Analysis libraries for molecular trajectories: a cross-language synopsis

Toni Giorgino

October 1, 2018

To appear in:
Biomolecular Simulations: Methods and Protocols
Edited by M. Bonomi and C. Camilloni

Corresponding author's address/affiliation:
Biophysics Institute, National Research Council of Italy
Department of Biosciences, University of Milan
Via G. Celoria 26, I-20133, Milan, Italy

Running head:
MD analysis libraries: a synopsis

Summary

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 show how to perform common analysis tasks within the VMD, Bio3D, MDTraj, MDAnalysis and HTMD environments.

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 operations); and (b) a side-by-side view of the object models used with reference to a simple but realistic analysis task.

This review is restricted to a few MD analysis libraries usable in *interpreted* (also known as scripting) languages, because they are best suited for interactive and rapid prototyping tasks. The chapter will be focused on five libraries which are actively developed, open-source, (Table 1), and whose scope was mainly trajectory analysis rather than modeling (although the line between the two may be blurred; Note 1 lists additional libraries).

Table 1 around here

2 Background

One important output of MD simulations are so-called *trajectory files*, i.e. the record of the coordinates of particles composing a system, taken at regular intervals (in atomistic simulations particles model individual atoms, while in coarse-grained models they represent more generic "beads"). While MD runs occupy computing resources for days or months, the analysis of trajectories is generally fast enough to enable an "iterative" hypothesis-calculation-assessment development cycle, e.g. in search of collective variables, collective modes, or any other of the observables which are most expressive for the system at hand and which can be computed from the trajectory.

Other chapters of this book presented a wealth of tools to perform specialized analysis types. Such tools can be distributed either as command-line utilities (e.g., GROMACS' utilities [1], Amber project's CPPTRAJ [2]), or with graphical user interfaces (either stand-alone, or embedded in molecular viewers; see Note 2). Of particular importance is the PLUMED library: originally developed for biasing MD simulations along selected collective variables (CV), the array of CVs has become increasingly rich and expressive [3, 4]. The libraries can therefore be used to perform analysis on pre-computed trajectories, defining the observables to be computed and atom sets through PLUMED's syntax [5] which, while not as general as a general-purpose programming language, is still very expressive for structure-oriented computations to be performed on each trajectory frame.

Developing one's own analysis routine in the form of computer code is, however, necessary whenever pre-made tools fall short of the task. This is a frequent occurrence for advanced MD users, especially when involved in method development. Traditional scientific computing languages such as Fortran, C and C++ in their "bare" form do not suit well the analysis of MD trajectories for two reasons: first, processing trajectories requires parsing a wealth of molecular formats, which have been developed over time to accommodate the needs of ever-increasing scales of simulations; these formats do not only encode the coordinates of atoms, but also a number of important attributes such as masses, charges and chemical bonding. Second, and related, the analysis of biological macromolecules does in large part make use of chemical (e.g., how does one tell protein from ligand from water?) or structural (e.g., how does one distinguish secondary structure elements?) characteristics of the underlying system. Accessing these atomic attributes becomes easier in presence of an appropriate *object model* specifying (a) which are the entities modeled in software, (b) how are they related (e.g. by chemical connectivities) and (c) what are their attributes (e.g. atoms have beta factors, bonds have orders) and (d) the methods that can be called on each. Developing a suitable object model is no easy task, greatly simplified in high-level, object oriented languages.

3 Programming languages

The scripting languages underlying the libraries examined in this review are TCL, R, and Python 3. They have in common their being dynamic (i.e. functions can be defined at run-time) and dynamically typed (i.e. there is no need to pre-declare the types of variables; but see Note 3 for remarkable cases when this is useful).

It will be out of the scope of this chapter a discussion of the details of each programming language (easily found outside of the scientific literature); nor shall it provide a systematic description of the feature of each library, for which the corresponding reference manuals are the best and most updated resource (see Note 4).

