

Molecular dynamics analysis libraries, part 2

*with an example based on the dynamics
in the physiopathology of gelsolin*



Toni Giorgino
toni.giorgino@cnr.it

www.giorginolab.it
@giorginolab



github.com/giorginolab/GSN-Tutorial-BCN-2021

Master projects available!

Part I. Motivation

Finnish-type amyloidogenic gelsolin variant -
an example of protein dynamics playing a role in
proteotoxicity and drug design discovered by MD.

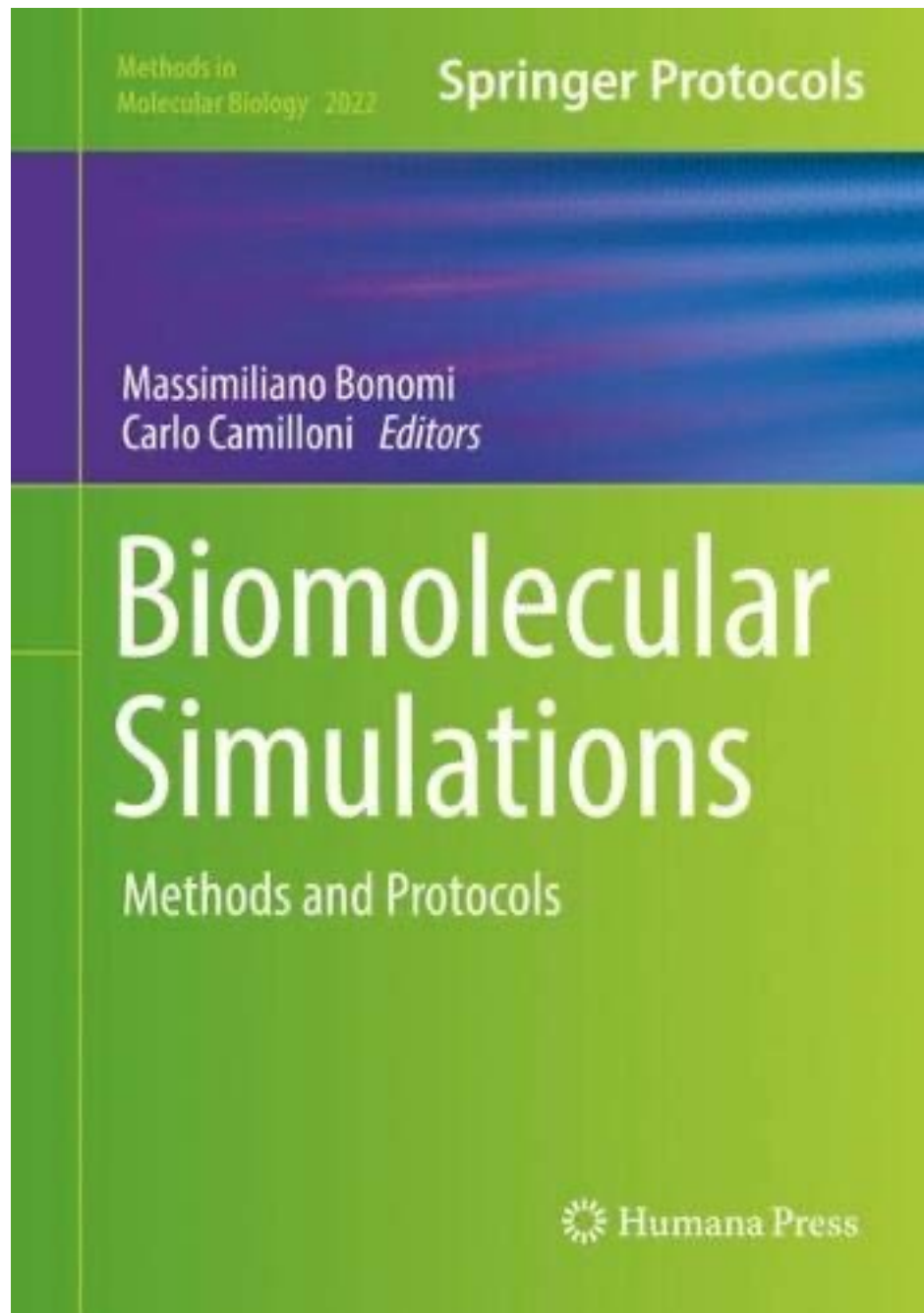
Part II. Practice

MD analysis libraries: intro and reproduction of
the analysis* shown in the paper.

