Tutorial 3

Introduction to ESPResSo: Kremer-Grest's Linear Polymer

Nadezhda Gribova, Olaf Lenz *

May 15, 2012
Institute for Computational Physics, Stuttgart University

Contents

*olenz@icp.uni-stuttgart.de

1 Introduction

Today, we will get acquainted with the simulation package ESPResSo. On that behalf, we will be using a simple model of a linear polymer. The model was first used by Kremer and Grest in 1986[1] (their paper can be found in the tutorial's materials collection) and is nowadays seen as a standard coarse-grained model for polymers.

Today's research on Soft Condensed Matter and allied fields has brought needs of having a flexible, extensible, reliable and efficient (parallel) molecular simulation package. For this reason ESPResSo (**E**xtensible **S**imulation **P**ackage for **Res**earch on **So**ft matter)[1] has been developed at the Max-Planck-Institute for Polymer Research in Mainz and the Institute for Computationaly Physics at Stuttgart University in the group of Prof. Dr. Christian Holm [2].

The software is probably the most flexible and extensible simulation package on the market. It is specially developed for coarse-grained molecular dynamics simulation of polyelectrolytes but not necessarily limited to this. It can be used even to model granular media, for example. The package was nominated for Heinz-Billing Price for Scientific Computing in 2003 [3].

In this short tutorial you will be introduced to the ESPResSo package as smooth as possible with a minimal set of skills.

From the user's point of view, ESPResSo is driven by the scripting language Tcl/Tk[2]. This means that the user interacts with the parallelized package core (that is written in the C programming language for optimal performance) via Tcl commands.

Using the software, we will be performing simulations of a Kremer-Grest polymer melt and analyze different observables.

2 Getting started

The first thing to be done is to find, download and compile the package. To learn how to do that, have a look at section 2.1 ("Quick installation") of the User's Guide of ESPResSo. A hardcopy is available in the CIP pool, but of course you can also find it in the web.

Comment: If you want to compile ESPResSo on your own computer, make sure that you have the necessary prerequisites installed. In the CIP pool, these requirements are already installed.

- The scripting language Tcl/Tk. Most Unix distibution provide a package called `tcl`. Note that you will also have to install the *development package*, that is usually called `tcl-devel` or similar.
- The parallelization environment MPI (optional, required for parallel execution). A number of implementations of MPI exist. Use either OpenMPI (packages `openmpi` and `openmpi-devel`) or MPICH2 (`mpich` and `mpich-devel`).
- The Fourier transform package FFTW ("Fastest Fourier-transform in the West") (packages `fftw` or `fftw3` and `fftw-devel` or `fftw3-devel`).

Once you have compiled ESPResSo, you can type in `./Espresso` to start the software. It will print out a short welcome banner and give you a prompt, where you can type in commands. To see that everything works fine, issue the command `code_info`, that will give you some information on the software. After that, you should have roughly the following lines on the screen:

[1] `http://espressomd.org`
[2] `http://www.tcl.tk`

```
> ./Espresso
0: Script directory: /auto.anoa/home/olenz/projects/espresso/obj.icp/default/scripts
*****************************************************
*                                                   *
*                    - ESPResSo -                   *
*                    ============                   *
*        A Parallel Molecular Dynamics Program      *
*                                                   *
* (c) 2010,2011                                     *
* The ESPResSo project                              *
*                                                   *
* (c) 2002,2003,2004,2005,2006,2007,2008,2009,2010  *
* Max-Planck-Institute for Polymer Research         *
* Mainz, Germany                                    *
*                                                   *
*****************************************************

ESPResSo is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

ESPResSo is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see <http://www.gnu.org/licenses/>.

>code_info
ESPResSo: 3.0.0
{ Compilation status { FFTW } { CONSTRAINTS } { TABULATED } { LENNARD_JONES }
  { BOND_ANGLE_COSINE } }
{ Debug status { } }
>
```

Congratulations, ESPResSo now runs on your machine!

3 Tcl tutorial

Tcl (Toolkit Command Language) is a dynamically interpreted programming language (*aka* scripting language). ESPResSo is an (extended) interpreter for the Tcl language. This means that you can type commands into your *script file* (*e.g.* `script.tcl`) and execute it via

```
./Espresso script.tcl
```

to see what it does.

You can even start ESPResSo without giving it a script name. In that case, ESPResSo will be started in interactive mode, where you can type your commands on the command line.

If you need help on how to use a Tcl command, you can use the Unix command `man`. For example, the following command will give you some help on the Tcl command puts:

```
man 3tcl puts
```

The `3tcl` is required to note that you need help about the Tcl command `puts`, not the Unix command of the same name.

If you need further and advanced language details please consult the official Tcl documentation[3].