3.1 TCL

The TCL (originally Tool Command Language) was created in 1988 as an interpreted language suitable for embedding in other software. It has an important role in the analysis of MD simulations mostly because it is the language of choice for the Visual Molecular Dynamics (VMD) software [6], an open-source package enabling the manipulation of long MD-derived trajectories (Section 5.1).

The structure of the TCL language is somewhat unusual in the sense that it is centered around strings (function bodies, lists, and numbers, all being strings by default) and a Polish notation for function calls – i.e. $f(x, y)$ is written as `[f $x $y]`. Square brackets execute the function which they contain, replacing the return value, while curly braces quote strings (including function bodies). Other features are:

- Variables are prefixed by `$` to be replaced by their value. A rule of thumb is therefore to use `$` when reading variables, and not when modifying them.
- Variables of outer scopes are *not* visible by default; they are exposed by constructs such as `global` (globals), `upvar` (access the upper evaluation frame) and `variable` (variables bound to a namespace).
- Lists are space-separated strings (items can be quoted by curly braces if necessary). Functions such as `llength` and `lindex` provide array-like access (including nested ones). Indices start from 0.
- There are two types of associative hashes, namely *arrays*, which use round parentheses and can *not* be nested; and *dictionaries*, which can be nested.
- Mathematical expressions can be written in the more customary infix notation if evaluated with `expr`.

3.2 R

The R programming language derives from the S language, itself rooted in '70s efforts at the Bell Labs to provide an interactive environment for statistical calculations. R is also the name of the interpreter, which is actively developed and distributed as an open-source project [7].

R is a higher-level language still, and its features enable a programming style which is not conductive to meaningful parallels with the other two languages considered. For example, instead of loops, functional "apply" is encouraged (and sometimes necessary for efficiency reasons); it is therefore excluded from some of the syntax comparisons. Implicit rules often allow one not to concern himself with array shapes, which for the most part follow the "natural behavior", carrying over annotations such as row and column names. Also, functions are heavily overloaded by optional arguments, so that e.g. the `seq` function will generate all kinds of numeric sequences (given length vs. given spacing and so on); likewise many variations of text parsing are accommodated by (say) the `read.table` function, or equivalent ones provided in external packages.

For the reasons above, R is a natural fit for statistics-heavy computations. Other arguably attractive features of the R language are: (a) its expressive functional foundation, and (b) two extensive, yet cohesive and well-curated, repositories of add-on packages, known as CRAN (general purpose) and Bioconductor (focused on bioinformatics, [8]). Of special relevance for MD analysis is the Bio3D package [9], which will be part of the side-by-side examples in this chapter.

3.3 Python

Python is a relatively new (first released in 1991) interpreted language for general purpose programming. Its main features is arguably a balance of readability, conciseness and speed; the object-oriented semantics are especially intuitive, and extension modules are easy to import (recently made even simpler with the centralized Conda package manager). The main interest of this language for the MD community is the number of MD-related libraries which are being released: beyond those listed in Table 1 and Note 1, it may be worth mentioning PyEMMA (Markov Model training and testing, [10]), OpenMM (MD engine with GPU acceleration, [11]), MSMBuilder (statistical models for biomolecular dynamics,[12]), and many others. Notable language features are:

- White space is significant, defining indentation-based control blocks.
- Built-in data types include integer, floating point, and associative arrays (hashes). Arbitrary classes can be defined with object-oriented constructs.
- Many notable libraries, of which NumPy (linear algebra) and Pandas (record-based data frames) are especially convenient for trajectory analysis purposes.
- Add-on packages (modules) become visible in the namespace when `import`-ed. The `pip` and `conda` package managers provide automated installations.
- Packages exist to compile compute-intensive portions into native code almost transparently (see Note 3).

4 Useful programming constructs

This section will briefly review how common structured programming constructs and input-output operations are expressed in the language mentioned above, by means of side-by-side parallels. The objective of this comparison is didactic and practical, in order to enable users to easily switch languages.

