

Documentation for *Tops*: Classes for Computing Rigid Body Dynamics version 2

Ramses van Zon*

*Chemical Physics Theory Group, Department of Chemistry, University of Toronto,
80 St. George Street, Toronto, Ontario M5S 3H6, Canada*

May 26, 2009

Abstract

This document describes how to use the c++ Top classes defined in the header file tops.h and implemented in the file tops.cc. These classes can efficiently compute the exact rotational dynamics of an arbitrary rigid body in the absence of forces, and have been written to be easy to use.

*rzon@chem.utoronto.ca

Contents

1	Introduction	3
2	Changes compared to version 1	4
3	Using tops	4
4	Initialization	5
5	Initial conditions	6
6	Computing rotational dynamics	6
7	Propagation	6
8	Example	7
9	Minimizing drift	8
10	Changing the precision	8
11	The classes	8
	11.1 The abstract base class Top	8
	11.2 TopSymmetric	9
	11.3 TopOblate	9
	11.4 TopProlate	10
	11.5 TopAsymmetric	10
	11.6 TopRecur	10
	Background references	11

1 Introduction

Rigid body dynamics is encountered in many physical models, such as for molecules, polymers, robotics and even the computer game industry. The classes defined in *tops* give an easy-to-use implementation of the exact rotation for arbitrary rigid bodies.

Before explaining how to use the classes in *tops.h*, it is necessary to explain a bit of the notation concerning rigid bodies.

A rigid body is a body in which the relative positions of all its parts are fixed. This still allows for the body to translate and to rotate. For a free rigid body, translation is governed by the linear velocity \mathbf{V} , such that the position \mathbf{R}_t of the mass center of the body at time t is related to that at time 0 by $\mathbf{R}_t = \mathbf{R}_0 + \mathbf{V}t$. This is so simple that there is no need to write a specialized c++ class for translation. For that reason, *tops* only deals with the rotational body of the motion.

The rotation takes place relative to the center of mass. The body have an arbitrary orientation (or “attitude”). Suppose that a special reference orientation has been chosen. Then any other orientation can be obtained by rotating the body. A rotation can be represented by a 3x3 orthogonal matrix A , which also represents the orientation. In this context, A is called an attitude matrix. If the position of a point on the body in the reference orientation is $\tilde{\mathbf{r}}$, then in the rotation orientation, its position is $A^T \cdot \tilde{\mathbf{r}}$, where A^T is the transpose of A .¹

For a rotating rigid body the attitude matrix A changes in time, but it will always remain orthogonal. The change is governed by the angular velocity vector ω . For a body with angular velocity vector ω , the velocity of a point \mathbf{r} is $\omega \times \mathbf{r}$. Different from the velocities for the velocity in translational motion, the angular velocity vector is not necessarily constant in time even for a free body. How A changes in time depends on the body inertial moment matrix. This matrix is symmetric and can be diagonalized. The orientation of the body in which the inertial moment matrix is diagonal will be taken as the reference orientation, and the resulting diagonal elements I_x , I_y and I_z are called the principal moments of inertia. The angular velocity vector is constant only if the principal moments of inertia are all the same. We then say that the body is “a spherical top” (as far as the rotation is concerned). Another kind of body is the symmetric top, for which two moments of inertia are the same. If the equal moments of inertia are smaller than the unequal one, the body is called “oblate”, while if the equal moments of inertia are larger than the unequal one, the body is called “prolate”. Finally, if all moments of inertia are different the body is said to be “asymmetric”. This same nomenclature is used for the names of the Top classes in *tops.h*.

In all cases, the rotation of the body can be solved exactly, although the mathematics gets somewhat involved for an asymmetric body[1, 2, 3]. This does not mean, however, that this solution cannot be implemented efficiently, and this is what *tops* does.

¹The transpose is equal to the inverse of a rotation matrix.

2 Changes compared to version 1

- Elliptic integrals and functions needed to compute the dynamics of an asymmetric top are now coded into *tops*.
- The GSL library is therefore no longer required.
- This also made the AsymmetricTop class much faster, because a lot of computations could be combined.
- A bug in ProlateTop was fixed.
- The other Top classes perform as before.
- This second version has the capability to minimize drift (switched off by default). The amount of drift left depends on the compiler and the computer architecture.
- There is no implementation of version 2 in c (yet).

3 Using tops

To use *tops*, the following general procedure should be followed:

- The Top classes use vectors and matrices defined in the header file `vecmat3.h`. While `tops.h` includes this file automatically, it needs to be in the same directory as the `tops.h` file.² *Vecmat3* is an efficient implementation of three dimensional vectors and matrices which is strongly recommended.³

The Top classes cannot be used if `vecmat3.h` cannot be found!

- Include the header file `tops.h`:

```
#include "tops.h"
```

- The classes `Top`, `TopSpherical`, `TopProlate`, `TopOblate`, `TopAsymmetric` and `TopRecur` and are now defined and can be used as explained in the next section.
- The implementation of these classes can be found in the source file “`tops.cc`”, which needs to be compiled and linked with any program using *tops*.
- Compiling or linking your program, can for instance be done as follows (for `test-tops.cc` in Sec. 8)

²or in a directory that is searched for header files by the compiler.

