File:brownrig-v1c

BrownRig

Version 1.c. September, 2006

M.X. Fernandes, A. Ortega and J. García de la Torre

Departamento de Química Física
Universidad de Murcia, Spain

Index

1. Introduction to BrownRig

The **`BrownRig`** software is intended to simulate the Brownian dynamics of arbitrary shaped rigid particles under the action of external agents.

The program has two essential aspects, the first of which is the interaction of the particle with its environment. The interaction takes place in one, or both, of the following ways:

1) The external agent produces some forces at specific points (force points) of the particle. Thus the particle moves under the action of a force, which is the sum of all the individual forces, and a torque, that is the sum of the torques of these forces. A common example of this will be a particle bearing charges in an electric field.

2) The particle is in the presence of barriers, enclosed in a cavity, etc. Particularly, there are some specific points (which may or may not be the same as the force points) or "elements" in the particle which cannot trespass the walls.

The second essential aspect is the particle hydrodynamics. The algorithm implements a fully anisotropic motion of the particle, meaning that diffusion along the three spatial directions takes into account the different drag due to the shape details of the arbitrary shape. To achieve this we use the generalized 6×6 diffusion tensor of the particle, and follow the coordinates of the diffusion center as well as the components of the vectors that define a reference system of axes fixed in the particle, both expressed in terms of the laboratory fixed coordinate system.

The program is modular and the interaction between the external agents and the particle must be described by routines contained within this package or supplied by the user. One of the subroutines will be for the calculation of the force and torque acting on the particle; they will determine one of the components of the particle's motion: the deterministic one, associated to the external agent (the other corresponds to the stochastic Brownian displacements). The other subroutine will check whether there is penetration of obstacles by the particle. This routine will consider the particular shapes of the obstacles and the particle to determine if there is penetration at each step of the simulated trajectory.

In addition to the modules needed to build your own executable, we also provide a self-contained program that includes the two mentioned subroutines for the case when the field is an uniform (although eventually time-varying) electro-optic field and the obstacles are planar walls. This serves as an example of how to build the program for other programs, and the resulting executable is indeed usable for a variety of frequent and relevant cases.

A previous step to the utilization of **`BrownRig`** is the calculation of the generalized diffusion tensor and the coordinates of the diffusion center of the simulated particle. To accomplish this we use the programs **`HYDRO, HYDROPRO`** or any other of the programs of the **HYDRO** family (see the specific instructions of these programs). Therefore, the output of **HYDRO/HYDROPRO** is used as input of **BrownRig** .

As in any other molecular simulation, work in Brownian Dynamics has two aspects: (a) simulation itself, i.e., generation of the trajectory followed by the system, and (b) analysis of the trajectory. The first stage is the one in which our software can be particularly helpful, and we have concentrated our effort in the development of the simulation program, **SIMRIG** . Future releases of this package will contain a collection of utilities for data analysis. For now, we supply two versions of a simple program, **ANARIG**, which may serve as a template for your own analysis program.

2. Literature

The primary reference for **BrownRig** is the paper:

- M.X. Fernandes and J. Garcia de la Torre, "Brownian dynamics simulation of rigid particles of arbitrary shape in external fields" . *Biophys. J.* (2002) 83:3039-3048.

For the calculation of the full, 6x6 diffusion tensor you may use the technique of bead modeling implemented in the **HYDRO** computer program, and if you are modeling with atomic detail, you may use the **HYDROPRO** program. The references are:

- J. Garcia de la Torre, S. Navarro, M.C. Lopez Martinez, F.G. Diaz, J. Lopez Cascales. HYDRO. A computer software for the prediction of hydrodynamic properties of macromolecules. *Biophys. J.* 67, 530-531 (1994).

- J. Garcia de la Torre, M.L. Huertas and B. Carrasco, "Calculation of hydrodynamic properties of globular proteins from their atomic-level structure. *Biophys. J.* 78, 719-730 (2000).

3. Previous HYDROxxx (or similar) calculation

One of the main data for the Brownian dynamics simulation of the rigid particle is the 6x6 diffusion tensor of the particle. If you are considering some specially simple cases (for instance, if your particle is an ellipsoid, or a cylinder) you may be able to calculate this tensor using analytical expressions. But in most cases, your particle will not have such a simple shape, and you will need one of the programs of the **HYDRO** suite **(HYDRO, HYDROPRO, HYDROSUB, HYDROMIC, HYDROPIX,** etc**)** to determine the 6x6 diffusion tensor.

In the following example, we consider a rigid, straight rod, having N=21 beads, with a diameter of 24.5 Å and total length of 200 Å. The simple program **HYDRO** suffices for this purpose, and the two data files needed by this program are:

- The main data file, compulsory named **hydro.dat**

```
StraighRod21beads                      Title
straightrod                            filename for output files
straightrod-coord.txt                  Structural filename
293.                              Temperature, Kelvin
0.010                             Solvent viscosity
177080.                            Molecular weigth
0.55                             Specific volume of macromolecule
1.0                               Solution density
0                Number of values of H
0                Number of intervals for the distance distribution
0                Ntrials
1                  IDIF=1 (yes) for full diffusion tensors
*
```

Figure 1. Main input data file for HYDRO

The molecular weight, partial specific volume and solution density may be fictitious (remember that diffusion do not depend on these three properties). However, temperature and viscosity must be correctly supplied, and the option `IDIF=1` is needed to obtain the 6x6 diffusion tensor

- The structural file, in this case named **`straightrod-coord.txt`**, contains the coordinates and radii. The centers of the elements lie on the z axis:

```
1.000E-08                unit of length
21      number of beads
    0.0    0.0    0.0   12.5
    0.0    0.0   25.0   12.5
    0.0    0.0   50.0   12.5
   .........................
   .........................
    0.0    0.0  225.0   12.5
    0.0    0.0  250.0   12.5
    0.0    0.0  275.0   12.5
```

Figure 2. Structural coordinates file for HYDRO (for brevity, some lines have been trivially supressed.

