Brownmove Tutorials

Tihamér Geyer

Zentrum für Bioinformatik, Universität des Saarlandes, D–66041 Saarbrücken, Germany
tihamer.geyer@bioinformatik.uni-saarland.de

V1.4 — Oct. 7, 2010

Covering Brownmove V1.1 of Oct. 2010

The following use-case scenarios assume that you successfully built brownmove and that the tutorials reside in the folder "doc/Tutorials" in the brownmove folder. For these tutorial cd into the Tutorials folder ("cd doc/Tutorials").

Freely diffusing particles

The most simple scenario is the diffusion of a single particle in an unbounded simulation volume. Such a case is set in the scene definition "freeParticle.brownscene". Read through the setup file and the referenced protein definition "simpleProtein.browndef". Next, run the simulation by

```
$> ../../brownmove freeParticle.brownscene
```

You will see a lot of text scroll through the terminal window. Scroll back and examine the output. First, the general setup like timestep size, number of steps, or vdW parameters are read by the "General setup". Then, at the "protein" keyword, the "ProteinParser" kicks in and parses the protein definition and sets up a template molecule of the protein "Simple". After that, the main global parser continues. One "Simple" protein is placed in the simulation volume and then the actual simulation starts. After this setup-diagnostics, the particle positions at every tenth timestep are printed to the terminal by the analysis command "cat". Because of the detour of the output via the shell and cat, it arrives at about the same time as the dump of the final protein position directly from brownmove.

To sort the data output to the terminal, first edit the analysis line in the setup file by adding an output redirection into a file "simple.txt":

```
analysis    cat > simple.txt
```

Next, add a line with the keyword "noProteinDump" somewhere between the timestep and the protein definitions and rerun the simulation. Now the particle positions are sent to the file "simple.txt" and the dump is gone. This trajectory output consists of lines starting with "timeline", followed by the simulation time and the number of rigid particles in the simulation volume (= Gestalt objects, here only one), and the respective number of positions per rigid particle. Each of these lines lists the protein's label, the numerical id, the label of the Gestalt object, and its position and orientation. The orientation is given with four numbers. The first gives the rotation angle (in radians) and the other three represent a unit vector defining the rotation axis.

Such a trajectory file contains all relevant information but is usually not directly useable for analysis. To extract the trajectory that the simple particle takes, you can use the supplied awk script "extractSimpleCM.awk" to extract the position of the center of mass of the protein:

```
$> gawk -f extractSimpleCM.awk simple.txt > simpleCM_BD.txt
```

This script reads to second field from the lines starting with "timeline" and appends fields four to six plus a newline from the lines starting with the protein label "Simple". You can now plot this file with, e.g., gnuplot to see the random Brownian motion of the particle.

Next, we will compare the Brownian and the Langevin propagation schemes. For this, make a copy of "freeParticle.brownscene" and set the outsteps in this copy to 1, i.e., have one output line at each timestep. The extraction of the CM performed after the simulation above will now be done on the fly. For this change the analysis line to

```
analysis    gawk -f extractSimpleCM.awk > simpleCM_BD.txt
```

and rerun the simulation. Plot the trajectory in the preprocessed output file "simpleCM_BD.txt".

The LD propagation is enabled by giving the protein a mass. Open "simpleProtein.browndef" and edit the "D0" line in the Gestalt to read (note the 10.0 on the fourth position. This is the protein's mass in kDa)

```
D0   1e-4   0.0   10.0   1.5
```

In "freeParticle.brownscene" change the output file from "simpleCM_BD.txt" to "simpleCM_LD.txt" and run the simulation. Use, e.g., gnuplot to compare the two trajectories. Of course, they are different.

To get comparable trajectories, we have to seed the random generator with the same value for both simulations. Add the line

```
randomseed 2009
```

to the setup file "freeParticle.brownscene". Additionally, reduce the timestep to 0.1 ps, set the number of steps to 10000, and repeat the two simulations (with mass = 10 kDa for LD and mass = 0 for BD). Now the plot should show you two trajectories with similar trends. One is more zaggy and the other more smooth and slightly delayed. To better see the differences of the finite damping with LD, zoom in on, e.g., the first 100 ps.

