New C functions to compute rotation matrices

Didier Pinchon

January 16, 2010

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 function of this package is the function `matl_rotation()` that computes simultaneously all the rotation matrices for $2 \leq l \leq l_{\max}$ and given Euler angles.

However, as noted by Christian Lessig in recent mail exchanges, the performance of the function `matl_rotation()` is very poor In this note, I describe a new set of function to compute a rotation matrix for a given value of l and Euler angles. Performance comparison are done between the old and new functions.

The code for the new package is distributed in an archive file **Rotation2.tar.gz** that contains in particular

- The source code in C for the functions of a library in directory `Rotation2/Src/Base`,
- The source code in C for two application programs in directory `Rotation2/Src/Util`,
- The trace of some of their execution results in directory `Rotation2/Src/Maple`.

However, for size reasons files, files containing the precomputed entries of J_l matrices that should be present in directory `Rotation2/Src/Maple/NumMatJ` has not been stored in the archive file. Only the files for $1 \leq l \leq 50$ are given in the archive file. Maple programs are given to generate the files for $51 \leq l \leq 200$.

2 Results

Four new functions have been introduced in source file `rotation.c`:

1. `gsl_matrix *read_matJ(int l)` that reads the entries of J_l for a given l in file `Maple/NumMatJ/numMatJ-l.dat` and construct the symmetric square matrix J_l of size $2l+1$.

[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.

2. `mat_product_Zalpha_left(int l,const gsl_matrix *matl1, const double alpha, gsl_matrix *res)`
   that computes the product $Z_\alpha A$ where A is matrix of size $2l+1$ and Z_α is the rotation matrix of size $2l+1$ for a rotation with axis Oz and angle α.

3. `mat_product_Zalpha_right(int l,const gsl_matrix *mat1, const double alpha, gsl_matrix *res)`
   that does the same job to compute a product AZ_α.

4. `mat_rotation(int l,double alpha, double beta, double gamma,gsl_matrix *res, const gsl_matrix *matj)`
   that compute the rotation matrix $R = Z_\alpha J_l Z_\beta J_l Z_\gamma$ for given values of l and Euler's angles α, β, γ.

These four function use the C-type `gsl_matrix` from the GNU Scientific Library (GSL). The code of function `mat_rotation()` is very short :

```
int mat_rotation(int l,double alpha, double beta, double gamma,gsl_matrix *res,
                 const gsl_matrix *matj)
{
  gsl_matrix *tmp;

  tmp = gsl_matrix_calloc(2*l+1,2*l+1);

  mat_product_Zalpha_left(l,matj,beta,tmp);
  gsl_blas_dgemm(CblasNoTrans,CblasNoTrans,1.0,matj,tmp,0.0,res);
  mat_product_Zalpha_left(l,res,alpha,tmp);
  mat_product_Zalpha_right(l,tmp,gamma,res);

  gsl_matrix_free(tmp);
  return EXIT_SUCCESS;
}
```

A first version uses the C-blas function `gsl_blas_dsymm`, because `matj` is a symmetric matrix with the line

```
gsl_blas_dsymm(CblasLeft, CblasLower, 1.0, matj, tmp, 0.0, res);
```

instead of `gsl_blas_dgemm(CblasNoTrans,CblasNoTrans,1.0,matj,tmp,0.0,res);`
As shown in Figure 1, it was not a good idea, at least on my Linux workstation.

The program `test_rotation` that received four arguments

```
test_rotation lmax alpha beta gamma
```

with $l_{\max} \leq 20$, checks that the computed rotation matrices for $2 \leq l \leq l_{\max}$ are exactly the same when computed by the old function `matl_rotation()` with one call and the new function `mat_rotation()` with one call per value of l.