* Marked with
this symbol →



Part II.
MD analysis libraries
(Practice)



Springer, ISBN 978-1-4939-9608-7

[Preprint here.](#)



Chapter 20

Analysis Libraries for Molecular Trajectories: A Cross-Language Synopsis

Toni Giorgino

Abstract

Analyzing the results of molecular dynamics (MD)-based simulations usually entails extensive manipulations of file formats encoding both the topology (e.g., the chemical connectivity) and configurations (the trajectory) of the simulated system. This chapter reviews a number of software libraries developed to facilitate interactive and batch analysis of MD results with scripts written in high-level, interpreted languages. It provides a beginners' introduction to MD analysis presenting a side-by-side comparison of major scripting languages used in MD and shows how to perform common analysis tasks within the Visual Molecular Dynamics (VMD), Bio3D, MDTraj, MDAnalysis, and High-Throughput Molecular Dynamics (HTMD) environments.

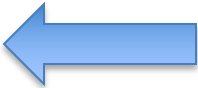
Key words Molecular dynamics, Trajectory analysis, Scripting languages, VMD, Bio3D, MDTraj, MDAnalysis, HTMD

1 Introduction

The backbone of molecular dynamics (MD)-based methods is to integrate the equations of motion of a system with a given Hamiltonian. The integration is performed by an MD engine with a finite time-step, sufficiently fine to capture the fastest motion of interest (e.g., bond vibrations). Commonly, one is interested in long-time behavior, and therefore, simulations are performed for several orders of magnitudes longer than the integration time-steps, making integration the most compute-intensive component of the MD workflow; this, in turn, makes it natural to keep a record (“trajectory”) of the states through which the system goes for later analysis.

The objective of this chapter is to provide an operative introduction to the libraries most often used in MD analysis in combination with the corresponding programming languages. In particular, I strive to provide (a) a side-by-side view of the constructs most important for analysis (including file input and output

Analysis of MD trajectories

- Interactive: VMD, Chimera, PyMol...
 - Intuitive
 - Suitable for one-off tasks
- Scripted: for...
 - repeated analysis (e.g. ensembles)
 - custom tasks (your own ideas)
 - automated analysis, e.g. machine learning
- Analysis libraries are needed 

MD analysis libraries

Library	Language
VMD	TCL
Bio3D	R
MDAnalysis	Python
MDTraj	Python
HTMD/MoleculeKit	Python



- We'll show examples for **HTMD**, but there are *direct* equivalents in the others.
- In fact, converting is a useful exercise.
- See the Chapter.

MoleculeKit*

* part of HTMD

<https://github.com/Acellera/moleculekit>

The screenshot shows the MoleculeKit documentation page. The top navigation bar is dark blue with the Acellera logo, links for DOCS, HELP, and DOWNLOAD, a GitHub icon, a YouTube icon labeled ACCELLERATIVE, and a search bar. The left sidebar is light gray and contains a 'Contents' menu with links to HTMD, Moleculekit (highlighted), Installation, The Molecule class, The SmallMol class, Molecule tools, Projections, Tutorials, ACEMD3, ACEMD (deprecated), and Parameterize. The main content area has a light blue background and features a breadcrumb 'Docs » Moleculekit', a large 'Moleculekit' heading, and three paragraphs of text. The first paragraph welcomes users to the documentation. The second paragraph describes Moleculekit as a Python library for manipulating biomolecular structures. The third paragraph lists the two main classes, *Molecule* and *SmallMol*, and their typical uses. The fourth paragraph mentions visualization support for VMD and NGL. The fifth paragraph provides the GitHub link and mentions the issue tracker.

acellera® DOCS HELP DOWNLOAD ACCELLERATIVE Search docs

Contents

- HTMD
- Moleculekit**
 - Installation
 - The Molecule class
 - The SmallMol class
 - Molecule tools
 - Projections
 - Tutorials
- ACEMD3
- ACEMD (deprecated)
- Parameterize

[Docs](#) » Moleculekit

Moleculekit

Welcome to Moleculekit's documentation!

Moleculekit is a python library that provides object-oriented classes and methods for manipulation of biomolecular structures. With a few simple python commands you can read, write and convert between many different file formats, align structures and calculate various projections from your molecules such as RMSD, RMSF, secondary structure, SASA, contact maps and much more.

The two main classes are *Molecule* and *SmallMol*. *Molecule* is used for manipulating larger molecules and typical molecular dynamics (MD) related formats such as PDB/PSF/PRMTOP etc. *SmallMol* (in conjunction with *SmallMolLib*) on the other hand is meant to handle large libraries of small molecules such as SDF files in a fast but more limited manner.