The smoothness of the LD trajectory is reflected in a reduced short time diffusion coefficient D_0. To evaluate this, you can use the supplied C program file "D0.c", which is compiled straightforward as

```
$> gcc -o D0 D0.c
```

It reads the time and three coordinates from standard input and determines the average squared displacements per coordinate and their average (have a look at the source code "D0.c").To determine D_0 from the BD simulation, which serves as a reference, feed the above generated output file "simpleCM_BD.txt" to D0

```
$> ./D0 < simpleCM_BD.txt
dt = 1.0000    D0 = 1.0031e-04   Di = 1.0631e-04, 9.7116e-05, 9.7503e-05
```

You'll get the timestep (check whether it is the one that you expect), the 3D diffusion coefficient, and the three 1D diffusion coefficients in x, y, and z direction. Apart from the statistical uncertainties, the diffusion coefficient specified in the setup file is reproduced. Next, do the same analysis with the LD propagation:

```
$> ./D0 < simpleCM_LD.txt
dt = 1.0000    D0 = 2.3973e-05   Di = 2.5339e-05, 2.3551e-05, 2.3029e-05
$> ./D0 < simpleCM_LD_0.1.txt
dt = 0.1000    D0 = 2.2181e-06   Di = 2.1037e-06, 2.2863e-06, 2.2642e-06
```

With the LD propagation the diffusion coefficient at these short times is smaller by a factor of about four for a timestep of 1 ps and by another order of magnitude for $\Delta t = 0.1$ ps. To get an impression of how D_0 depends on the observation times, further increase (and decrease) the timestep for the LD simulation and determine D_0 for different integration (observation) intervals. At which Δt is the given long time D_0 from BD reproduced? Instead of increasing the integration timestep, you can also increase the length of the output interval. This has the same effect, but is less efficient for this one particle scenario.

Next, we'll investigate a "many" particle scenario :-) Copy the file "simpleProtein.browndef" and call it "simpleProteinMinus.browndef". In this file, rename the Protein to SimpleMinus and change the sign of the charge in the EstatShape to –1.75. Make another copy of the original scene definition file and add the new protein below the other:

```
protein   Simple          simpleProtein.browndef      1
protein   SimpleMinus     simpleProteinMinus.browndef 1
```

Alternatively, you can define both proteins within one setup file (say "simpleProteinPlusMinus.browndef"). Then, you would define them via

```
protein   Simple          simpleProteinPlusMinus.browndef     1
protein   SimpleMinus     simpleProteinPlusMinus.browndef     2
```

Also place a copy of the new protein into the simulation. For this, displace the original protein away from the center:

```
startWith      Simple          3.0 -2.0 1.0   0.0 1.0 0.0 0.0
startWith      SimpleMinus    -3.0 2.0 -1.0   0.0 1.0 0.0 0.0
```

To fill up the volume, we add (quasi) periodic boundary conditions. Add the following line after the protein definitions and before the "startWith" lines:

```
periodicBox    12.0 12.0 12.0
```

Finally, increase the simulation length to 100000 steps with an output every 50 steps and redirect the output to the file "periodic.txt" (on the analysis line), run the simulation, and examine the output. Use the above gawk script to extract the CM of the positively charged "Simple" protein. It's CM should not go beyond ±6 nm on all three coordinate axes.

Starting from this setup, you can now increase the size of the simulation box, add more proteins of both types, and then, e.g., determine the density dependent diffusion coefficients or radial density distributions from the output. This is left as an exercise to the reader :-)

A Bead-Spring Polymer

The following part is based on the simulations reported in [T. Geyer and U. Winter, *JCP* **130** (2009) 114905, "An $O(N^2)$ approximation for hydrodynamic interactions in Brownian dynamics simulations"], where the fast approximation to the hydrodynamic interactions that is used by brownmove was introduced.

First, have a look at the file "polymer_n5.browndef", which defines a polymer of five uncharged van-der-Waals spheres connected by harmonic springs. In this file you find a line starting with "constant". The next token is the constant's name and the rest of the line (or up to a comment sign '#') is its value. Whenever the

parser encounters a dollar sign followed by the variable name, this is replaced by the constants content. With such a constant we can later change all masses of the polymer beads at once. Actually, the constant declaration could reside in the scene definition, too.

There is also a minimal scene definition file "polymer.brownscene". Run the simulation, while saving the final polymer configuration in the dump file "poly_dump.browndef":

```
$> ../../brownmove polymer.brownscene > poly_dump.browndef
```

Examine the output files. With the polymer dump, the simulation can be continued from where it was stopped in the first run. To do so, replace the polymer definition "polymer_n5.browndef" with the just generated "poly_dump.browndef", pipe the output into "poly_2.txt", and restart the simulation, now dumping into "poly_dump_2.txt". Compare the last part of the first output file "poly.txt" with the begining of the second output file "poly_2.txt". In real-life situations, you would define the parameter "offsetsteps" in the setup definition file.

