Tutorial

# 7: Introduction to Espresso: Lennard-Jones Liquid *

Mehmet Süzen †

June 14, 2007
SimBio group, FIAS, Frankfurt

## Contents



*Initial LJ script written by H. J. Limbach, modified by M. Sega and further modified by M.Süzen
†suzen [add] fias dot uni-frankfurt dot de

# 1 Introduction

Today's research on Soft Condensed Matter and allied fields has brought needs of having flexable, extensible, reliable and efficient (parallel) molecular simulation package. For this reason **ESPResSo** (Extensible Simulation Package for Research on Soft matter) [1] has been developed in Max Planck Institute for Polymer Research, Mainz by the Group of PD Dr. Christian Holm [2]. The Espresso package is probably the most flexable and extensible simulation package in the market. It is specially developed for coarse-grain molecular dynamics simulation of polyelectrolytes but not necessarily limited to this. It can be used even in simulating granular media for example. Package has nominated for Heinz-Billing-Preis 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.

What is Espresso? It is not a coffee, indeed. It is extensible, efficient Molecular Dynamics package specially powerful on simulating charged systems. In depth information about the package can be found in the relavant sources [1, 3] and a recent paper [2].

From User point of view, Espresso is driven by TCL(/TK) [4], so user can interact with the package core via command line interface (CLI) or scripts by using TCL scripting language. So, basically an Espresso command or directive is transparent to user regardless of its implementation level, either it is implemented on the core C-level or TCL-level.

First thing a user can do is type $Espresso$ in the command line shell (assuming you have compiled package), then you will end up in CLI of Espresso, and an Espresso command $code\_info$ can be issued to see how Espresso compiled.

```
% suzen@limasol:~$ Espresso
0: Script directory: /home/fias/suzen/src/local/espresso_dev/lib/ESPResSo/scripts/

**************************************************
*                                                *
*                    - Espresso -                *
*                    ============                *
*      A MPI Parallel Molecular Dynamics Program     *
*                                                *
*                                                *
* (c) 2002-2006                                  *
* Max-Planck-Institute for Polymer Research      *
* Mainz, Germany                                 *
*                                                *
**************************************************
% code_info
ESPResSo: v1.9.8s (Seska), Last Change: February 7th, 2006
{ Compilation status { production } { MPI lam } { FFTW2 }
{ TK } { PARTIAL_PERIODIC }
{ ELECTROSTATICS } { MASS } { EXTERNAL_FORCES }
{ CONSTRAINTS }
{ LENNARD_JONES } { BOND_ANGLE_COSINE } }
{ Debug status { MPI_CORE FORCE_CORE } }
%
```

What kind of output you are receiving from $code\_info$ command?

# 2 Short TCL Tutorial

Tcl (Tool Command Language) is a very powerful but easy to learn dynamic interpreted programming language. User can write any valid tcl code on the Espresso CLI, as well as valid Espresso commands. Here we will review basic TCL tutorial [5] in Espresso CLI. Now type $Espresso$ and get into CLI and make yourself familiar with basics of TCL.

## 2.1 Assignments, Evaluation

Simple text output, or a print statement can be carried out with $puts$

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

The Assignment command is $set$

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

C-like backslash squences can be used along $puts$ and comments are as follows

```
set X 1.2 ;# this is a comment
set Y 2.1 ;# another comment
puts "\t tab \t another tab \n X=$X and Y=$Y "
```

To get the result of an evaluation of mathematical expressions you can use $expr$ command

```
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"
```

Mostly C-like operators and math functions are valid TCL syntax.

## 2.2 Comparisons, Looping

Syntax of numeric comparison is as follows

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

You might want to write this in different lines by using backslash [1]

[1] You must keep in mind that one must use backslash (\) for line continuation in Espresso CLI

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

You can loop with standard $for$ or $while$ constructs. For example finding $10!$;

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

Or with a $while$ construct

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

## 2.3 The List

An ordered collection of entities can be assigned to a variable that makes it a $list$. This is the basic data structure in TCL. To access the list data one can use $lindex$ by using corresponding index value.

```
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 $foreach$ looping consruct as well

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

Also we 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 $lappend$ and inserting 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
```

### 2.4 Adding New Tcl Command

In Tcl there is actually no distinction between commands (often known as 'functions' in other languages) and "syntax" [5]

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

### 2.5 Writing to a File

It is useful to write the data into a file. For example

```
set file_handle [open "file.dat" "w"] ;# open a file called file.dat to write,
                                      ;# and file channel is $file_handle
puts $file_handle "This will go into file!"
for {set i 0} {$i <10} {incr i} {  puts $file_handle "counting $i"  }
close $file_handle ;# close file channel
set file_content [exec cat file.dat] ;# with exec one can run shell commands
puts "$file_content"
exec rm file.dat ;# remove the file
```

For further and advanced language details please consult with official tcl documentation [4].

**Task 1.** *Write two tcl procedures (custom tcl commands) to compute an arithmetic average*

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

*and standard deviation* $\sigma$*,*

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

*out of given list of real numbers respectively. Use* $rand()$ *command to produce arbitrary number of real numbers between 0 and 1 to test your new commands. Write your code into a file called task1.tcl and run as follows* **Espresso task1.tcl 1**. *Check your result with smaller data set that you can verify the correctness manually.*

## 3 Basics of Espresso

An Espresso script is a tcl script that is a simulation engine that drives the C-core of the package. In this section we will review very basic commands that will help you to understand the sample script, which is quite basic. Real life script may look much more complicated.