3.1 Variables, Commands

- `x` is the variable name.
- `$x` refers to the value of the variable.
- `COMMAND` *parameter1 parameter2* `...` calls a function.
- `[ COMMAND` *parameter1 parameter2* `... ]` returns the value of the called function.

3.2 Assignments, Evaluation

- Simple text output can be carried out with `puts`:

```
puts "Hello Espresso \n"
puts "This is line 1"
puts "this is line 2"
```

- To assign values to a variable, use the command `set`:

```
set X "This is a string"
set Y 1.24
puts $X
puts $Y
```

- Comments are started with the "`#`" sign. Either semicolon ";" or newline character can be used to separate commands:

```
set X 1.2; # this is a comment
```

- To obtain the result of an evaluation of mathematical expressions you can use the `expr` command. Don't forget `[]`, since you are interested in the value that the operation returns.

```
set X  60
set Y  30
set Z [expr $X+$Y]
puts " X=$X and Y=$Y and X+Y=$Z"
set cosX [expr cos($X)]
puts "cos ($X) =  $cosX"
```

  Most operators and math functions that you might know from the C language are also valid in Tcl.

[3] `http://www.tcl.tk`

3.3 Comparisons, Looping

- The syntax of numeric comparison is as follows:

```
set x 5
if {$x == 5} {
  puts "$x is 5"
} else {
  puts "$x is not 5"
}
```

  Comment: Mind the spaces between the curled braces, they are important! If you forget the space, you'll get error messages about the statement in general.

- To loop, you can use a `for` loop. For example, to compute 10!, you can use

```
set factorial 1.0
for {set i 1} {$i <11} {incr i} {
  set factorial [expr $factorial*$i]
}
puts "10! is $factorial"
```

- Also, there is the `while` loop. Repeating the above example of computing 10!:

```
set factorial 1.0
set i 1
while {$i <11} {set factorial [expr $factorial*$i] ; incr i}
puts "10! is $factorial"
```

3.4 Lists

A basic data structure in Tcl is the list. It is an ordered collection of arbitrary objects (*e.g.* numbers, strings, other lists, ...) and can be stored in a variable.

- A list can be construed like this:

```
set x { 1 2 3 }
set y "1 2 3"
```

  Note here that a string can also be interpreted as a list! Whitespace characters delimit the different elements.

- To access the list data one can use `lindex` by using the corresponding index values:

```
set x "1 2 3"
puts "first element is [lindex $x 0]"
puts "second element is [lindex $x 1]"
puts "and the last [lindex $x 2]"
```

- One can access all the elements by using the `foreach` loop:

```
foreach j $x {
  puts "$j is item number $i in list x"
  incr i
}
```

- Also one can access list of lists:

```
set y "{l00 l01} {l10 l11} {l20 l21}"
puts "first element of second list is [lindex $y 1 1]"
puts "second element of third list is [lindex $y 2 1]"
```

- We can also find the length of a list by `llength`, append an element by `lappend`, or insert an element by `linsert`

```
set x "1 2 3 4";  # generate a list x
llength $x; # get the size of list x (number of elements)
lappend x 5 ;# add a new member end of list
puts "x is {$x}"; # print list again
set x [linsert $x 3 3a]; # insert an element "3a" at index 3
puts "x is {$x}"; # print list again
```

3.5 Adding new Tcl commands

You can easily add new commands/functions to Tcl like this:

```
proc sum {arg1 arg2}  {
set x [expr {$arg1 + $arg2}]
return $x
}
sum 1 4
```

3.6 Writing to a file

It is often useful to write the data into a file:

```
set file_handle [open "file.dat" "w"]; # open a file called file.dat to write,
puts $file_handle "This will go into file!"
for {set i 0} {$i <10} {incr i} {  puts $file_handle "counting $i"  }
close $file_handle # close the file channel
```

3.7 Arithmetic average and standard deviation

Task: (1 point)
Write a sequence of Tcl commands (a script) that computes the arithmetic mean

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

from a given list of real numbers. Use the math function `[expr rand()]` *to produce arbitrary number of real numbers between 0 and 1 to test your new command. Write your code into a file called* `task1.tcl`.

Task: (1 point)
Write a Tcl command that computes the standard deviation of uncorrelated samples into the same file

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2$$

Check your result with a smaller data set where you can verify the correctness manually.

4 Basics of ESPResSo

In this section we will review the basic ESPResSo-commands that will help you to understand the sample script. Note that a real life script may look much more complicated.

4.1 Units

Novice users must understand that ESPResSo uses no fixed unit system. Read subsection 1.4. (*On units*) in the ESPResSo User's Guide to understand what that means. In the following, we will use the Lennard-Jones units, where $\sigma = 1.0$ and $\epsilon = 1.0$.