To extract some useful information from the polymer simulations, extend their length to one million timesteps. From this large amount of data we only need the positions of the center of mass of the complete polymer and the vector pointing from the first to the last bead. This information can be extracted on the fly with the perl script "PolymerCompactor.pl":

```
analysis    perl PolymerCompactor.pl > polyCompact.txt
```

Run the simulation (this may take a few minutes) and examine the output (plot the data columns vs. time). When plotting the second, third, or fourth column (x, y, and z positions of the CM) vs. the first column (time) you will recognize the typical diffusive trajectories.

Cut the first four columns from the compacted output file and determine the CM diffusion coefficient. Without HI the CM diffusion coefficient scales as N^{-1}. Consequently, it should be one fifth of the diffusion coefficient of a single bead:

```
$> cut -f1-4 polyCompact.txt | ./D0
dt = 0.010000    D0 = 2.0015e-01   Di = 1.9999e-01, 2.0032e-01, 2.0014e-01
```

To verify the scaling, make a copy of "polymer_n5.txt" and extend the polymer to have 15 beads. For this, you need a total of 14 bonds. Take care to place each next bead another 2.5 nm away from the previous one. Adapt "polymer.brownscene" to read in the new longer polymer and also change the output file and the dump file to not overwrite your previous precious simulation results. Do you get the predicted one fifteenth of the single bead diffusion coefficient?

Now add hydrodynamic interactions. Open the configuration file for the shorter polymer, add the following lines to each of the five Gestalts and save as "polymer_n5_HI.browndef":

```
HiShape HiS
   1.0
End
```

It does not matter that you use the same label for the same shape within different Gestalt objects. Adapt "polymer.brownscene" (protein definition and output filename), rerun the simulation (check the dump file), and determine the CM diffusion coefficient. Then repeat with the longer 15 bead polymer. Together with the diffusion coefficient for a one-bead polymer (a single bead), you should get an approximate $N^{-0.58}$ scaling with HI.

Starting here, you can now reproduce all the data and analysis of the above referenced publication.

Particles above a Membrane

The following simulations are loosely based on on [T. Geyer, C. Gorba and V. Helms, *JCP* **120** (2004) 4573 "Interfacing Brownian dynamics simulations"] and [C. Gorba, T. Geyer and V. Helms, *JCP* **121** (2004) 457 "Brownian dynamics simulations of simplified cytochrome *c* molecules in the presence of a charged membrane"]. The setup consists of a rectangular simulation box that is bounded with a membrane on one side and a constant density reservoir on the opposite side. For the other four sides reflecting walls are used. The relevant setup files are "membrane.brownscene", "boxWithExchange.browndef", and "singleBead.browndef".

As you can see, the simulation output is piped into the perl script "histogram_x_gnuplot.pl", which creates a histogram of the *x* positions of the particles and then uses gnuplot for an online display of the current density and of the average density since the begining of the simulation. The densities are not normalized with the *y* and *z* lengths of the simulation box, the constant value of 0.64 nm^{-1} corresponds to the the externally set reservoir density of 4 x 10^{-4} nm^{-3} multiplied by the membrane area of 40 x 40 nm^2. The particles are inserted at $x = 20$ nm and subsequently diffuse to the left until they hit the wall at $x = -20$ nm. When the simulation runs long enough so that this initial filling process of the box is negligibly short, the time averaged density above the bottom wall converges to the externally set value of 0.64 nm^{-1}.

In the setup file "membrane.brownscene" an additional output file "aboveMembrane.txt" is defined, which contains the particle density in the acceptor "PunktRaus" and the number of particles of type "Punkt" in the simulation volume. PunktRaus has an inverse volume of 1 nm^3, i.e., the density corresponds to the number of particles that have left the simulation through the top wall up to that point in time.

Next, we add something like a membrane protein to the bottom wall. Add the following line to the VdwShape of the bottom wall

```
0.0 0.0 0.0   8.0   0
```

and repeat the simulation. Towards the end of the simulation you can clearly see the dip in the density due to the excluded volume from the "membrane protein". To have a closer look at the particle distribution closer to the membrane, redirect the simulation output into a file,

```
analysis  cat > punktsAbove.txt
```

run the simulation again, and then extract all particle positions close to the membrane:

```
$> gawk -v XMIN=-20 -v XMAX=-17 -f slice.awk punktsAbove.txt > punktDens.txt
```

Plot this file and observe the "hole" around the origin, which is a negative image of the membrane protein. A slice for, e.g., $-3 < x < 3$ does not show such a hole. If you want to both watch the density evolve and save the output, use

```
analysis tee punktsAbove.txt | perl histogram_x_gnuplot.pl -25 22 94 1000 | gnuplot
```