4.1 Iterations

Listing 1 shows how four common loop idioms can be implemented, namely (a) the common "indexed for" which increments an integer index `i` from 0 (included) to `N` (excluded); (b) iterating over the contents of a list, assigning the element to the variable `x`; (c) iterating two vectors in parallel, which is useful e.g. when Cartesian coordinates are stored separately; (d) iterating at the same time over a list contents as in (b), but also keeping an integer index. Of note, R has the `for(x in vec)` construct, but it is often replaced by implicit vectorization and mapping operators such as `apply`.

TCL

```
for {set i 0} {$i<$N} {incr i}
↪  { ... }

foreach x $vec
↪  { ... }

foreach x $xvec y $yvec
↪  { ... }

set i 0
foreach x $vec { ...; incr i }
```

R

```
for (i in 1:N) { ... }


for (x in vec) { ... }


mapply(function(x,y) { ... },
       xvec, yvec)

# R indexes from 1
for (i in seq_len(vec)) { ... }
```

Python

```
for i in range(N):
    ...

for x in vec:
    ...

for x,y in zip(xvec, yvec):
    ...

for i,x in enumerate(vec):
    ...
```

Listing 1: Four iteration styles: integer index, list contents, parallel lists, list contents plus index.

4.2 File input and output

Listing 2 shows TCL and Python idioms for accessing files in read and write modes. In the case of read, line-based iterations are shown. Note that Python modules are an excellent alternative to parse and create files in common formats; of particular note are comma separated values (via the `csv` standard module); Excel files (via `openpyxl`, `xlrd` and others) and HDF5 (via `h5py`). Furthermore, the `numpy` package can parse text files into numeric matrices (function `genfromtxt()`); and `pandas` reads and writes data frames (PDB-like data structures representing tables with multiple attributes of heterogeneous types) in various formats via its `read_X` and `to_X` methods. Explicit file IO is seldom necessary in R, given the flexibility of its high-level parsing functions (see e.g. `read.table` and `write.table`, the `openxlsx` package, and so on).

TCL

```
set f [open $name r]
set data [read $f]
set lines [split $data "\n"]
close $f

set g [open $name w]
puts $g "Hello world"
close $g
```

Python

```
with open(name,"r") as f:
     # data=f.read()  # Read whole file
     lines=f.readlines()


with open(name,"w") as g:
     g.write("Hello world")
```

Listing 2: File input and output.

4.3 Strings

Listing 3 shows a selection of common string manipulation operators. In addition to the split and join operators (on chosen delimiters), a several other operators are provided as subcommands of the `string` command (TCL), in the `str` module (Python), and the `stringr` package (R).

4.4 Functions

Listing 4 shows how functions are defined in TCL, Python and R. Of note, TCL provides optional arguments with defaults. Both Python and R provide named arguments with defaults. Python functions may uncharacteristically return multiple values at once: `x,y = f()`.

4.5 Arrays and hashes

The nomenclature of data structures differs between languages. For homogeneity I shall use the names array (ordered lists of objects indexed by an integer) and hash (unordered lists indexed by arbitrary objects, also known as associative array). Listing 5 shows typical semantics in the three languages.

```
TCL
set sl [string length $s]

set l [split $s ,]

set s2 [join $l ,]

string range $s 10 20

set v 123
format "%5.2f" $v
```

```
R
sl <- nchar(s)

l <- strsplit(s,",")[[1]]

paste(l,collapse=",")

substr(s,11,21)

v <- 123
sprintf("%5.2f",v)
```

```
Python
sl = len(s)

l = s.split(",")

s2 = ",".join(l)

s[10:21]

v = 123
f"{v:5.2f}"
   "%5.2f" % v        # equivalent
  "{:5.2f}".format(v) # equivalent
```

Listing 3: Basic string operations.

Accessing arrays in Python and R occurs with a square bracket notation, with the caveat that the latter uses 1-based indices. TCL uses list operators such as `lindex` (indexing), `lset` (assignment), `linsert`, `lsort`, etc., all zero-based.

All three languages also support hashes, with slightly different semantics. In particular, TCL provides two hash-like structures, both shown, namely more flexible dictionaries, which can also implement arbitrarily nested data structures, and so-called arrays.