³See `docvecmat3.pdf`.

```
c++ testtops.cc tops.cc -lm -o testtops
```

or, if drift correction is needed,

```
c++ testtops.cc tops.cc -DREFINE -lm -o testtops
```

with the files tops.cc, tops.h and vecmat3.h in the current directory.

4 Initialization

The Top classes can be initialized as follows:

```
TopSpherical   sphere (Ix);           // Ix = Iy = Iz
TopProlate     prolate(Ix, Iz);       // Ix < Iy = Iz
TopOblate      oblate (Ix, Iz);       // Ix = Iy < Iz
TopAsymmetric asymtop(Ix, Iy, Iz);    // Ix < Iy < Iz
TopRecur       recurse(Ix, Iy, Iz);    // Ix < Iy < Iz, faster implementation
```

Notes:

1. These classes require arguments, i.e., one cannot define a Top without specifying its moments of inertia.
2. Note that the Top classes will not find the moments of inertia (nor the body reference frame) of a given body, rather, these are needed as input.
3. For spherical top, as single moment of inertia is enough, for the two types of symmetric tops, one needs two, while in general three moments of inertia are required.
4. The ordering of the moments of inertia indicated above is required, i.e., the specified moments of inertia need to be given *in ascending order*.
5. As the names suggest, TopSpherical computes the rotation of a spherical top, TopProlate, that of a prolate symmetric top, TopOblate that of an oblate symmetric top, and TopAsymmetric that of an asymmetric Top. All of these classes are derived from a parent class Top, in which the member functions discussed below are virtually overloaded.
6. The class TopRecur is a variant of TopAsymmetric which works only for small enough times, as explained below, but which is considerably faster than TopAsymmetric.

5 Initial conditions

Given an object `top` of any of the above mentioned Top classes, one can set the initial angular velocity ω_0 (denoted by `omega0` in the code) and the initial attitude matrix A_0 (simply `A0`), using

```
Vector omega0 (1,2,3);
Matrix A0 (0, 1, 0,
          1, 0, 0,
          0, 0,-1);
top.Initialization( omega0, A0 );
```

6 Computing rotational dynamics

Once the moments of inertia have been specified and the initial conditions are set, the angular velocity ω and attitude matrix A at time t can be computed as follows:

```
Vector omega;
Matrix A;
double t = 1.5; // arbitrary time
top.Evolution(t, omega, A);
cout << omega << A; // write out the result
```

Note that one can request the the angular velocity ω and attitude matrix A at several times without having to specify the moments of inertia or the initial conditions again.

7 Propagation

In some applications, one only needs to update the old ω and A to new ones a time interval dt later. For this purpose, the Top classes contain a function `Propagate`, to be used as follows:

```
Vector omega (1,2,3);
Matrix A ( 0, 1, 0,
          1, 0, 0,
          0, 0,-1);
double t = 1.5;
top.Propagate(t, omega, A);
cout << omega << A; // write out the result
```

Note that after calling `Propagate`, the original value of `omega` and `A` are lost and replaced by their new values.

Repeated application of Propagate can lead to drift in energy, angular momentum and orthogonality of the A matrix, which is addressed in Sec. 9.

8 Example

```
#include <fstream>
#include "tops.h"
using namespace std;
void testTop( Top & rotor, const char* filename ) {
    Vector omega0 ( 1.0, -1.0, 0.5 );
    Matrix A0 ( 1,0,0,
               0,1,0,
               0,0,1 );
    rotor.Initialization( omega0, A0 );
    ofstream f( filename );
    for ( double t = 0.0; t < 12.0; t += 0.1 ) {
        Vector omega;
        Matrix A;
        rotor.Evolution( t, omega, A );
        f << fixed << t
          << ' ' << A.row(0)
          << ' ' << A.row(1)
          << ' ' << A.row(2)
          << ' ' << omega
          << endl;
    }
}
int main() {
    double Ix = 1.0, Iy = 1.5, Iz = 2.0;
    TopSpherical sphere (Ix);           // Ix = Iy = Iz
    TopProlate prolate(Ix, Iz);         // Ix < Iy = Iz
    TopOblate oblate (Ix, Iz);         // Ix = Iy < Iz
    TopAsymmetric asymtop(Ix, Iy, Iz); // Ix < Iy < Iz
    TopRecur recurse(Ix, Iy, Iz);
    testTop( sphere, "sphere.dat" );
    testTop( prolate, "prolate.dat" );
    testTop( oblate, "oblate.dat" );
    testTop( asymtop, "asymtop.dat" );
    testTop( recurse, "recurse.dat" );
}
```

9 Minimizing drift

This second version of *tops* has the capability to minimize drift in energy, angular momentum and orthogonality, using some (not all work) of the ideas of Vilmart[4], and some addition least significant bit twiddling. This drift is due to cumulative effects of round-off bias in the last significant digits which occurs when the Propagation method is called in frequently repetition.