Running HYDRO you get the output file, in this case **`straightrod.res`**, from which you will extract the results which are needed for the present purpose, contained in the following lines:

```
....
               Center of diffusion (x):  0.000E+00 cm
               Center of diffusion (y):  0.000E+00 cm
               Center of diffusion (z):  2.500E-06 cm

           Generalized (6x6) diffusion matrix:  (Dtt  Dtr)
                                                (Drt  Drr)


 2.507E-07 0.000E+00 0.000E+00    0.000E+00-1.114E-08 0.000E+00
 0.000E+00 2.507E-07 0.000E+00   -5.930E-09 0.000E+00 0.000E+00
 0.000E+00 0.000E+00 3.703E-07    0.000E+00 0.000E+00 0.000E+00


 0.000E+00-5.930E-09 0.000E+00    7.182E+04 0.000E+00 0.000E+00
-1.114E-08 0.000E+00 0.000E+00    0.000E+00 7.182E+04 0.000E+00
 0.000E+00 0.000E+00 0.000E+00    0.000E+00 0.000E+00 3.923E+06

....
```

Figure 3. Some lines from the HYDRO output, with information needed for the BROWNRIG calculation

This is all you need from **HYDRO** in the present example. Of course, you can make similar calculations for more complex examples: for instance, you can obtain the diffusion tensor for a molecule with atomic detail using **HYDROPRO**, or for a multi-subunit structure using **HYDROSUB**, etc.

It should be pointed out the hydrodynamic elements ("beads") used internally by the bead modeling programs of the HYDRO suite may or may not coincide with the force points or excluded elements used later to specify the interaction of the particle with the environment.

In the following sections we describe the essential features of the usage of the simulation program, **SIMRIG**, along some examples in which we illustrate the procedure with a simple case, in which we simulate the Brownian dynamics in an electric field or in a confined environment of a rigid, bent rod

4. General structure of the program. Input data and output files.

SIMRIG is available in two forms:

1) A module in which all the common aspects of the simulation (input, simulation engine, output) are already programmed. You have to provide, as indicated below, two additional modules that specify how the particle interacts with the environment.

2) As a full, executable program, intended for the specific case of particles interacting with uniform fields and planar walls, as described below

Now we describe some general aspects of the program, including its structure and the input and output files.

4. a. Input data files.

The main input file has a name that is entered by the user when the program is started (in some places in this document will be generally denoted **`maindata`** file). Additionally, there may be other input files, whose names are given within this main input file, as described below.

(A) Contents of `maindata`, *the main input data file* :

- **`title`** (C*30) (designation of the case to be simulated)
- **`trajfilename`** (C*30) (name of the file where the trajectory of the particle will be recorded).
- **`logfilename`** (C*30) (name of the file with intermediate results, program messages, etc)
- **`temp`** (R) (temperature, in Kelvin)
- **`ttraj`** (R) (total trajectory time, in seconds)
- **`deltat`** (R) (simulation time step, in seconds)
- **`nconstep`** (I) (number of consecutive steps in a block)
- **`nregs`** (I) (number of positions registered in trajectory file)
- **`ntraj`** (I)(number of independent trajectories). Note that you may (a) generate a single trajectory (**`ntraj`**=1), i.e., the trajectory of a single molecule or (b) generate some number of independent trajectories that will be individually initialized (see file **`simrig-ini.txt`**).
- **`iseed`** (I)(seed for random number generation)
- **`dc`** (i=1,2,3) (R) (particle diffusion center coordinates in cm) (three lines, one with each coordinate)
- **`d`** (k, l =1,...6) (R) (generalized diffusion tensor) This tensor is composed by 4 boxes with 3×3 dimension. The upper left box corresponds to the translational diffusion tensor (cm^2s^{-1}). The upper right and lower left boxes correspond to the coupled translational/rotational diffusion tensor (cm^1s^{-1}). The lower right box corresponds to the rotational diffusion coefficient tensor (s^{-1}). Given in six lines, with six columns. You can cut and paste the lines from the output given by **`HYDRO`** or **`HYDROPRO`**.
- **`initialfile`** (C*30) (name of file with information on the initial position an orientation) See description below
- **`elementfile`** (C*30) (name of file with information on the "elements" that you would eventually consider within the particle). See description below. This file is optional; if there are no such elements, give a "**-**" as the first and only character of this name
- **`paramfile`** (C*30) (name of file containing parameters that specify the interaction with the environment). See description below. This file is optional; if there are no such parameters, give a "**-**" as the first and only character of this name

After each variable name we have indicated its type between parenthesis: (C*nn), for character type of nn characters; (I) for integer type (you cannot use decimals); (R) for real type (you must put a decimal separation "."). All values of the input files are in free format. This means that any format, as long as you respect variable types, is valid and that values in the same line can be separated either by commas or by blanks. The C*nn files must contain the name followed by the spaces needed to complete the nn characters.

Some more details about **`nconstep`** and **`nregs`** seem pertinent. The total number of steps will be **`ntotal=ttraj/deltat`** and the number of steps between successive registers will be **`nfrecreg=ntotal/nregs`**. On the other hand, **`nconstep`** is the number of steps that are taken consecutively, i.e., without checks, recordings, etc. Thus **`nconstep`** must be equal or smaller – specifically, an integer divisor - of **`nfrecreg`** . When there are interactions other than forces and torques, and particularly when there are exclusion interactions with walls, obstacles, etc., the time corresponding to a block of **`nconstep`** , equal to **`nconsted*deltat`** must be small so that those interactions are not strongly violated; in such cases it may be advisable to set **`nconstep=1`** .

For **`nreg`**, number of conformations recorded along the trajectory, you will typically give a round number, e.g. 10 000. The program will include, in the trajectory file, one more (e.g. 10 001), which is the initial conformation of the trajectory, as specified by the user.

A typical file could be as follows:

```
Brownrig straight rod N=21                !Title
mytrajecfile.txt                          !Name of trajectory file
mylogfile.txt                             !Log file
293.,                             !Temperature, K
1.E-4,                            !total trajectory duration
1.E-11,                           !time step, s
10                                !number of consecutive steps
10000                             !number registers in trajectory
5,                              !number of trajectories
654322,                           !random number seed
0.000E+00                       !coordinates of center of diffussion
0.000E+00 cm
2.500E-06 cm
 2.507E-07  0.000E+00 0.000E+00    0.000E+00 5.930E-09 0.000E+00
 0.000E+00  2.507E-07 0.000E+00   -5.930E-09 0.000E+00 0.000E+00
 0.000E+00  0.000E+00 3.703E-07    0.000E+00 0.000E+00 0.000E+00
 0.000E+00 -5.930E-09 0.000E+00    7.182E+04 0.000E+00 0.000E+00
 5.930E-09  0.000E+00 0.000E+00    0.000E+00 7.182E+04 0.000E+00
 0.000E+00  0.000E+00 0.000E+00    0.000E+00 0.000E+00 3.923E+06
myinitialfile.txt
myelementfile-elec.txt
myparamfile-elec.txt
```

Figure 4. A typical **`maindata`** file for BROWNRIG.

IMPORTANT NOTE: BROWNRIG works **internally** with physical quantities expressed in the **c.g.s. system of units**.