4.6 Algebra

Listing 6 shows basic math and linear algebra constructs. Using the common infix syntax in TCL requires the `expr` function. Common linear algebra operators are part of core R functions; of the `numpy` module in Python; and (to a limited degree) of the `math::linearalgebra` TCL package and VMD's built-in functions such as `vecscale`.

4.7 Exceptions

Finally, Listing 7 shows the common idioms for recovering from errors (catching) or signaling them to the callers (raising).

TCL

```
proc sum {a b} {
    return [expr $a+$b]
}

proc norm {v {n 2}} {
    set s 0.0
    foreach x $v {
        set s [expr $s+$x**$n]
    }
    return [expr $s**(1.0/$n)]
}

norm {3 4}     ;# = 5
norm {3 4} 1   ;# = 7
```

R

```
sum_n <- function(x,y) x+y


norm_n <- function(v, n=2) {
    s <- sum(v**n)
    return(s**(1/n))
}

# Avoid built-in sum, norm


norm_n( c(3,4) )         # = 5
norm_n( c(3,4), 1 )      # = 7
norm_n( n=1, v=c(3,4) )  # also 7
```

Python

```
def sum(x,y):
    return x+y

def norm(v, n=2):
    s=0
    for x in v:
        s+=x**n
    return s**(1/n)

norm([3,4])        # = 5
norm([3,4], 1)     # = 7
norm(n=1, v=[3,4]) # also 7
```

Listing 4: Defining functions. The `norm` function takes the L_n norm of the first argument (a list of floating-point values), with `n` defaulting to 2.

TCL

```
set v {10 20 30}
set v [list 10 20 30];  # equivalent

lindex $v 1
lset v 1 42;            # no £ prefix
llength $v

set m {{1 2} {3 4}}
lindex $m 0 1;          # = 2

set a_dict [dict create beta 1 occ .7]
dict get $a_dict beta;  # 1
dict keys $a_dict;      # beta, occ

array set a_arr {beta 1 occ .7}
puts $a_arr(beta);      # 1
array names a_arr;      # beta, occ
```

R

```
v <- c(10,20,30)


v[2]
v[2] <- 42
length(v)

m <- matrix(c(1,2,3,4),
            byrow=T,ncol=2)
m[1,2]

a=list(beta=1, occ=.7)
names(a)           # beta, occ
a[['beta']]
```

Python

```
v = [10, 20, 30]


v[1]
v[1] = 42
len(v)

m=[[1,2], [3,4]]
m[0][1]             # = 2

import numpy as np
mn = np.array(m)    # more flexibly
mn[0,1]

a = {'beta': 1 ,
     'occ':  .7}
list(a.keys())      # beta, occ
a['beta']
```

Listing 5: Array- and hash-wise manipulation.

TCL

```
# Floating point math requires "expr"
set d [expr sqrt($x**2+$y**2)]


# Expr is implicit in conditionals
if { $x>0 && $y>0 }
↪  { puts "First quadrant" }

# Part of tcllib
package require math::linearalgebra
set m {{1 2} {2 1}}

math::linearalgebra::matmul $m $m
math::linearalgebra::det $m
math::linearalgebra::eigenvectorsSVD $m
# Eig. for symmetric matrices only

package require math::complexnumbers
namespace import math::complexnumbers::*
sqrt [complex -1 0]
exp [complex 0 3.1416]
```

R

```
## Assignment arrow is common
d <- sqrt(x**2+y**2)


if(x>0 && y>0) {
    message("First quadrant")
}


m <- matrix(c(1,2,2,1),
            byrow=T,ncol=2)

m %*% m
det(m)
eigen(m)$values

## Imaginary is postfix i
sqrt(-1+0i)
exp(pi*1i)
```

Python

```
# Import math functions
from math import *
d=sqrt(x**2+y**2)

# Note the non-C-like Boolean operators
if x>0 and y>0:
    print("First quadrant")

# Linear algebra by numpy
m=np.array([[1, 2], [2, 1]])

m @ m       # Matrix product; also
↪  m.dot(m)
np.linalg.det(m)
np.linalg.eig(m)


# Use numpy or cmath for complex maths.
↪  Imaginary unit is postfix j
np.sqrt(-1+0j)
np.exp(pi*1j)
```

Listing 6: Arithmetic and linear algebra.