Both classes support visualization in both the VMD desktop viewer and the NGL jupyter-notebook viewer for easy visualization of the molecular structures.

The source code of Moleculekit is available under <https://github.com/Acellera/moleculekit> and you can use the issue tracker there to report any bugs or issues you encounter using the library.

1. Please clone or download...

github.com/giorginolab/GSN-Tutorial-BCN-2021

which contains the papers and **data files**.

2. Open the Colaboratory link



It is a “live” Python notebook to run your code on Google’s servers. (Account needed)

Alternatively, work locally on your PC.

Loading a trajectory

- First, import (activate) the library
- Then, load the *topology* and *trajectory**

_____ VMD _____

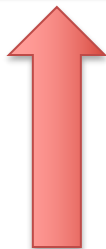
```
set t [mol new $pdb]
animate delete all
mol addfile $xtc waitfor all
```

_____ MDAAnalysis _____

```
import MDAAnalysis as mda
t = mda.Universe(pdb, xtc)
```

_____ HTMD _____

```
from htmd.ui import *
t=Molecule(pdb)
t.read(xtc)
```



_____ Bio3D _____

```
library(bio3d)
tp <- read.pdb(pdb)
tp$xyz <- read.dcd(dcd)
```

_____ MDTraj _____

```
import mdtraj as mdt
t = mdt.load(xtc, top=pdb)
```

* Atom names, types, bonds, etc.
Usually a PDB or PSF file.

Several formats are supported.
Here we use PDB+XTC.

VMD

```
# Number of frames
molinfo top get numframes

set t [atomselect top all]
$t num;           # Number of atoms

$t frame 0
$t get {x y z}; # Coordinates

pbc get;          # Unit cell
```

MDAnalysis

```
# Self-explanatory
t.atoms.n_atoms
t.trajectory.n_frames

# Atoms by 3
t.atoms.positions

# Unit cell
t.atoms.dimensions
```

HTMD

```
t.numFrames
t.numAtoms

# Atoms by 3 by frames
t.coords

# Unit cell
t.box[:,0]
```

Bio3D

```
nrow(tp$xyz)      # 40 frames
nrow(tp$atom)     # 28799 atoms

## Accessing coordinates in frame 0
## reshaped for convenience
xyz <- tp$xyz[1,]
xyz <- matrix(xyz, ncol=3, byrow=T)

## Or: array(xyz,c(40,3,28799))
```

MDTraj

```
# Number of frames
len(t)

# Frames by Atoms by 3
t.xyz.shape
# Coordinates in frame 0
t.xyz[0]

# Unit cell
t.unitcell_lengths[0,:]
```

Access molecular data

Access molecular data

⇒ `t.numFrames`

⇒ `t.numAtoms`

⇒ *# Atoms by 3 by frames*

`t.coords`

Unit cell

`t.box[:,0]`

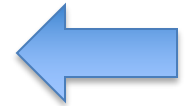
Selections

HTMD

```
y72_oeta = t.atomselect("resid 72 and  
↪ name OH and chain 0")
```

```
w288_chi1 = t.atomselect("resid 288 and  
↪ name N CA CB CG and chain 0")
```

```
t.occupancy[y72_oeta]
```



*

* This library uses VMD-like atom selection strings

Filters

Make a duplicate

```
bb = t.copy()
```

Keep backbone only

```
bb.filter("backbone")
```

Keep frame 0 only

```
bb.dropFrames(keep=0)
```

Write resulting PDB file

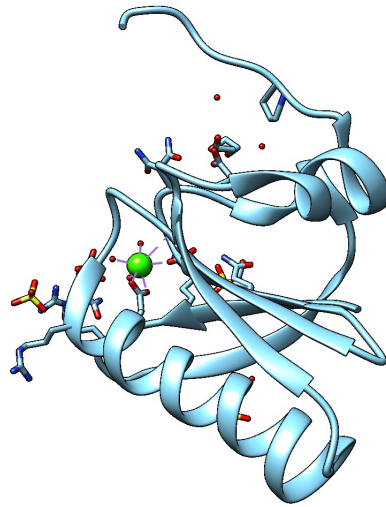
```
bb.write("bb_frame0.pdb")
```

Alignment

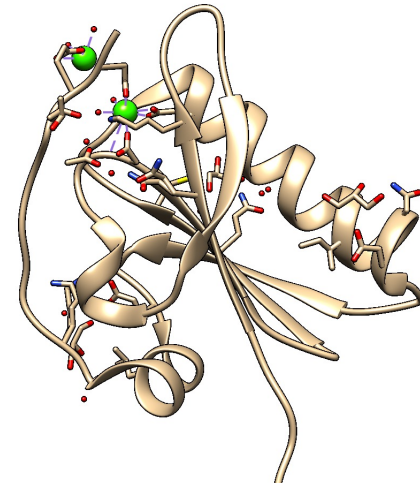
```
meas_t.align("backbone",meas_r)
```

Calculations are often performed *after* a rigid transformation which *optimally* superimposes two structures (or two frames). *It removes diffusion.*