The program `time_rotation` that received two arguments

```
test_rotation lmax maxiter
```

computes `maxiter` rotation matrices for each values of l such that $2 \leq l \leq l_{\max}$. The three Euler's angles are chosen at random in their definition domain $0 \leq \alpha, \gamma \leq 2\pi$ and $0 \leq \beta \leq \pi$ with an

uniform distribution. This doesn't correspond to the uniform distribution on $SO(3,\mathbb{R})$ but was estimated to be sufficient to test the time performance of function `mat_rotation`.
Figure 1 gives the mean time in seconds for one call of this function as a function of l. The upper curve was obtained for $l_{\max} = 200$ and 100 runs for each value of l, when using the C-blas function `gsl_blas_dsymm` and the lower with `gsl_blas_dgemm`.

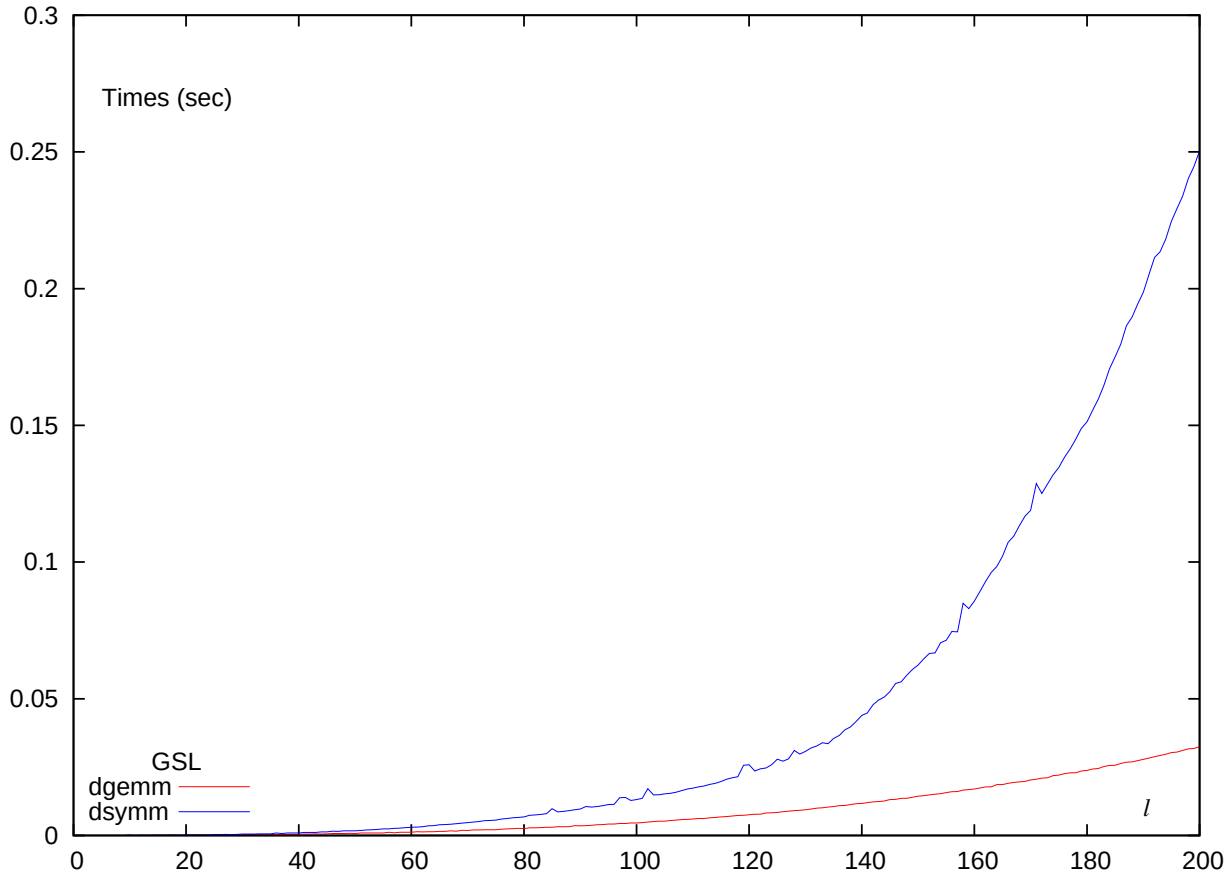

Figure 1: Comparison between programs using dsymm or dgemm GSL routines.