The drift is often so small that it is a negligible effect. Because correcting for drift increases the computational cost, the drift correction code is turned off by default.

To switch on the drift corrections, define the switch “REFINE”. When switched on, the drift is less than one bit per iteration (on average). How much drift is left depends on the compiler and the machine that the code it run on.

10 Changing the precision

All top classes use the precision of the *vecmat3* module, which is double precision by default. This is achieved by defining a type `DOUBLE`, which defined in *vecmat3.h* to be equal to `double` by default.

It is however easy to use floating point numbers of different precision, such as `float` or `long double`. One only has to make sure `DOUBLE` is already `#define`'s as the correct type before *tops* is called, i.e., by putting `#define DOUBLE float` before the `#include "vecmat3.h"` in the file *tops.h*.

11 The classes

11.1 The abstract base class Top

```
class Top
{
    public:
        virtual void Initialization(const Vector& omega, const Matrix& A) = 0;
        virtual void Evolution(DOUBLE t, Vector& omega, Matrix& A) = 0;
        virtual void Propagation(DOUBLE dt, Vector& omega, Matrix& A) = 0;
        virtual ~Top()
};
```

Below, only the public parts of the derived Top classes will be shown. There are five derived classes:

1. TopSpherical : for spherical tops, for which $I_x = I_y = I_z$. Based on [1] and [2].
2. TopProlate : for prolate tops, for which $I_x \neq I_y = I_z$. Based on Refs. [1] and [2].
3. TopOblate : for spherical tops, for which $I_x = I_y \neq I_z$. Based on Refs. [1] and [2].
4. TopAsymmetric: for asymmetric tops, for which $I_x \neq I_y \neq I_z$. Based on Ref. [2].
5. TopRecur : also for asymmetric tops, but computed using a recursive scheme. Based on Ref. [3].

The latter to use the arithmetic geometric scale to compute elliptic integral and functions in a highly optimized fashion[5].

11.2 TopSymmetric

```
class TopSymmetric: public Top
{
    public:
        TopSymmetric(DOUBLE I);
        void Initialization(const Vector& omega, const Matrix& A);
        void Evolution(DOUBLE t, Vector& omega, Matrix& A);
        void Propagation(DOUBLE dt, Vector& omega, Matrix& A);
        ~TopSymmetric()
};
```

11.3 TopOblate

```
class TopOblate: public Top
{
    public:
        TopOblate(DOUBLE Ix, DOUBLE Iz);
        void Initialization(const Vector& omega, const Matrix& A);
        void Evolution(DOUBLE t, Vector& omega, Matrix& A);
        void Propagation(DOUBLE dt, Vector& omega, Matrix& A);
        ~TopOblate()
};
```

11.4 TopProlate

```
class TopProlate: public Top
{
    public:
        TopProlate(DOUBLE Ix, DOUBLE Iz);
        void Initialization(const Vector& omega, const Matrix& A);
        void Evolution(DOUBLE t, Vector& omega, Matrix& A);
        void Propagation(DOUBLE dt, Vector& omega, Matrix& A);
        ~TopProlate()
};
```

11.5 TopAsymmetric

```
class TopAsymmetric: public Top
{
    public:
        TopAsymmetric(DOUBLE Ix, DOUBLE Iy, DOUBLE Iz);
        void Initialization(const Vector& omega, const Matrix& A);
        void Evolution(DOUBLE t, Vector& omega, Matrix& A);
        void Propagation(DOUBLE dt, Vector& omega, Matrix& A);
        ~TopAsymmetric()
};
```

11.6 TopRecur

```
class TopRecur: public Top
{
    public:
        TopRecur(DOUBLE Ix, DOUBLE Iy, DOUBLE Iz);
        void Initialization(const Vector& omega, const Matrix& A);
        void Evolution(DOUBLE t, Vector& omega, Matrix& A);
        void Propagation(DOUBLE dt, Vector& omega, Matrix& A);
        ~TopRecur()
};
```

Background references

- [1] Lisandro Hernandez de la Pena, Ramses van Zon, Jeremy Schofield and Sheldon B. Opps, *Discontinuous molecular dynamics for semi-flexible and rigid bodies*, Journal of Chemical Physics **126**, 074105 (2007).
- [2] Ramses van Zon and Jeremy Schofield, *Numerical implementation of the exact dynamics of free rigid bodies*, Journal of Computational Physics **225**, 145 (2007).
- [3] Ramses van Zon and Jeremy Schofield, *Symplectic algorithms for simulations of rigid-body systems using the exact solution of free motion*, Physical Review E **75**, 056701 (2007).
- [4] G. Vilmart, *Reducing round-off errors in rigid body dynamics*, Journal of Computational Physics **227**, 7083 (2008).
- [5] M. Abramowitz and I. A. Stegun, *Handbook of Mathematical Functions with formulas, graphs, and mathematical tables* (Dover, New York, 1965).