(B) Purpose and contents of the compulsory file `initfile`

This file is intended to provide starting positions for the trajectories. It will specify the initial position and orientation of the particle, giving the initial situation of the origin and axes of the particle-fixed system of coordinates with respect to the origin and axes of the lab-fixed frame of reference.

As indicated in the description of main input file, there are several possibilities concerning how and how many trajectories are generated in a single run of program **`SIMRIG`**. This are:

a) If **`ntraj`** is 1, one single trajectory is generated
b) If **`ntraj`** is >1, **`ntraj`** trajectories are generated. For this case there are two possibilities:
   - (b.1) The second, third, and successive trajectories begin with the final conformation of the previous trajectory. Thus, this is like running a single "super-trajectory" having **`ntraj`** "sub-trajectories". This may be useful in some situations, perhaps for statistical purposes.
   - (b.2) Each trajectory must be started from a different (user supplied) initial position. This is like generating individual trajectories of each molecule in a sample of **`ntraj`** molecules.

The contents of this file, whose name will be necessarily **`initial.dat`** , is as follows:

- **`ulenght`** (R) unit, or token, or length, expressed in cm, used for the coordinates that follow are expressed; for instance, 1.e-8 if the initial coordinates are in Å.
- **`icase`** (I) flag to indicate: **`icase=1`**, when there is a single initialization as in the above mentioned cases (a) and (b.1); **`icase=2`**, when each trajectory is individually initialized. Then, we will have either **one** (if **`icase=1)`**, or **ntraj** , (if **`icase=2)`**, blocks of data, with the following information:

- **`r(k)`**, k = 1, 2, 3, (R) the initial coordinates of the particle's diffusion center, the three values in one line, and
- **`e(k,l)`**, (R) the components of the 3 unitary director vectors of the particle fixed coordinate system expressed in the laboratory fixed coordinate system. They will be placed in a 3x3 matrix, **`e(k,l)`**, with k and l = 1, 2, 3, containing the vectors in its columns. Thus the three lines will contain the three values of each row in the matrix

A simple choice would be as follows: the particle is placed at the origin (r(1) = r(2) = r(3) = 0) and the particle and laboratory reference systems coincide (e(3,3) is equal to the 3×3 identity matrix):

```
1,
0., 0., 0.,
1., 0., 0.,
0., 1., 0.,
0., 0., 1.,
```

Figure 5. A typical, and simple, **`initfile`** . Each successive trajectory will begin where the previous one ended.

Alternatively, you may consider special placements and/or initial orientations of the particle. For instance, in the following example of **`initfile`**, the diffusion center is placed at coordinates (20, 30, 40), and the unitary vectors of the x, y and z particle-fixed axes are initially (0.7746,-0.6324, 0), (0.6324, 0.7746, 0) and (0, 0, 1)

```
1    ¡icase
20. 30. 40. ¡coordinates of diffusion center
 0.7746  0.6324  0.0000  ¡first row of matrix
-0.6324  0.7746  0.0000  ¡second row of matrix
 0.0000  0.0000  1.0000  ¡third row of matrix
```

Figure 6. Another example of **`initfile`**.

These would be the initial conditions for the trajectory (if **`ntraj`**=1) or for all the trajectories (if **`ntraj`**>1). It is essential that the three unitary vectors must be ortonormal: the sum of the squares of the value in any row and in any column of the matrix should be unity.

In the more complex case **`icase=2`**, there would be **`ntraj`** blocks of four lines, each block with the initial conditions. Probably, you will have to write a separate computer program to generate the initial conditions by some given procedure (perhaps by a Monte Carlo method)

(C) Purpose and contents of the optional **`elementfile`** :

Although this is not strictly necessary, in some applications we will consider that the whole particle (or just some part of it) contains some “elements”. For instance, these may be point charges, if you are considering a charged particle in an electric field. The elements may also be used to describe excluded-volume interaction with walls, obstacles, etc; then the elements will be the points that cannot trespass such obstacles. Or, simply, if the original hydrodynamic model was a bead model, the elements may be the beads in that model. In general, elements are points that play some role during the simulation and/or the analysis.

Of course, you have to supply the coordinates of the elements in the particle-fixed system of axes (the same one in which the diffusion center and the diffusion matrix have been given in the previous **maindata** file). You can either give the coordinates within this file, or alternatively, you can give the name of an external file which contains a list of the coordinates with the PDB format. In such a case, make sure that the PDB file is the one that has been used to obtain the diffusion center and matrix (for instance, in a **HYDROPRO** calculation)

Optionally, the elements may have attributes. Thus, if the elements are point charges, then the value of the charges will be an attribute. Also, the elements representing an excluded-volume interaction may have a size (radius of the spherical element), which will be treated as another possible attribute. You can even assign integer attributes (used for indices or whatever), and character strings (e.g. names). In the present version, we consider up to six attributes, three given by real numbers, two given by integer numbers, and one given by a six-character string. The attributes given here may not be even used in this program; they could be intended for use by the trajectory-analysis programs, which could read also the **elementfile**.

So, this optional file will contain, first, the following data:

- **nelem** (I) (number of elements).
- **iflag** (I) (flag that controls next section of this file. Presently acceptable values are 0 and 1).
- If **iflag** =0: **ulength** (R) Unit of length (expressed in cm – centimeters) in which the element coordinates are given next
- If **iflag** =0 the next **nelem** lines will contain the three Cartesian coordinates (R), **xe(1)**, **xe(2)**, **xe(3)**, of each element, referred to the particle-fixed system of axes, which is centered at an arbitrary (or conveniently chosen) origin. Actually, the line may contain more than three values (for instance, if you cut and paste from the structural data of HYDRO, which in addition to bead coordinates contains also bead radius). The three coordinate must be the three first (real) values in the line, and the remaining ones will be ignored.
- If **iflag** =1: **coordsfile** (character*30) Name of the PDB-formatted file with the list of coordinates (the program will asume that they are given in Å and will take internally **ulength** =1.E-8).

and then that will be followed by:

- **nelematt** (I) (number of elements that posses at least one attribute). This will be followed by **nelem** lines containing
- **ind** (I) (index of the element); **attrib1** (R), **attrib2** (R) and **attrib3** (R), values of the attributes of the first three kinds, given by real numbers; **ittrib1** (I) and **ittrib2** (I), values of the attributes of the fourth and fifth kind, given by integer numbers; **cttrib** (character*6), name or code of the sixth alphanumerical attribute. For kinds of attributes that you do not use, fill the fields in this line with real zero (0.), integer (0), or six spaces.

We emphasize that, among the many possibilities of using elements and attributes we can mention the following (which will be later considered with some detail):

- **point charges** are located at the given coordinates.
- **hard spheres** are employed to represent excluded-volume interations with walls, other bodies, etc.