4.2 Simulation parameters

There are a few global parameters of the whole simulation system. Some of them are dynamic, that is to say we can change them on the fly, some others can only be read. The main command to address these parameters is `setmd`.

```
setmd time_step 0.001; # this sets integrator's time step to 0.001
setmd box_length 10.0 10.0 10.0; # this sets cubic box L =10
```

To obtain the value of a global parameter, you can simply omit the value it should be set to:

```
puts "The current time step is [setmd time_step]\n"
```

4.3 Thermostat

By default, ESPResSo simulates an NVE ensemble. However, in soft matter research, one usually wants to simulate a system in the NVT-ensemble. To do that, it is necessary to turn on a *thermostat*. Here, we switch on the Langevin thermostat with a temperature `$temperature` and friction coefficient `$gamma`:

```
thermostat langevin $temperature $gamma
```

4.4 Particles

The power of the ESPResSo package lies in the flexibility to manipulate particle data, which is driven by the `part` command. Each particle in the system has a unique *particle id* that can be used to address the particle. For example, one can obtain information on a specific particle via

```
part 0 print pos
```

This command will return the position vector of the particle with id 0 in the form of a Tcl list.

Besides a position, each particle must have a `type`, which is specified via the *type id*. The type is important to define interactions between all particles of that type (see below). For example to create particle 0, give it the `type` 2 and place it at the position *(x,y,z)* we write:

```
part 0 pos $x $y $z type 0
```

To set up a polymer chain, the following command is used:

```
polymer $num_polymers $monomers_per_chain $bond_length SAW $hield
```

This command will create $*num_polymers* polymer chains with $*monomers_per_chain* monomers per chain. The length of the bond between two adjacent monomers will be set to $*bond_length*. The polymer is generated as a self-avoiding random walk (SAW). Monomers can't be closer to each other than $*shield*. For further details and options consult the ESPResSo User's Guide, subsection 4.2.1.

4.5 Interactions

To define interactions between particle types, one uses the `inter` command. For example, a LJ interaction between particles of type 0 can be defined as follows

```
set lj1_eps     1.0
set lj1_sig     1.0
set lj1_cut     1.12246
set lj1_shift   [calc_lj_shift $lj1_sig $lj1_cut]
set lj1_off  0.0
inter 0 0 lennard-jones $lj1_eps $lj1_sig $lj1_cut $lj1_shift $lj1_off
```

This setting corresponds to the following potential form

$$U(r) = 4\epsilon \left[\left(\frac{\sigma}{r - \text{off}} \right)^{12} - \left(\frac{\sigma}{r - \text{off}} \right)^{6} + \text{shift} \right]$$

The command `calc_lj_shift` calculates the shift of the LJ potential for given sigma and cutting radius.

4.6 Integrating the system

To integrate the equations of motion, ESPResSo invokes the integrator via the `integrate` command. The only argument it needs is the number of time steps to integrate. Most of the basic simulation parameters must be set before integration.

4.7 Analysis

ESPResSo has a lot of built-in tools for analysis of the system. For example, the command `analyze energy` has several variants:

```
analyze energy
analyze energy total
analyze energy nonbonded $typeid1 $typeid2
```

It returns the energies of the system. The first variant returns all the contributions to the total energy. The second variant returns only the numerical value of the total energy. The last variant computes the energy of the non-bonded interactions.

There are many other analysis functions in ESPResSo. To learn what they do and how they work, have a look at the User's Guide.

4.8 Warmup integration

There is one special issue to be considered when the particles are set up randomly. In this case, some of the particles may overlap, so that they would have extremely high interactions energies. This energy first has to be removed from the system. To do that, we first have to *cap* the forces, *i.e.* we modify the potential such that it never gets larger than the capping value. Then we can safely start to integrate the system.

Note that we should not use this setting to compute and analyze the observables, as it is actually unphysical. Therefore, as soon as we have removed the extreme energies, we should unset the capping to perform the main integration.

5 The Kremer-Grest polymer melt

In the article [1] (that is part of the tutorial's material), an efficient algorithm for simulating polymers was proposed. In this MD algorithm each particle is weakly coupled to a heat bath (Langevin thermostat) to mimic the influence of the solvent on the polymers. The equations of motion are:

$$\ddot{\mathbf{r}}_i = -\nabla U_i - \Gamma \dot{\mathbf{r}}_i + \mathbf{W}_i(t) \tag{1}$$

where U_i is potential energy of the bead i, Γ is the bead friction describing the drag of the solvent on the bead and $\mathbf{W}_i(t)$ describes the random kicks of the heat bath acting on the monomer.