## 3.1 Units

Novice users must understand that Espresso has no fixed unit system. Unit system is set by user. Conventionally reduced units are employed, in other words LJ units. [2]

## 3.2 Simulation Parameters

There are global parameters of simulation system. Some of them are dynamic, that is to say we can change on the fly, some others only to read or both. 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 100.0 100.0 100.0 ;# this sets cubic box L =100
set number_of_particles [setmd max_part] ;# read the number of existing particles
```

Espresso needs to know which integrator to use for dynamics. It can use NVE or NVT (Langevin) as well as NPT-isotropic ensembles.

```
thermostat off ;# this implies to use NVE
# this implies to use langevin thermostat (NVT ensemble)
#with temperature set to 1.0 and damping coefficient to 0.5
thermostat langevin 1.0 0.5
```

## 3.3 Assigning Interactions and Particle Properties

Power of Espresso package lies on manipulation flexability of particle data. Which is driven by $part$ command that recognise unique particle id. Each particle must be a member of a group which is called $type$, and interactions among those types can be defined through $type$ number with $inter$ command. [3] For example to place a particle $id$ 0 and $type$ 0 at given $(x, y, z)$

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

it is also possible to read the information on the given particle

```
part 0 print pos
```

which returns position vector of particle id 0. LJ interaction among type 0 particles can be defined as follows

```
set lj1_eps     1.0
set lj1_sig     1.0
set lj1_cut     1.12246
set lj1_shift   0.0
set lj1_offset  0.0
inter 0 0 lennard-jones $lj1_eps $lj1_sig $lj1_cut $lj1_shift $lj1_offset
```

This setting corresponds to following potential form

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

[2] If we have charges there is additionally a concept of Bjerrum length, consult Espresso original paper for more details.

[3] Note that, In most electrostatic algorithms, one does not need a type id for interaction specification.

## 3.4 Generating Data: Lennard-Jones Liquid Simulation

Here we insvestigate static and dynamics properties of Lennard-Jones Liquid Simulation. Espresso invokes integrator with $integrate$ command, only argument it needs is number of time steps to integrate. Most of the basic simulation parameters must be set before integration. For convenience it is a general practice to write simulation data to a disk. Espresso provides quite powerful tool, i.e. $blockfile$ structure/command set. Basic idea behind blockfile tool is, writing different group of data like particle information, variable information and other simulation related information into logical blocks consequetively.

**Task 2.** *Inspect the file $lj\_tutorial.tcl$. Where system is mimic to case study 4 of chapter 4, in the seminal book [6]. How one can define truncated-shifted potential in $lj\_tutorial.tcl$? ( keep on mind that Espresso has already a factor of 4 at shifted part with cutoff $r_c = 2.5$)*

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

$$U(r)^{tr-sh} = \begin{cases} U(r) - U(r_c) & r_c > r \\ 0 & r_c < r \end{cases}$$

**Task 3.** *Inspect the file $lj\_functions.tcl$ for the procedure $save\_sim$ and how it is called in $lj\_tutorial.tcl$ file. Delete unnecessary information that is being written into $sim\_info.dat$, then run $lj\_tutorial.tcl$ and check $data$ directory for the simulation data file. Inspect the simulation data file $sim\_info.dat$. Apply following in command line;*

```
Espresso lj_tutorial.tcl 1
```

*Compute how much space you have saved by deleting some data?*

**Task 4.** *Inspect and run $blockfile\_read.tcl$ to see how to read offline data. Modify this script to print out frame time and average of position vector components $x_{avg}, y_{avg}, z_{avg}$*

## 3.5 Simple Error Analysis on Time Series Data with $uwerr$

Espresso provides build-in time series analysis tool called $uwerr$. Inspect how $uwerr$ is used to determine time average and its standard error of mean of Total energy per particle, by using Espresso **analyze energy** command.

**Task 5.** *Inspect what* **analyze energy** *command returns. Make the similar error analysis for kinetic temperature, kinetic energy and potential energy, in the main script.*

**Task 6.** *Inspect what* **analyze pressure total** *command returns. Make the similar error analysis for total pressure.*

**Task 7.** *Plot time evalution of pressure and energy that is written under* $data$ *directory as* $energy.dat$

**Task 8.** *Radial Distribution Function (RDF): Run* $rdf.tcl$*, inspect the code and plot rdf from* $data/rdf.dat$*. Try different parameters for* **analyze rdf** *command, such as bin size. What do you observe?*

**Task 9.** *Velocity Autocorrelation Function (VACF): Run* $vacf.tcl$*, inspect the code and plot vacf from* $data/vacf.dat$*. Where vacf* $C(t)$ *can be computed in direct way*

$$C(t) =< \mathbf{v}_i(0)\mathbf{v}_i(t)$$

$$C(t) = \frac{1}{N}\sum_{i=0}^{N} \mathbf{v}_i(0)\mathbf{v}_i(t)$$

*where* $N$ *is the number of particles. Try to modify* $vacf.tcl$ *by using* $vecsub$ *and* $veclen$ *or a like tcl math functions.*

**Task 10.** *Mean Square Displacement (MSD): Run* $msd.tcl$*, inspect the code and plot msdfrom* $data/msd.dat$*. msd can be computed simply:*

$$< \Delta r(t)^2 >= \frac{1}{N}\sum_{i=0}^{N} \Delta\mathbf{r}_i(t)^2$$

$$< \Delta r(t)^2 >= \frac{1}{N}\sum_{i=0}^{N} |r_i(t) - r_i(0)|^2$$

## References


[1] http://www.espresso.mpg.de/.

[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.

[4] http://www.tcl.tk.

[5] http://www.tcl.tk/man/tcl8.5/tutorial/tcltutorial.html.

[6] Daan Frenkel and Berend Smit. Understanding molecular simulation. 2002.