As indicated above, the particle has previously been represented as a bead model to calculate the hydrodynamic tensors. Then, the elements may be some of the beads, but not necessarily all the beads will be elements. Furthermore, elements may be points other than the bead centers.

Next we present two examples, pertaining to the simulation of the straight rod previously introduced as the main example in this document: one in which this file is used to assign charges to the end beads, and another, where the hard-sphere radius of the end-beads are declared. The use of these sample files will be described in detail later on:

```
21,   !number of elements
0,   !iflag=0 : free format
1.e-8  !unit of length (cm); following: coords...
     0.0    0.0    0.0   12.5
     0.0    0.0   25.0   12.5
    .. .. .. .. .. .. .. ..
    .. .. .. .. .. .. .. ..
     0.0    0.0  475.0   12.5
     0.0    0.0  500.0   12.5
2,   !number of elements with attributes (charges)
 1  +1.000000  0.  0. 0 0 end--1
21  -1.000000  0.  0. 0 0 end--2
```

Figure7. Example of **`elementfile`** , used later to assign charges to the end beads of the rod used as example

```
21,   !number of elements
0,   !iflag=0 : free format
1.e-8  !unit of length (cm); following: coords...
     0.0    0.0    0.0
     0.0    0.0   25.0
    .. .. .. .. .. .. .. ..
    .. .. .. .. .. .. .. ..
     0.0    0.0  475.0
     0.0    0.0  500.0
2,   !number of elements with attributes (sizes)
 1  0.0  12.5  0. 0 0 end--1
21  0.0  12.5  0. 0 0 end--2
```

Figure 8. Example of **`elementfile`** , used later to assign hard-sphere sizes (radii= to the end and central beads of the rod used as example

The option of using a PDB formatted file will be particularly helpful for the Brownian dynamics simulation of molecular structures, whose hydrodynamic tensors may be previously evaluated with HYDROPRO.

(D) Purpose and contents of the optional **`paramfile`** :

If the application considers any external agent, you will have to specify some parameters that specify its characteristics. In a most general cases, the parameters may vary with time, and we accept that they do in a stepwise form. The trajectory duration may be divided into several intervals. The **`paramfile`** will provide the values of the parameters in each interval, with the following structure:

- **`np`** (I) number of parameters .
- If **`np`** is not zero**`,`** **`nt`**(I) (number of time intervals), and this will be followed by **`nt`** lines, each of which will contain (everything in the same line) the following data
- **`timepar(i), param(i,j),j=1,…np`** (R) The time when the i-th interval ends, and the values of the parameters during this interval

The first interval begins at zero time, and the **`timepar(i)`** value, for the last interval, must coincide with the trajectory duration as entered the main data file**.**

```
3           !number of electric field parameters
4           !number of time intervals
0.50e-5 , 0. , 0. ,  0.0e+06, !off-field
3.00e-5 , 0. , 0. , -1.0e+06, !negative field on
5.50e-5 , 0. , 0. ,  1.0e+06, !positive field on
7.00e-5 , 0. , 0. ,  0.0e+06, !off-field decay
```

Figure 9. Example of **`paramfile`**

In this example, there are 3 parameters, and the trajectory is divided into 4 intervals, up to a total duration of trajectory of 70 microseconds. Two of the parameters are 0 all the time, and the third parameter takes, in each interval, the changing values specified in the file. The meaning of this example will be later described in Example 2.

Note that this scheme restrict the application to cases where the parameters vary in a stepwise fashion. Other time-dependencies of the parameters could be implemented in a more sophisticated use of the program: there can be no **`paramfile`** , and instead the user can program the time-dependence of the parameters within the user-supplied subroutines **`TOFORRIG`** and **`WALLRIG`**.

4. b. Ouput data file.

There are two output files, one containing the trajectory, and the other with information and messages concerning the program progress.

The **trajectory file**, with the title given by the user (**`title2`**) has the record of the trajectory (or more properly, of the successive independent trajectories if **`ntraj`** > 1). The first three lines contain:

-- **`ntraj`** : number of trayectories
-- **`nregs`** : number of registers in each trajectory
-- **`deltatreg`** : the time interval between succesive registers

Then there will be **`nreg*ntraj`** registers containing 12 numbers corresponding to each step, the first three give the coordinates of the diffusion center of the particle (in Å), and the remaining nine correspond to the components of the unitary vectors of the particle fixed axes. These components, arranged in a matrix E(k,l), define a transformation matrix that allows the transformation of any particle-fixed vector from the instantaneous particle system coordinates into the laboratory system coordinates. The order in which they appear is E(1,1), E(2,1), E(3,1), E(1,2),E(2,2)..., etc.

If the coordinates some point or vector in the particle-fixed system of coordinates are given in array **`P`**, then the coordinates in the laboratory system, array **`PLAB`**, can be obtained with the code

```
PLAB(1)=E(1)*P(1)+E(4)*P(2)+E(7)*P(3)
PLAB(2)=E(2)*P(1)+E(5)*P(2)+E(8)*P(3)
PLAB(3)=E(3)*P(1)+E(6)*P(2)+E(9)*P(3)
```

Figure 10. Lines of source code needed to transform coordinates in the particle-fixed axes into coordinates expressed in the lab axes.

The second output file is the log file, with the name given by the user, which contains messages produced by the program during its execution, that have been also displayed in the terminal.

5. Trajectory analysis.

Owing to the very general scope of the program and the great variety of problems to which it may be applied, the trajectory analysis will be very specific of the case under consideration. That is why we do not provide a general program for analysis. The information provided in the previous section, about the structure of the output file is sufficient to devise any analysis program. Nonetheless, we provide two examples of the analysis program, corresponding to the applications used later as examples, that also serve as templates for other analysis program, as they contain the code needed to read the trajectory file

Program **`anarig-1.for`** , included with this distribution, reads the trajectory file, and computes, for each recorded position, for each molecule or subtrajectory into which the sample was divided, the coordinates of the particle´s center of diffusion *(xc, yc, zc)* and that of a particle-fixed vector ***p*** $=(p_x, p_y, p_z)$, writing a separate file with the values of and z_c and *cos(?)* where *cos(?)* is the cosine of the angle subtended by vector ***p*** the z axis of the laboratory fixed system. This time evolution is registered in a separate file that can be used for graphics or other manipulations. The input data file **`anarig-1.dat.txt`** contains the following lines

* the name for the trajectory file to be analyzed (C*30)
* the name for the general output file (C*30)
* the name for the file with the time evolution of the center of mass and p (C*30)
* the three coordinates of unitary vector ***p*** in the particle-fixed axes (R)

The outcome of the program, in addition to the list of values of time, z_c, and *cos(?)* gives, for each molecule or subtrajectory, the following averages along the positions for that molecule or subtrajectory:

* The averages $< z_c^2>$ and $< cos^2$ *(?)* >
* The mean square displacement of the center of mass between two successive records, $<d^2>$

Finally, the program gives the minimum and maximum values of x_c, y_c, and z_c, reached during the whole Brownian trajectory

Program **`anarig-2.for`** is similar; the input data file **`anarig-2.dat.txt`** is the same as above, with the particularity that not just one, but several unitary ***p*** vectors can be given. This program is specifically intended for illustrating the simulation of time-dependent properties of a multi-molecule sample (simulation with multiple subtrajectories). For any time, the average < *P2[cos*^{2} *(?])* > is evaluated over the set of molecules or subtrajectories. As it will be described in Example 2, this function is related to the transient electric birefringence.

