Extended Tutorial

Charge distribution around a charged rod: Comparison of Poisson-Boltzmann Theory and computer simulations

Olaf Lenz *and Mehmet Süzen †

July 12, 2007
SimBio group, FIAS, Frankfurt

Contents

1 Introduction

In this extended tutorial, you are to apply the different methods that you have learned throughout the lecture and the previous tutorials to a simple physical system. The tutorial bases on an article by Deserno et al. [1]. Throughout the tutorial, you will learn how to reproduce the plots that are used in the article. As further reading, you can refer to [2], which is probably a bit more comprehensive.

*olenz@fias.uni-frankfurt.de
†suzen@fias.uni-frankfurt.de

Task:
As a homework to the second tutorial, read the article and try to understand it.

The system under consideration is the so-called *cell model* of a polyelectrolyte, i.e. a polymer that dissociates charges in solution (see in the lecture). In the cell model, a polyelectrolyte is modelled as a single, charged, infinite rod with its counterions and maybe some additional salt that is confined to a cylindrical cell. The observable of interest is the distribution of ions $P(r)$ around the rod.

To obtain the charge distribution, we will introduce two methods that can be used to tackle the problem. The first method is the Poisson-Boltzmann theory, an analytical mean-field theory, the second method are computer simulations. We will learn about the strength and weaknesses of both methods.

The cell model is defined by the following parameters:

Bjerrum length l_B The Bjerrum length is the distance at which the Coulomb energy of two elementary charges equals the thermal energy k_BT. Here, it defines the length scale of the simulation: all length are measured in units of the Bjerrum length. In water, the Bjerrum length is 7.1Å.

Line charge density λ The line charge density of the rod is the number of charges per length unit. It is closely coupled to the *Manning parameter* $\xi = \frac{\lambda l_B}{e_0}$.

Rod radius r_0 The radius of the charged rod defines the minimal distance between an ion center and the rod center.

Cell radius R The radius of the cylindrical cell defines the maximal distance between an ion center and the rod center.

Valency of the counterions v The valency of the counter ions.

The default values that we are going to use are:

$$\begin{aligned} l_B &= 1.0 \\ \lambda &= 1.0/2.0 \\ r_0 &= 1.0 \\ R &= 28.2 \\ v &= 1 \end{aligned}$$

Note that we define two values for λ. Make sure that different people in the tutorial use different values for λ! Compare your results. When you have read the article, you will be able to understand the differences.

Figure 1 on the following page shows a plot of the analytical solution of the Poisson-Boltzmann equation for the default parameters. Note that the x-axis of the plot of the distribution $P(r)$ is usually in logarithmic scale to stress the structure close to the rod. To create such a plot, you can use

```
xmgrace -log x data.dat
```

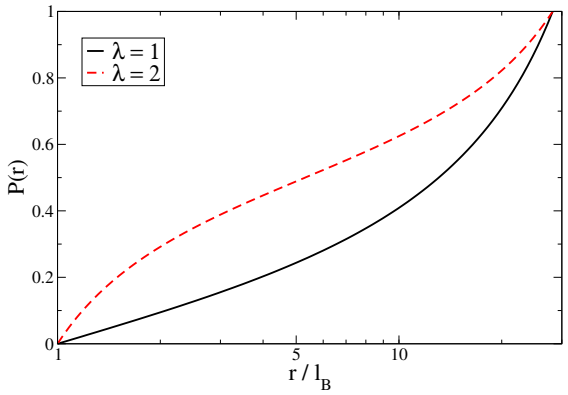

Figure 1: Poisson-Boltzmann solution for the charge distribution $P(r)$ over radius r for the default parameters.

2 Poisson-Boltzmann theory

On the one hand, the cell model for infinite charged rods can be solved within the *nonlinear Poisson-Boltzmann theory* (PB).

2.1 Analytical solution for the salt-free case

For the salt-free case, an analytical solution of the charge distribution $P(r)$ exists. It is given by equations (8) and (9) in the article. The equation contains two free parameters γ and R_M, that are defined by the equations (6) and (7) in the article.

