval`, et les vecteurs propres dans les colonnes d'une matrice `avec`.

```
1 void CalculVecteursPropresSymmetrique(void)
2 {
3     gsl_matrix *A;
4     int i, j, n, m;
5
6     /* d claration des variables n cessaires */
7     /* au calcul des vecteurs propres : */
8     gsl_matrix *evec;
9     gsl_vector *eval;
10    gsl_eigen_symmv_workspace *w;
11
12    puts("Veuillez saisir une matrice sym trique :");
13    A = SaisieMatrice(&n,&m);
14
15    /* calcul des vecteurs propres de A : */
16    copieA = gsl_matrix_alloc(n,m);
17    gsl_matrix_memcpy(copieA, A); /* sauvegarde de A */
18    w = gsl_eigen_symmv_alloc(n);
19    evec = gsl_matrix_alloc(n,n);
20    eval = gsl_vector_alloc(n);
21    gsl_eigen_symmv(copieA, eval, evec, w);
22
23    /* affichage des r sultats : */
24    for (j=0 ; j<m ; j++)
25        {
26            printf("La valeur propre num ro %d est %f\n",
27                j, gsl_vector_get(eval, j));
28            puts("et le vecteur propre correspondant est :");
29            for (i=0 ; i<n ; i++)
30                printf("%f ", gsl_matrix_get(evec, i, j));
31        }
32    gsl_matrix_free(A); /* Lib ration de m moire */
33    gsl_matrix_free(copieA);
34    gsl_matrix_free(evec);
35    gsl_vector_free(eval);
36    gsl_eigen_symmv_free(w);
37 }
```