6. Modular distribution: object modules.

(A) Modules included with the distribution

The modular distribution consists of a main module covering the common aspects of the simulation, and the user will construct two modules that specify the interaction of the particle with the environment.

The modules contained in this distribution are as follows:

- **simrig_nn.xxx** . This module is composed by the main program and the subroutines that calculate the trajectory of the particle. **nn** is a version number The file supplied within this distribution should be useful in most cases, and cannot be modified by the user. With this distribution we supply (a) object file **simrig_nn.o7w** for MS DOS/Win g77 compiler (b) object file **simrig_nn.o7l** for Linux g77 compiler, valid for Linux RedHat 7.3 (and hopefully later versions of this O.S.) (c) object file **simrig_nn.obj** for MS DOS/Win Compaq Fortran compiler.

- **common_r1.for** is an include source file needed for compilation (***it must not be modified*** by the user). It has to be present in the directory where the compilation is being done. The following piece of this file shows the main variables, and gives some hints on the internal functioning of the program. It may also be consulted for preparation of user-written modules

```
      REAL time
      COMMON/CLOCK/time
      REAL r0,r,e0,e
      COMMON/INSTCONF/r0(3),r(3),e0(3,3),e(3,3)

      INTEGER nelem,nelemmax,nelematt
      PARAMETER (nelemmax=10000)
      REAL x(nelemmax,3)
      COMMON/elements/nelem,x

      INTEGER na,ni,nc
      PARAMETER(na=3,ni=2,nc=1)
      CHARACTER*6 cttrib(nelemmax,nc)
      INTEGER ind(nelemmax),ittrib(nelemmax,ni)
      REAL attrib(nelemmax,na),
      COMMON/attributes/nelematt,ind,attrib,ittrib,cttrib

      INTEGER ntim,ntimmax,npar,nparmax
      REAL timepar,param,paractual
      PARAMETER(ntimmax=1000,nparmax=20)
      COMMON/params/ntim,timepar(ntimmax),
     & npar,param(nparmax,ntimmax),paractual(nparmax)
```

Figure 11. Selected lines of the **common_r1.for** file

Array **x** contains the coordinates of the **nelem** elements. **nelematt** is the number of elements with attributes, **ind** is the index of each attribute-bearing element, **attrib** is an array containing the three real attributes of each element, and similarly **ittrib** and **cttrib** are arrays with the two integer and the two character attributes, respectively.

Regarding the parameter data, they are as explained in the description of the **paramfile,** The values to be used are contained in **paractual** which is the set of parameters during the present time steps .

And, finally,

- **`time`** is the elapsed time for the present trajectory
- **`r0`** is the instantaneous position of center of diffusion of the particle, with coordinates expressed in the lab system, previous to the time step.
- **`e0`** is a matrix containing in its columns the unitary vectors of the particle-fixed axis, expressed in the lab system. Any vector expressed in the lab system, when premultiplied by this matrix, gives the vector in the particle system. We can also say that it is a matrix containing in its rows the unitary vectors of the lab-fixed axis, expressed in the particle system. Any vector expressed in the particle system, when premultiplied by the transpose of this matrix, gives the vector in the lab system
- **`r`** and **`e`** will be these vector and matrix after the time step.

(B) Additional modules

The user must additionally supply two other files:

- A subroutine named **`TORFORRIG`** that calculates the deterministic force (in dyn) , `FQPT(K)`, k = 1, 2, 3, and torque (in dyn.cm, referred to center of diffusion), `TQPR(K)`, k = 1, 2, 3, acting on the particle due to its interaction with the external field or agent. The components of these vectors are computed in the particle-fixed coordinate system. A general scheme of this file could be

```
      SUBROUTINE TORFORRIG(FQPT,TQPT)
      IMPLICIT NONE
      INCLUDE 'common_r1.for'
      REAL TQPT(3),FQPT(3)
      ...
      FQPT(1)=...
      FQPT(2)=...
      FQPT(3)=...
      TQPT(1)=...
      TQPT(2)=...
      TQPT(3)=...
      RETURN
      END
```

Figure 12. Scheme of the **`TORFORRIG`** subroutine

- If an impenetrable obstacle is placed in its way, the particle must somehow avoid it. The user-supplied subroutine **`WALLRIG`** is invoked after the calculation of a block of **`nconstep`** Brownian step and has the function of validating each and every step of the trajectory. If in some step, the particle penetrates the obstacle the step is not valid (exclude =.true.) or if the particle does not penetrate the obstacle the step is valid (exclude =.false.).

```
       SUBROUTINE WALLRIG(exclude)
       IMPLICIT NONE
       INCLUDE 'common_r1.for'
       LOGICAL exclude
       ...
       exclude=.TRUE.
       ...or…
       exclude=.TRUE.
       RETURN
       END
```

Figure 13. Scheme of the **`WALLRIG`** subroutine

If there were no deterministic forces or torques, or if there are no walls, the user should provide “empty” subroutines with the same structure (calling arguments)

