A set of C functions to compute rotation matrices and spherical harmonics expansions

Didier Pinchon

July 13, 2011

1 Introduction

In a recent paper[1], Philip Hoggan and I have introduced a new algorithm to compute general rotation matrices using the precomputation of matrices J_l for $l \geq 1$. A package of C functions has been distributed that computes rotation matrices for $1 \leq l \leq l_{\max}$ provided the coefficients of matrices J_l are stored, in a compact way, inside a file. The main functions of this package are functions that compute simultaneously all the rotation matrices for $0 \leq l \leq l_{\max}$ and given Euler angles or all rotated vectors of spherical harmonics expansions. Other functions are also provided to compute individually, for a given value of l, the rotation matrix or a rotated vector of size $2l+1$.

This version 2.0 improves a previous version 1.0 and is distributed under the GNU General Public License (see file COPYING).

2 Description of the code

Two structures `VecL` and `MatL` are introduced to represent a list of vectors $V_L = (V_l)_{0 \leq l \leq L}$ and a list of matrices $M_L = (M_l)_{0 \leq l \leq L}$, where V_l has $2l+1$ elements and M_l is a $(2l+1) \times (2l+1)$ matrix:

```
typedef struct S_VECL {
   int lmax;
   gsl_vector **lvec;
} VecL;

typedef struct S_MATL {
   int lmax;
   gsl_matrix **lmat;
} MatL;
```

These structures and the signature of available functions are defined in file `rotation.h`. The code of the functions are located in the following C files.

alloc.c

[1] D. P. and Ph. Hoggan, *Rotation matrices for real spherical harmonics: general rotations of atomic orbitals in pace-fixed axes*, J. Phys. A 40(2007), 1597-1610.

This file contains four functions to allocate, for a given value of L, and desallocate space for `VecL` and `MatL` objects.

```
VecL *vecl_calloc(int lmax);
void  vecl_free(VecL *vecl);
MatL *matl_calloc(int lmax);
void  matl_free(MatL *matl);
```

matj.c

The core of the algorithm presented in our paper is the precomputation of a set of matrices $(J_l)_{0\leq l\leq L}$. Two functions allows to recover J_l matrices whose coefficients are stored in file `Bin/numMatJ.dat` for $2 \leq l \leq L$ or individually for each value of l in file `Bin/NumMatJ/numMatJ-l.dat`. These last files may be extracted from `Bin/numMatJ.dat`. For reason of place, `Bin/numMatJ.dat` is provided in this distribution for $L = 150$ and individual `numMatJ-l.dat` for $2 \leq l \leq 50$.

`MatL *read_lmatJ(int lmax)` recovers $(J_l)_{0\leq l\leq L}$.

`gsl_matrix *read_matJ(int l)` recovers an individual matrix J_l.

rotation_matrix.c

`int matl_matl_product(const MatL *matl1, const MatL *matl2, MatL *matres)`
computes the list of product matrices $C_L = (C_l)_{0\leq l\leq L}$ where $C_l = A_l B_l$ given $A_L = (A_l)_{0\leq l\leq L}$ and $B_L = (B_l)_{0\leq l\leq L}$.

```
int matl_rotation(int lmax,double alpha, double beta, double gamma,MatL *res,
       const MatL *matj)
```

computes the rotation matrices $R_L = (R_l)_{0\leq l\leq L}$ for a set of Euler' angles α, β, γ.

```
int matrix_rotation(int l,double alpha, double beta, double gamma,gsl_matrix *res,
      const gsl_matrix *matj)$}
```

computes R_l for given angles α, β, γ.

Four local fonctions are defined in `rotation_matrix.c`

```
int Zalpha_matl_product(const MatL *matl, const double alpha,MatL *res)
int matl_Zalpha_product(const MatL *matl, const double alpha,MatL *res)
```

to compute $(Z_l(\alpha)M_l)_{0\leq l\leq L}$ and $(M_l Z_l(\alpha))_{0\leq l\leq L}$ where $Z_l(\alpha)$ is the rotation matrix of a rotation with z-axis and angle α. in these two functions the iteration on l is done in an inner loop to compute the products row by row. This allows to compute $\cos l\alpha, 0 \leq l \leq L$ only once. However this may present the following disadvantages for computation efficiciency: an access to distant elements during the computation and a drawback to parallelization.

```
int Zalpha_mat_product(int l,const gsl_matrix *mat, const double alpha,gsl_matrix *res)
int mat_Zalpha_product(int l,const gsl_matrix  *mat, const double alpha,gsl_matrix  *res)
```

compute $Z_l(\alpha)M_l$ and $M_l Z_l(\alpha)$ for a given value of l.

rotation_vector.c

`int matl_vecl_product(const MatL *matl, const VecL *vecl, VecL *res)`
computes $W_L = (W_l)_{0\leq l\leq L}$ where $W_l = A_l V_l$ given a `MatL` set of matrices $A_L = (A_l)_{0\leq l\leq L}$ and a `VecL` set of vectors $V_L = (V_l)_{0\leq l\leq L}$.

```
int vecl_rotation(int lmax,double alpha, double beta, double gamma, const VecL *x,
      VecL *res, const MatL *matj)
```

computes the `VecL` set of vectors $(R_l V_l)_{0\leq l\leq L}$ for a rotation given by its Euler' angles.

```
int vector_rotation(int l,double alpha, double beta, double gamma,const gsl_vector *x,
        gsl_vector *res,const gsl_matrix *matj)
```

does the same job for a single value of l.

Two local function are provided to compute $(Z_l(\alpha)V_l)_{0\leq l\leq L}$ or $Z_l(\alpha)V_l$ for a single value of l:

```
int Zalpha_vecl_product(const VecL *vecl, const double alpha,VecL *res)
int Zalpha_vec_product(int l,const gsl_vector *vec, const double alpha,gsl_vector *res)
```

3 Test programs

Two test programs are located in directory `Src/Util`.

test_rotation.c

It is called by the command `./test_rotation lmax alpha beta gamma` to check several features to insure that rotation functions do the right job. Here is an example of execution:

```
./test_rotation 40 0.3 0.7 0.9

1 -> Global rotation in MatL and rotations by bands in gsl_matrices are equivalent
2 -> Rotating all SH coefficients for lmax = 40 by two methods gives equivalent results,
        dist = 8.8817841970012523e-16
3 -> Rotating SH coefficients for each l=0..40  gives identical results,
        dist = 0.0000000000000000e+00
4 -> Conservation of the squared l2-norm by rotation :
        5.6276343149376009e+02, 5.6276343149375964e+02 -> err rel = 8.0806130149468742e-16
5 -> Inverse rotation applied to the rotated vector : dist = 7.7715611723760958e-16
```

time_rotation.c

This program evaluates times to compute $R_l V_l$ for each l with $2 \leq l$ L for a fixed random set $(V_L = (V_l)_{0\leq l\leq L}$ and a number of random choice of α, β, γ for the rotation by two methods. In the first method, matrix R_l is computed and then applied to vector V_l while in the second method uses only matrix-vector product using the product decomposition of R_l described in our paper.
The command is `./time_rotation lmax nb_test` where `lmax` stands for the maximum value L of l and `nb_test` is the number of random rotations. A second run of each computation, without

timing, is done to check that the methods provide equivalent results (not exactly tha same because the order of arithmetic operations is not the same). Execution times are recorded in a file `res.dat`. The following graphic has been done using results in `res.dat.save` obtained by the command `./time_rotation 150 10000`.