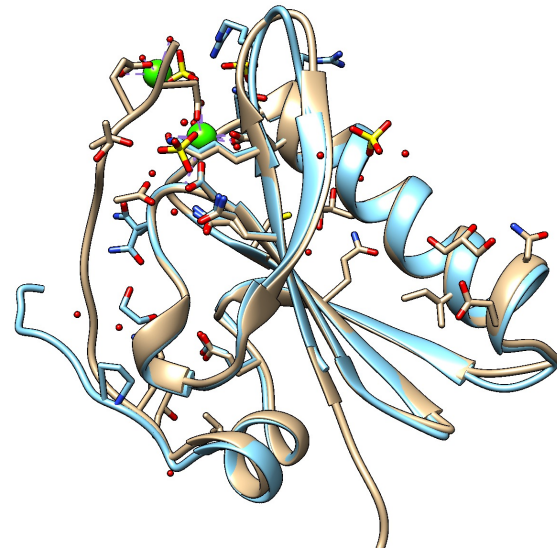
Before



Minimize
RMSD



After



RMSD

Is the *mean squared* displacement between two sets of atoms

$$\text{RMSD}(\mathbf{x}, \mathbf{y}) = \sqrt{\frac{\sum_{i=1}^{N_{\text{atoms}}} (\mathbf{x}_i - \mathbf{y}_i)^2}{N_{\text{atoms}}}}$$

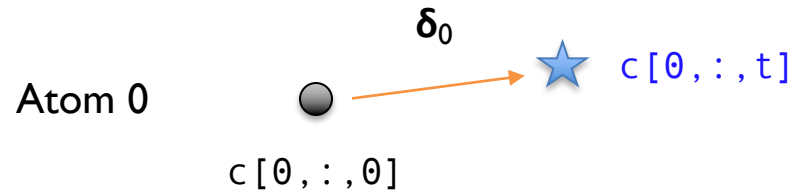
`moleculekit.util.molRMSD(mol, refmol, rmsdsel1, rmsdsel2)`



Must have the same number of atoms

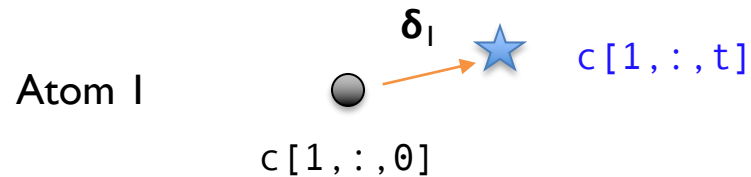
RMSD

Coordinates array: `coords[atom_index , axis , frame]`

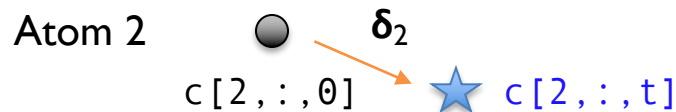


$$\delta_i = c[:, :, t] - c[:, :, 0]$$


(vectors)



$$|\delta_i| = \sqrt{\delta_{ix}^2 + \delta_{iy}^2 + \delta_{iz}^2}$$



 = Time 0

 = Time t

Data files (directory data)

	Apo form	In complex with nanobody Nb11
Wild type	WT	WT+Nb
D187N mutant	D187N	D187N+Nb

For each combination you will find:

- a PDB file
- a PSF file
- an *unwrapped* trajectory in XTC format (10 ns/frame)

IF present, Nb was held restrained

Useful for RMSD-based calculations

MoleculeKit has ready-made *projections*

- MetricRmsd
 - from `moleculekit.projections.metricrmsd` import *
 - <https://software.acellera.com/docs/latest/moleculekit/moleculekit.projections.metricrmsd.html>
- MetricFluctuation
 - from `moleculekit.projections.metricfluctuation` import *
 - <https://software.acellera.com/docs/latest/moleculekit/moleculekit.projections.metricfluctuation.html>

**Now we open the Colaboratory
Notebook and try to solve the
exercises. 😊**

**Advanced: rewrite it to
use a different library.**