A more specific, but still quite general situation is when the particle’s force and torque are the resultants of point forces that apply at some of the “elements”, according to their “attributes”. Also, for the interaction with obstacles, the restriction may be placed at the elements, with cannot trespass the limits. In such cases, the individual elements must be considered, and these two files would look like:

```
      SUBROUTINE TORFORRIG(FQPT,TQPT)
      IMPLICIT NONE
      INCLUDE 'common_r1.for'
      INTEGER IQ,I,J
      REAL TQPT(3),FQPT(3),TQP(3),FQP(3),DRQL(3),RQL(3)
      REAL EFIL(3),FORCELAB(3)
C*** Set total force and torque to zero
      DO I=1,3
          FQPT(I)=0.
          TQPT(I)=0.
      ENDDO
c*** For all the elements with attributes, calculate their
c    coordinates in the lab frame
      DO IQ=1,nelematt
          DO I=1,3
              DRQL(I)=0.
              DO J=1,3
                  DRQL(I)=DRQL(I)+E0(I,J)*x(ind(IQ),J)
              ENDDO
              RQL(I)=R0(I)+DRQL(I)
          ENDDO

C*** call user provided subroutine that calculates forces at
c    each element, expressed in the lab frame, FORCELAB
         CALL YOUR_SUBROUTINE (....FORCELAB....)
C    And then transform the force to the particle frame
          DO I=1,3
              FQP(I)=0.
              DO J=1,3
                  FQP(I)=FQP(I)+E0(J,I)*FORCELAB(J)
              ENDDO
          ENDDO
C*** calculates torques in particle frame
          TQP(1)=x(ind(IQ),2)*FQP(3)-x(ind(IQ),3)*FQP(2)
          TQP(2)=x(ind(IQ),3)*FQP(1)-x(ind(IQ),1)*FQP(3)
          TQP(3)=x(ind(IQ),1)*FQP(2)-x(ind(IQ),2)*FQP(1)
C*** Add element forces and torques to the total one
            DO I=1,3
                FQPT(I)=FQPT(I)+FQP(I)
                TQPT(I)=TQPT(I)+TQP(I)
          ENDDO

      ENDDO

RETURN
END
```

Figure 14. Scheme of the **`TORFORRIG`** subroutine for forces acting on the particle's elements

and:

```
      SUBROUTINE WALLRIG(exclude)
      IMPLICIT NONE
      INCLUDE 'common_r1.for'
      LOGICAL exclude
      INTEGER i,k,l
      REAL value,radesf
      exclude=.FALSE.
      DO i=1,nelematt
          IF(... condition for element I...) THEN
              exclude=.TRUE.
              RETURN
          ENDIF
      ENDDO
      RETURN
      END
```

Figure 15. Scheme of the **WALLRIG** subroutine for excluded-volume interactions referred to the particle's elements

The particle's element coordinates **xe(k)** will be given as data, as mentioned above, either directly in this **elementfile** or separately in **coordsfile,** expressed in the particle-fixed system of axes that is centered at an arbitrary (or chosen for convenience) origin. But, for the internal calculations, the origin must be placed at the center of diffusion. Therefore, after data input the coordinates will be internally transformed to **xe(k)= xe(k)-dc(k)**. This observation is not really necessary for using the executable version of program (described below), but must be taken into account for custom-building the executable, with user-written subroutines **TOFORRIG** and/or **WALLRIG**.

7. Precompiled, executable program: uniform field and planar walls.

Now, we give an illustrative example of how to build a complete, executable program. The resulting executable is itself of wide applicability, covering a number of cases of interest. Therefore, the direct use of this executable, which is provided within the present distribution, will be also described in detail.

The specific problem considered in for this executable includes one or both or the following two particle-environment interaction:

- Charged particle interacting with an uniform but eventually time-dependent (step-wise varying) electric field, ***E(t)***
- Particle in the presence of planar walls that are perpendicular to the lab-fixed axes

In the first case, the components E_x, E_y, and E_z, are the parameters that -for each interval into which the trayectory is divided - must be given in the **`paramfile`**. The **`elementfile`** would contain the coordinates of the charged elements, and the values of the charges are stored in array **`attrib(k,1)`** (second index is 1). For this case, the subroutine **`TORFORRIG`** is in file **`tofori_xxx.for`** (xxx is a version number). This file is contained in the present distribution. Note that the scheme follows the general structure in Figure 12 and the specific scheme in Figure 14. Subroutine **`FIELD`** gives the values of the field parameters for the present instant, and subroutine **`ELECFORCE`** determines the force acting on the particle as the sum of the forces acting on each charged element (see the code of this file) .

Also subroutine **`WALLRIG`** considers the case of particles in between three pairs of walls, situated at $x = \pm x_{\max}$, $y = \pm y_{\max}$, $z = \pm z_{\max}$. In other words, the particle is enclosed in a box centered at the origin, with half-sides $x_{\max}$, $y_{\max}$, and $z_{\max}$, which are the three parameters (now time-independent) . The **`paramfile`** would contain, for a single interval that covers the whole trajectory, the dimensions of the box, expressed as the half-lengths, x_{max}, y_{max}, and z_{max}. The coordinate x_i of any bead, i, with hard-sphere radius r_i must satisfy $| x_i + r_i | < x_{max}$, and similarly for coordinates y_i and z_i . The **`elementfile`** would contain the radii of the elements stored in array **`attrib(k,2)`** (second index is 2). For this case, subroutine **`WALLRIG`** is in file **`walrig_xxx.for`** (xxx is a version number). This file is contained in the present distribution. Note that the scheme follows the general structure in Figure 13. Confinement within a slab or in a tube with rectangular section can be included by giving very large values (e.g. 1.e+30) to two or one of the three parameters x_{max}, y_{max}, and z_{max}.

The executable file produced by the compilation of the source codes in this example is included in the present version with the file name **`simrig_xxx-zzz.exe`**, where **`xxx`** is a version number and **`zzz`** is the platform where it runs (**`msd`** from MS Windows/DOS Compaq Fortran compiler, **`w77`** from MS Windows/DOS g77 compiler, and **`g77`** from Linux g77). Examples of the use of this executable will be presented later.

8. Examples

The following examples illustrate the use of the executable program for fields and/or walls. The particle is the straight rod with 21 beads whose **`HYDRO`** calculation was reported in Section 3 (Figures 1-3).

Example 1: Free diffusion

We first run a simulation of the free Brownian motion of the particle, in absence of external agents. As the parameter and element files are not needed, the main data file is as in Figure 4, but having "**`-`**" for **`elementfile`** and also for **`paramfile`** .
Then, **`mytrajecfile`** is analyzed with program **`anarig-1`**

From the evolution of the angle θ, subtended by the rod axis, and any lab fixed axis (z, for instance, we obtain $<\cos^2\theta>=0.34\pm0.02$, when the mathematical result for arbitrary orientation should be =1/3=0.333…. **`anarig_1`** also evaluates the mean square displacement of the particle between registers, with the result $<d2>=(1.758\pm0.013)10^{-14}$ cm^2. There are 10^4 registers in each trajectory of 10^{-4} seconds; therefore, the time between registers is $\delta t=10^{-8}$ s. According to the Einstein equation, for a particle with diffusion coefficient $D=2.906\ 10^{-7}$ cm^2/s (as obtained in the **HYDRO** calculation) the square displacement should be $<d2> = 6D\delta t = 1.744\ 10^{-14}$ cm^2 . In both calculations we obtain an excellent agreement between the simulation and the theoretical results.