Figure 2 shows the final performance curve. Using a linear regression to find the best constants a and b such that $\log T(l) \sim a \log l + b$ where $T(l)$ is the mean time to compute a rotation matrix for l with $100 \leq l \leq 200$ gives

$$T(l) \sim 2.81 \log l - 18.34 \ .$$

So $T(l) = O(l^3)$ where the proportionality constant is machine dependent, this result beeing confirmed by a precise study of the arithmetic complexity of the algorithm.

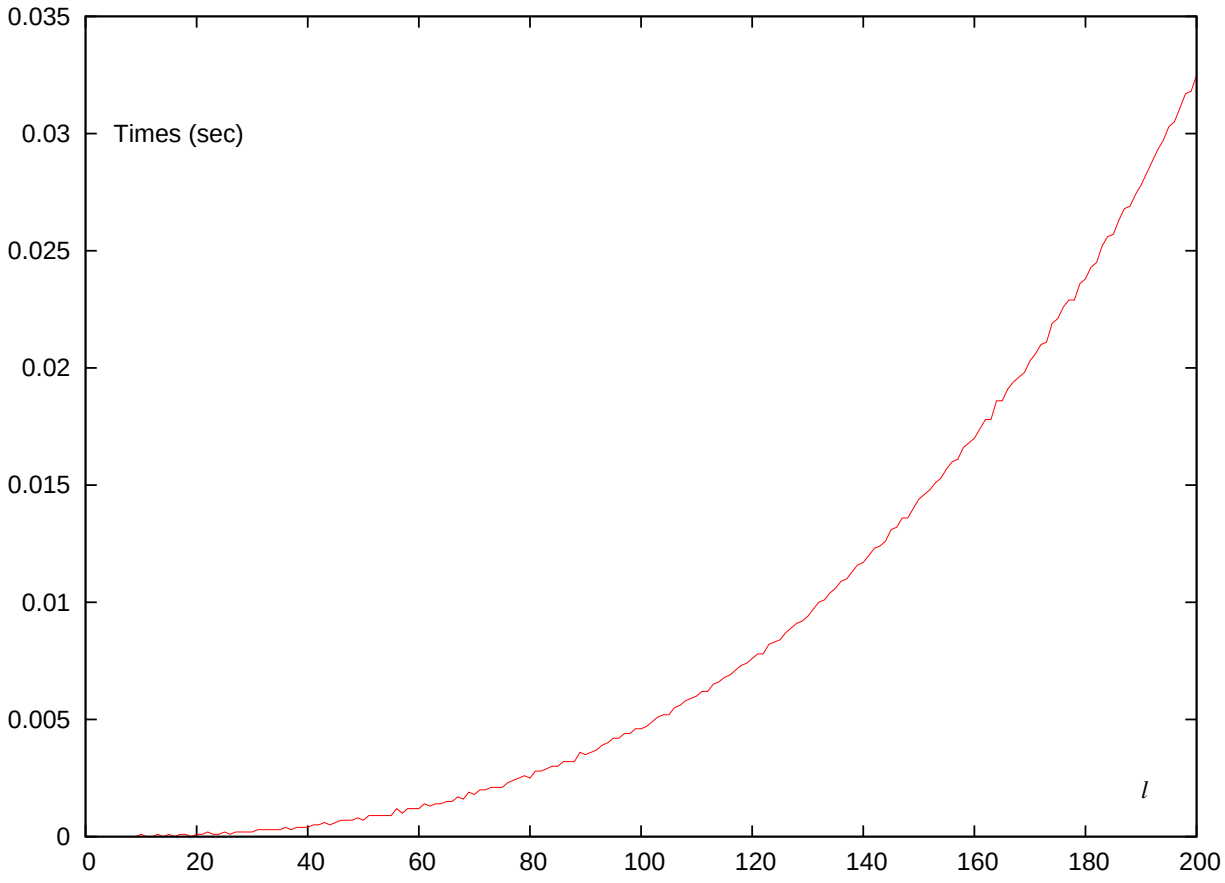

Figure 2: Time for computing a rotation matrix.