Task:
Recreate the plot in figure 1. To complete this task, you can use whatever programming language or program that suits you best. We suggest using `Matlab`*. To do that, numerically solve equations (6) and (7) to obtain values for R_M and γ. Substitute the values in equations (8) and (9) to obtain a solution for the distribution. Create a file that contains the solution $P(r)$ over r that can be used for plotting.*
Keep the program that you have written - you will need it later on to obtain solutions for other parameters.

Task:
Compare your results with the results from the other students that have used the other value of λ. When you have read the article above, you should be able to understand the differences.

2.2 Monte-Carlo solution

Furthermore, a Monte-Carlo solver for the Poisson-Boltzmann theory in the cylindrical cell model can be found in the subdirectory `MC/` of the sources (`MCPB.c`). You do not need to understand how the code works. Near the beginning of the program, you can

find a block where the parameters of the model can be changed. When you compile the program (using `make`) and run it, it will write the integrated charge distribution to the file `MC.dat`.

Task:
Recreate the plot in figure 1 on the preceding page using the Monte-Carlo-solver. Test that it matches the plot from the last task!

3 Computer simulations

Furthermore, the charge distribution can be obtained via computer simulations.

3.1 Mapping the cell model onto an ESPResSo computer simulation

Of course, it is not possible to simulate the full cell model, as it requires an infinite rod. However, we can simulate a quasi-infinite system by exploitation of periodic boundary conditions: we create a rod that spans the whole simulation box size and use periodic boundary conditions in that direction.

ESPResSo does not allow for cylindrical particles, therefore we will model the rod by a number of fixed charged particles on a line parallel to the z-axis in the center of the simulation box.

Furthermore, we would like to be able to use the fast P3M method for computing the electrostatics. Therefore, our system has to be cubic (i.e. $L = L_x = L_y = L_z$) and it has to employ periodic boundary conditions in all three dimensions.

How can we map the cylindrical cell with a radius R onto a cubic simulation box with a box length L while still retaining the correct charge distribution? The trick is to use the same *ion density* in both systems. When the total ion density is the same in the cell model and in the simulation, we expect them to show the same charge distribution.

Note, that the box length L defines the length of the simulated segment of the rod, and consequently the charge of this segment. As the whole system needs to be charge neutral, this alos predefines the number of counterions in the system!

Task:
Map the default cell model parameters onto a cubic simulation box, i.e. compute L for the given value of R. How many ions need to be simulated?
In the sources, you will find the ESPResSo-script `sim.tcl`. This scripts sets up a system in the described way, but for a different set of parameters. Adapt the script to the given default parameters.
This script will be the basis for the solution of all following tasks. Therefore, it is not wasted time if you try to understand how the script works!

3.2 First runs

Now the script is prepared for the first simulations. A version of ESPResSo that can be used to run the script is installed in `/home/simbio/tutorial/t11/espresso/`. You can run the script via

```
/home/simbio/tutorial/t11/espresso/bin/Espresso sim.tcl [CHECKPOINT]
```

If you omit `CHECKPOINT`, the system will be set up randomly, otherwise, the script will start from the given checkpoint. At the end of each run, the script will write such a *checkpoint* `checkpoint_`*time*`.chk`. This can be used to continue a simulation from the point where it left off before.

The `timestep` is the time step in the Verlet algorithm, measured in simulation units. It does not make sense to measure observables after each timestep. Instead, observables are measured only after a fixed number of timesteps `steps_per_frame` has passed (which is referred to as a *simulation frame*). The number of frames that are done in a single simulation run is defined by `max_frames`.

The script will create two measurement files and a checkpoint file:

`energy.dat` contains the total energy and its components (kinetic, coulomb and LJ energies) over the time.

`dist.dat` will contain the average integrated charge distribution $P(r)$ over r. The distribution will only be measured, when the parameter `measure_distribution` is set to 1 in the script.

`checkpoint_`*xxx*`.chk` is a checkpoint file that contains all the data required to restart the simulation from the point where the simulation ended. *xxx* is the simulation time that has passed.

Task:
Run the script `sim.tcl` and plot the energies over time. Do you notice anything strange? Look at the different components of the energy. Which component behaves strange? What is the reason of this problem? Modify the script such that the problem disappears.
Note that the energies are usually of the order of 1000.