Example 2: Transient electric birefringence

Here we simulate a sample of rods with a permanent dipole moment, interacting with a varying electric field. The field induces a preferential orientation in the rods, and the sample becomes anisotropic. The birefringence, or anisotropy of optical polarizabilty is proportional to the quantity

$$<P2(t)> = (3<\cos^2\theta(t)>-1)/2$$

where, θ is the angle subtended by the rod axis, and the lab-fixed axis. The simulation is carried out for a sample containing a sufficiently large number of particles. The average <…> is over all the particles, at a given time, t. The **`maindata`** file, is similar to that in Figure 4, with the following modifications:

```
. . . .
7.E-5,                          !total trajectory duration
2.E-11,                         !time step, s
10                              !number of consecutive steps
2000                            !number registers in trajectory
6000,                           !number of trajectories
. . . .
myinitialfile.txt
myelementfile.txt
myparamfile.txt
```

Figure 16. **`maindata`** file for the transient electric birefringence simulation.

The **`elementfile`** for this case is that that was used as example in Figure 8. Note that opposite charges +1, -1 (in electron units) , are placed at the ends of the rod, so that the dipole moment is 275 Debye. The **`paramfile`** is that in Figure 9. Note that there is no applied field in the first interval, which lasts for 5 μs. In the second one, with a duration of 25 μs, up to 30 μs of simulation, a field of $-1.0\text{x}10^6$ Newton/Coulomb is applied along the –Z direction. In the third interval, the field is reverted, now being along +Z for 25 μs, up to 55 μs of simulation, and in the fourth, final interval, the field is again switched off, and the field-free simulation proceeds for 25 μs, up to 70 μs

Using **`anarig-2`**, the time-dependent birefringence, expressed as <P2(t)> , is evaluated for any characteristic vector expressed in the particle fixed system of coordinates; for the case of the rod, this is (0,0,1), because the bead centers where placed along the particle's z axis. The results are displayed in Figure 17, which shows the build-up of birefringence when the field is switched on, its transient and recovery after field sign reversal, and the off-field decay. The latter is the stage more easily analizable; according to theory, for this case the decay is a single exponential, const.exp(- $6D_r^{(perp)}$ t), where $D_r^{(perp)} = 7.18 \times 10^4$ s^{-1} is the rotational diffusion coefficient for a perpendicular axis, obtained in the HYDRO calculation, as the double degenerated diagonal component of the rotational tensor; see Fig.4. The slope in a semilog plot should be $6D_r = 4.31 \times 10^5$ s^{-1} . Considering the initial, more accurate part of the semilog plot in Figure 17, we obtain an slope of 4.5×10^5 s^{-1}, in good agreement with the expectation.

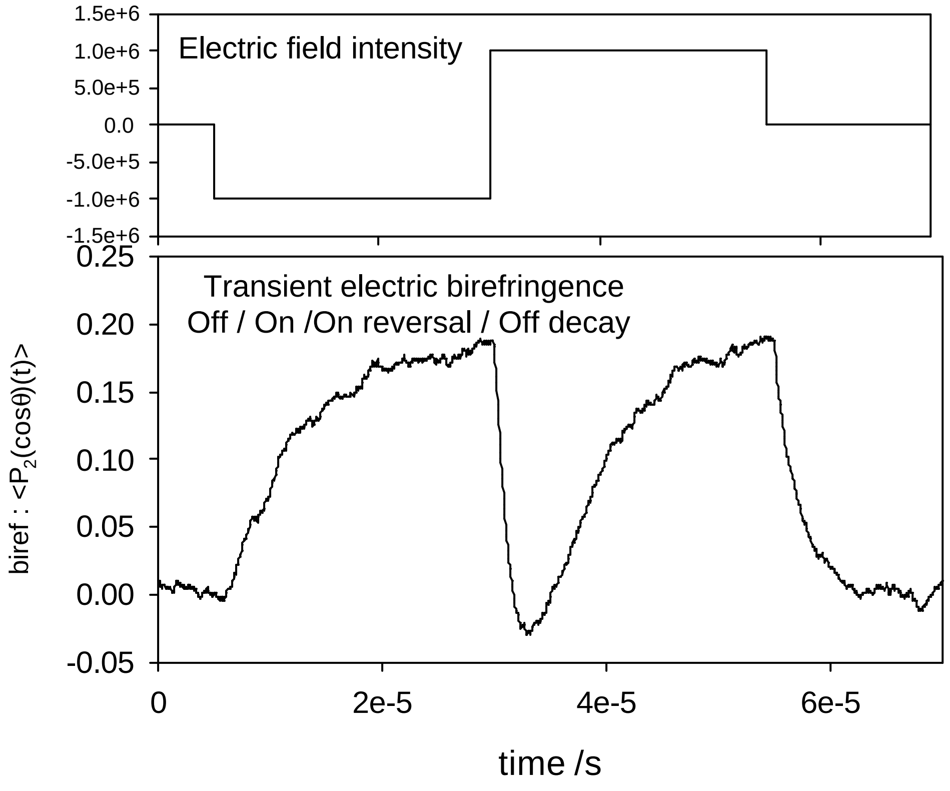

Figure 17. Transient electric birefringence. Arrows indicated when the field is applied, reverted and switched off

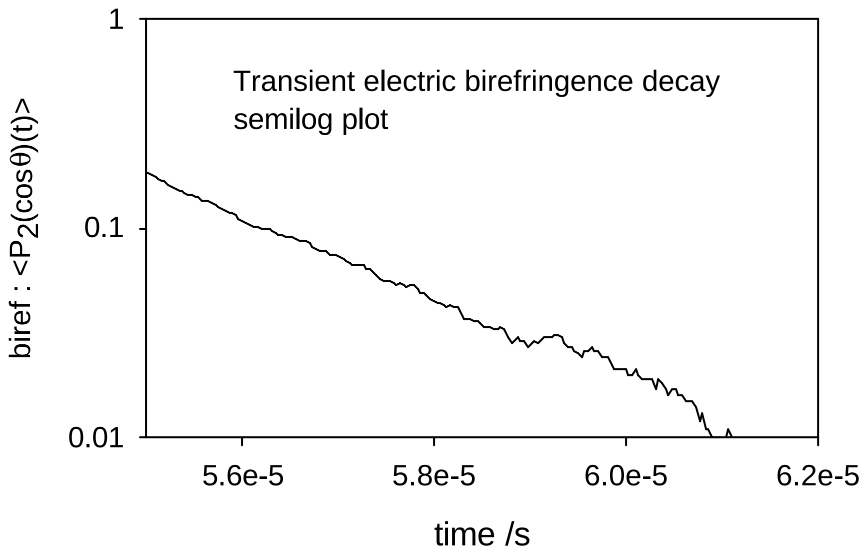

Figure 18. Off-field decay of electric birefringence.