5.1 System setup

Purely repulsive LJ potential acting between any pair of particles:

$$U_{ij}^0 = \begin{cases} 4\epsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^6 + \frac{1}{4} \right], & r_{ij} \leq \sigma 2^{1/6} \\ 0, & r_{ij} > \sigma 2^{1/6} \end{cases} \tag{2}$$

In addition, there is a FENE (finite extension nonlinear expander) potential between particles which are topologically nearest neighbours along the polymer chains:

$$U_{ij}^{\mathrm{ch}} = \begin{cases} -0.5 k R_0^2 \ln \left[1 - (r_{ij}/R_0)^2 \right], & r_{ij} \leq R_0 \\ 0, & r_{ij} > R_0 \end{cases} \tag{3}$$

which accounts the additional bond interaction. The FENE potential is set up via

```
inter 0 FENE $fene_k $fene_r
```

where `$fene_k` and `$fene_r` correspond to k and R_0 in eq. 3.

5.2 Analysis

The Rouse model for the dynamics of a single polymer describes the motion of an ideal chain immersed in a viscous solvent. The model neglects self-repelling of the chain monomers as well as any hydrodynamic effects. For an ideal chain (random walk) of N monomers, one finds the longest relaxation time (Rouse time) $\tau_N \propto N^2$. For the motion of a single monomer, the following relations hold:

$$g_1(\tau) = \langle [\mathbf{r}_j(t) - \mathbf{r}_j(t+\tau)]^2 \rangle \propto \begin{cases} \tau^{1/2}, & \tau \ll \tau_N \\ \tau, & \tau \gg \tau_N \end{cases} \tag{4}$$

$$g_2(\tau) = \langle ([\mathbf{r}_j(t) - \mathbf{R}_{\mathrm{cm}}(t)] - [\mathbf{r}_j(t+\tau) - \mathbf{R}_{\mathrm{cm}}(t+\tau)])^2 \rangle \propto \begin{cases} \tau^{1/2}, & \tau \ll \tau_N \\ \tau^0, & \tau \gg \tau_N, \end{cases} \tag{5}$$

where $\langle \cdot \rangle$ denotes averaging over all possible time origins t and over different segments and $\mathbf{R}_{\mathrm{cm}}$ is the position vector of the centre of mass of the chain,

$$\mathbf{R}_{\mathrm{cm}} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{r}_i \, .$$

The center-of-mass motion finally follows

$$g_3(\tau) = \langle [\mathbf{R}_{\mathrm{cm}}(t) - \mathbf{R}_{\mathrm{cm}}(t+\tau)]^2 \rangle \propto \tau. \tag{6}$$

At the Rouse time, $\tau \approx \tau_N$, all three g functions attain similar values:

$$g_1(\tau_N) \approx g_2(\tau_N) \approx g_3(\tau_N) \approx \langle R_G^2(N) \rangle. \tag{7}$$

where $\langle R_G^2(N) \rangle$ is the mean-square radius of gyration. Qualitatively, equations (4)–(6) mean that the monomer motion is dominated by fluctuations around the center of mass on small timescales, until the average displacement reaches approximately R_G. On longer timescales the diffusion of the chain as a single object dominates. On intermediate timescales there is a smooth crossover between the two limiting regimes.

5.3 The Kremer-Grest polymer melt

Task: (0 points)
Read the article [1].

Task: (0 points)
Study the file `tutorial3.tcl`*, which simulates the system studied in [1]. Pay attention to the warmup integration, how sampling is done, what observables are written out and to which files.*

Task: (1 point)
Look into the User's Guide (section 8.3) what the command `analyze append` *does. Explain how it works and why it is used in the script.*

Task: (3 points)
Run the tutorial script. Plot g_1, g_2, g_3. Do they obey the expected scaling given by Equations (4)-(6)*? Try to estimate τ_N from your data.*

Task: (4 points)
Change the script to simulate non-bonded LJ particles instead of polymer chains. A rough pattern for assigning initial positions for LJ particles is provided in part Interaction and Particle setup *of the script. For non-bonded particles it does not make sense to use g_2 and g_3, since the center of mass of a chain of length N is exactly the position of the one bead. To use g_1 one has to specify explicitly the chain start (0), the number of chains (n_part) and the length of each chain (1). For the proper syntax consult the User's Guide, section 8.2. Compare g_1 for LJ particles with the one of polymer chains. Is there any difference? Where does the difference come from?*

References

[1] G.S. Grest and K. Kremer. Molecular dynamics simulation for polymers in the presence of a heat bath. *Physical Review A*, 33(5):3628–3631, 1986.

[2] HJ Limbach, A. Arnold, and B. Mann. ESPResSo; an extensible simulation package for research on soft matter systems. *Computer Physics Communications*, 174(9):704–727, 2006.

[3] A. Arnold, BA Mann, HJ Limbach, and C. Holm. ESPResSo–An Extensible Simulation Package for Research on Soft Matter Systems. *Forschung und wissenschaftliches Rechnen*, 63:43–59, 2003.