3.3 Equilibration and sampling time

Now we can start with the real simulations. First, we need to make sure that the systems is equilibrated, and we need to get an idea how many simulation frames we will need to sample to get good statistics.

To do that, you should monitor the slowest observable that you can find. In general, the energies and the different energy components are a good starting point.

During equilibration, you will notice that the observable has a trend, i.e. it grows or drops. Only after you can not observe any trend anymore, the system can be assumed to be equilibrated. Usually, the values themselves fluctuate very strongly, so that the trend might well hide within the fluctuations. Therefore, it is useful to create *running averages* of the observables over a few hundered frames that average out the fluctuations. `xmgrace` can create running averages (choose `Data - Transformations - Running Averages...` from the menu).

Now you need to find out for how many frames you need to sample your simulation. To get useful statistics, the sample should encompass at least several of the slowest fluctuations.

Task:
Run the simulation. Monitor the energies. Increase the number of frames done in the script (`max_frames`) and rerun the simulation from the last checkpoint several times if necessary. When you think you do not see a trend anymore, let the simulation run again for at least the number of steps performed so far as a safety margin.
Now analyze the fluctuations. How many timesteps do the slowest fluctuations span? If you have a number, multiply it by 2 and you have the number of frames that your sample should minimally encompass.

3.4 Measuring the charge distribution

Finally, the charge distribution can be measured. In the script, you need to set `measure_distribution` to 1 and set the number of frames to be performed to the number of frames that you got out above.

Task:
Measure the charge distribution around the rod. Take care that you restart the simulation from an equilibrated checkpoint!
Plot the achieved charge distribution and compare it to the distribution obtained from the both Poisson-Boltzmann solutions. Do they match? They should!

4 Divalent counterions

Now that we know how to perform a simulation and to compute the PB solution, we can vary the parameters of the model. The first thing that we want to vary is the valency of the counterions.

Task:
Compute the charge densities (analytical, MC and simulation) for a valency of the counterions of 2 (i.e. every counterion carries two elementary charges). Again, different participants of the tutorial should do this for two different values of λ.
Compare the results from PB theory with the simulation results. Do they match? When you have read the article, you should be able to understand why there are differences!
The simulations might take quite sime time. Start with the next task while the simulations run!

5 Additional salt

After the detailed study of the salt-free case with respect to the role of the line charge density λ and the counterion valency v, we can now see what happens when we insert additional (monovalent) salt ion pairs.

Note that you can not distinguish between the counterions and the additional positive salt ions. Both will be referred to as *counterions*, while the negative salt ions will be called *coions*.

Also note, that the distributions that are measured in ESPResSo is the particle distribution, *not* the charge distribution. As in the previous cases, there was only a single ion type, both were identical. Now, however, we have another ion type with a charge of the opposite sign. To compute the charge distribution $P(r)$ from the given particle distributions $n_{\text{counterion}}(r)$ and $n_{\text{coion}}(r)$, you can use

$$P(r) = \frac{N_{\text{ci}}\, n_{\text{ci}}(r) - N_{\text{coion}}\, n_{\text{coion}}(r)}{N_{\text{ci}} - N_{coion}}$$

where Q_{rod} is the total charge of the rod and N_{ci} and N_{coion} are the total numbers of the respective ion types. This computation can either be done in the simulation script or externally, using `xmgrace` or `gnuplot`.

Task:
Modify the script such that you can insert additional monovalent salt ion pairs into the simulation.

Task:
Compute the charge density $P(r)$ using the MC solver and the simulation for 50 additional salt ion pairs. Again, different participants of the tutorial should do this for two different values of λ.
What is the Debye length l_B in the system?
Compare the results from PB theory with the simulation results. Do they match? Does it behave as expected?

References

[1] M. Deserno. *Counterion condensation for rigid linear polyelectrolytes*. PhD thesis, Universität Mainz, February 2000.

[2] Markus Deserno, Christian Holm, and Sylvio May. The fraction of condensed counterions around a charged rod: Comparison of Poisson-Boltzmann theory and computer simulations. *Macromolecules*, 33:199–206, 2000.