Example 3: Simulation of a confined particle

We now simulate the Brownian motion of the rod within a box having the shape of a parallelepiped, with dimensions 2a, 2b and 2c, centered at the center of the lab system of coordinates. The simulation of an enclosed particle requires some caution in the simulation parameters. Thus, only one, rather long trajectory is simulated. The number of consecutive steps in a block of steps should be just 1, and a smaller time step is advised, both things in order to avoid that, after a block of steps, the particle had trespassed appreciably a wall.

```
. . .
2.E-2,                            !total trajectory duration
1.E-11,                           !time step, s
1                                 !number of consecutive steps
1000000                           !number registers in trajectory
1,                              !number of trajectories   .. .. ..
. . .
```

Figure 19. Somelines of the **`maindata`** file for the enclosed rod, that differ from the file displayed in Figure 4

In this example, the bead attributes to be specified in the **`elementfile`** are the bead radii. We do so for the two end beads; this surely suffices to detect collision of the whole rod with the wall. This file takes now the form as in Figure 8, with the final lines

```
    .. .. .. .. .. .. .. ..
2,   !number of elements with attributes (sizes)
 1  0. 12.5  0.  0  0 end-1
21  0. 12.5  0.  0  0 end–2
```

Figure 20. Final part of the **`elementfile`** for the enclosed rod. The bead sizes are placed as the second attribute, as indicated above.

Now the **`paramfile`** contains the positions of the wall. For a box of 6000 x 4000 x 2000 **Å,** centered at the origin, the negative and positive coordinates of each pair of opposite walls are the three parameters in this file:

```
3           !number of  parameters for walls
1           !number of time intervals
2.e-3 , 3.e-05 , 2.e-05 , 1.e-05  !ending time and params
```

Figure 21. The **`elementfile`** for a box of 6000 x 4000 x 2000 **Å**

As the box dimensions do not change with constant, the parameters are constant during the trajectory, and there is only one time interval covering the whole trajectory.

The trajectory is analized by means of program **`anarig-1`** finding the minimum and maximum value of the coordinates of the rod center. The results are;

max / min (**Å**) x: `-2985   +2986`
y: `-1987   +1985`
z: `-986    +987`
$<\cos^2 \theta> = 0.3091$

Thus the rod has been in contact with the six box walls. As the wall dimensions are much greater than the rod length, the enclosure has not induced any appreciable preferential orientation; the value of $<\cos^2 \theta>$ is quite close to that for absence of order, 0.333… However, if the one of the wall dimensions had been of the same order as the rod dimension, this would be noticed in the $<\cos^2 \theta>$ value.

9. Hints and notes.

Some words of guidance about the choice of the simulation time step, Δt**`.`** **`(delta`** in **`input.dat).`** As in other simulation methodologies, Δt should be as small as possible, so that the quantities (forces, diffusivities, etc) evaluated at the beginning of the step are have a small variation during the step. The translational and rotational displacements of the particle should small. These displacements can be estimated from the components of the diffusion matrix evaluated form the hydrodynamic calculation. The translational displacement is of the order of $(6D_t\Delta t)^{1/2}$ and the largest rotational displacement is of the order of $(4D_{r1}\Delta t)^{1/2}$, where D_t is the translational diffusion coefficient, and D_{r1} is the smallest (duplicate) values of the rotational diffusion tensor. In the example in **`bentrod30.dat`**, D_{r1} is about $7x10^4$ s^{-1}. Taking $\Delta t = 10^{-10}$ s, the largest rotational displacement is about 0.005 radian, i.e., about 0.3 degree, and the translational displacement is about 1.2 Å. As the simulation is appreciably fast, for further security we took an even smaller time step, $\Delta t = 10^{-11}$ s.

The number of steps in a trajectory will be **`ttraj/deltat`**. This number must be large, but not too much; say, at most 10^9. One trajectory with 10^9 steps takes about one hour of CPU in a modest PC. These steps are taken in blocks of **`nstepcon`** consecutive steps. After a block of steps, a record is written in the trajectory file. Thus, the number of records will be **`ttraj/(deltat*nstepcon)`**. This will obviously determine the size of the resulting trajectory file. If the total number of steps is 10^9 and **`nstepcon`** is 10^3, there will be 10^6 registers for one trajectory, which amounts to about 50 MB in disk space.

For simulations with hard walls, it may be necessary to reduce the time step, with the subsequent increase in computing time which, anyway, will be within reasonable limits.

In data files, lines containing filenames must be padded with spaces (to the right) up to fill the number specified number of characters. When you run the program in a Windows PC, we advise not to start clicking a Windows icon. You should better open the 'black' MSDOS and run the program entering the name of the executable file at the system prompt.

10. Release notes

Version 1.c is the second release of BrownRig. The first, preliminary version was 0.5 (release in 2002).

11. Distribution

Our web site is
`http://leonardo.fcu.um.es/macromol/`

The present distribution contains:

- Object modules for the main program and ancillary subroutines (**`simrig_1c2.obj`**, .**`o7w`** and .**`g7w`**) and include source file common_r1.for
- Sample of source codes for **`TOFORRIG`** and **`WALLRIG`** subroutines for the particular cases of electric fields and walls, and the corresponding object modules
- Executable programs for the particular cases of electric fields and walls, including sample data files
- Source code and executables for two examples of analysis programs
- Files needed to install a MSDOS/Windows Fortran g77 compiler

A file named **`contents.txt`** gives a detailed listing of all the files included within the distribution.

NOTE: Source code simrig_1c2.for, which could be needed for eventual modifications and re-compilations of the whole program, is not included in this distribution. **This source code may be available from the author (jgt@um.es) upon request.**

12. Compilers

(A) g77 for Linux is a public-domain Fortran compiler from the Free Software Foundation (GNU), available within many Linux Distribution or downloadable from the GNU web site. The compilation option that we used was **`-O2 -fno-automatic`**

(B) g77 is available also for MSDOS/Windows. You can download from `http://www.geocities.com/Athens/Olympus/5564.We` have included within our distribution the main files needed to install this compiler in your PC, and give brief installation instructions.

(C) We have also used the Compaq Visual Fortran Compiler for Windows, version 6.6a

GOOD LUCK WITH THE PROGRAM!