To check whether fluxes are handled correctly, replace the Gestalt of the bottom wall by an acceptor.

```
Acceptor UntenRaus
   # Mask  Center     extension (+-ly +-lz)
   Mask  0.0  0.0   20.0  20.0
   # molecule  Name  Reservoir
   Molecule    Punkt PunktUnten
End
```

Leaving away the corresponding injector means that we enforce a zero-density boundary condition at the bottom and a constant finite density at the upper wall. This acceptor has to be defined in "membrane.brownscene", too:

```
reservoir   PunktUnten   0.0   1.0
```

Also add an output definition for the particles collected by this acceptor:

```
density   PunktUnten
```

Run the simulation and verify that a linear density profile develops between the two interfaces. In this quasi-1D setup we should get a diffusion current of $D_0 \Delta\rho / L = 10^{-9}$ ps^{-1} nm^{-2}, i.e., a flux of 1.6 x 10^{-6} particles per picosecond through the 40 x 40 nm^2 cross section of the simulation box. Plotting the particle number in PunktUnten vs. time reproduces this analytical result with the expected large statistical uncertainties. Actually, the effective cross section between the side walls with their vdW repulsion is smaller, resulting in a slightly smaller current. For a better result, one should use 2D periodic boundary conditions.

What happens when you reduce the size of the acceptor mask to, e.g., 20 x 20 nm^2?

Further steps to explore this setup could be to replace the four side walls by 2D periodic walls, add point charges to the bottom wall (a charged lipid membrane) or to the membrane protein, or try different particle types.

Diffusion between fixed Obstacles

In a cell there are many obstacles to free diffusion like the cytosceleton or membrane bounded compartments. These fixed obstacles can be modelled by a wall with a gestalt (a wall with a gestalt has no implicit interactions anymore). Have look at the scene definition file "boxWithObstacles.browndef". It contains the same top and bottom walls with their injector and acceptors as the previous example plus another wall at $x = 0$ with nine large vdw spheres. Here, the spheres are in the wall plane, but they could be placed anywhere in the simulation volume. The sides are now bounded by periodic walls defined in "obstructed.brownscene". When you run this simulation, you will notice that the x-histogram shows a density step around $x = 0$.

From the saved output you visualize the obstacles by running

```
$> gawk -v XMIN=-3 -v XMAX=3 -f slice.awk punktsBetween.txt > punktDens.txt
```

followed by plotting punktDens.txt. Now check the particle count vs. time in the file "passedThrough.txt" and try to fit a straight line through the number. The onset gives you an effective diffusion time between the top injector and the bottom acceptor and the slope the diffusion current through the holes between the obstacles. How do the current of particles absorbed at the bottom and the diffusion time scale with the size of the obstacles?

External Forces

External forces can, e.g., be used to artificially lengthen a polymer chain, to model shear flow conditions, or to confine particles in a certain region in space. Here, we'll investigate how a shear flow induces particle rotation. Have a look at the protein definition "externalBead.browndef". It is a variant of the "simpleBead" with roation enabled and an additional ExternalShape, which defines a number of hook-up points which are sensitive to one of three external force fields.

These three fields are defined in "externalForce.brownscene". The first, "Drift", is a shear field parallel to the *y*-*z*-plane (normal pointing in *x*-direction) with the force along e_y. For this field, seven hook-up points are defined in the protein, six on the surface and one in the protein center with a weight to counterbalance the total force on the four peripheral hooks. This arrangment of hooks effectively induces a torque in the particle because the hooks that are further away from the *y*-*z*-plane encounter more force than thos closer to the plane.

With these seven hooks in the shear potential, the particle rotates around the z-axis. This rotation, however, is very hard to observe because it is overlaid by the stochastic fluctuations. Consequently, the particle tumbles around with an underlying tendency to rotate. The deterministic rotation can be seen better when the particle's orientation is confined. For this, the constant external force is used here. It keeps the two hooks at z = ±2 aligned along the z-axis.

Now, run the simulation and observe that the CM of the particle is in fact not moving away (apart from a random diffusion), while the orientation constantly changes. For this, plot columns 5, 6, and 7 of the pre-sorted output "shear.txt". The x and y componenets of the orientation vector fluctuate around zero, while the z component (column 7) jumps between +1 and –1, i.e, the orientation vector is aligned with the z axis. A plot of column 4 shows a pattern with rising and falling phases, while the product of columns 4 and 7 finally verifies that the particle is rotating in one direction.