TCL

```tcl
# To catch
if [catch {dangerous} e] {
    puts "Error caught: $e"
}


# To raise
error "Singularity encountered"
```

R

```r
## To catch
tryCatch(dangerous(),
         error = function(e) {
             message("Error caught:",
             ↪  e)})

## To raise
stop("Singularity encountered")
```

Python

```python
# To catch
try:
    dangerous()
except Exception as e:
    print(f"Error caught: {e}")

# To raise
raise Exception("Singularity")
```

Listing 7: Exception handling.

5 MD analysis libraries

This section will present parallel comparisons for the five MD analysis libraries listed in Table 1. Each of the packages contains extensive reference material and examples (see Note 5 for pointers). I won't discuss installation procedures, found in the available documentation, but only remark that the TCL interpreter is embedded in the VMD software (accessible under "Extensions/Tk Console"); that Bio3D is available through R's `install.packages("bio3d")` call; and that Python-based libraries can be easily installed via the Conda package manager (see Note 6).

5.1 VMD

VMD [6] is one of the most widely used software packages for MD visualization and analysis (it also include modeling facilities). Its main strength is that it deals well with large systems (of the order of millions of atoms) and/or very long trajectories (millions of frames). Of note, VMD has a plug-in system [13, 14, 15, 16], which allows graphical interfaces to be developed with Tcl/Tk, an unusually programmer-friendly GUI toolkit.

5.2 Bio3D

Bio3D [9] is an R package for comparative analysis of protein structures. It is notable for its integration with the R statistical environment and object model, which facilitates interoperability with the large array of statistical methods implemented in CRAN packages, and the fact that it provides methods for analysis on the basis of sequence alignments (multiple-PDB objects).

5.3 MDAnalysis

MDAnalysis [17, 18] is an object-oriented Python library for the processing of MD trajectories. Notable features are its "streaming" design enabling larger-than-memory processing, methods dedicated to lipid bilayers identification, and an object model for set-oriented manipulation of atom selections.

5.4 MDTraj

MDTraj [19] is a Python library dedicated to the manipulation of MD trajectories, with an eye to the integration with external packages. Of note, the library is well integrated with the OpenMM GPU-accelerated simulation engine [11].

5.5 High-Throughput Molecular Dynamics (HTMD)

HTMD [20] is a Python-based environment integrating facilities for MD analysis, system preparation [21], building [22], ligand parameterization, and simulation (with the included ACEMD engine [23]). Of note, HTMD suits well the analysis of multiple independent trajectories ("high-throughput") *via* Markov-state model analysis.

6 Examples of trajectory analysis constructs

To provide a concrete example, I demonstrate side-by-side how a simple but realistic analysis task is implemented in the various analysis libraries. It is applied to a publicly-available trajectory containing 40 ns of constant-pressure simulation of the acid sensing ion channel (ASIC) 1 trimer [24] embedded in a POPC membrane, retrieved from the PlayMolecule membrane protein repository [22].

The comparison is restricted to the basic features that could be reasonably compared side-by-side. They constitute the "least common denominator" of MD processing: each of the libraries has far more advanced capabilities, to whose documentation readers are referred. Finally, note that there are differences in the physical units returned.

The code blocks provided in the next subsections build on each other and are meant to be executed in order. Note 7 provides code to initialize the `pdb` and `xtc` file name variables and download the data files.

6.1 Loading trajectories

The first step of analysis is to load trajectories into memory (Listing 8). This generally requires supplying both a topology file, containing atom types and residue information, and a binary trajectory file, containing the coordinates taken at regular intervals during the simulation. Note that usually the number of atoms is assumed constant throughout the simulation (this also a limitation of common MD trajectory formats). Note that MDAnalysis does not hold the whole trajectory in memory but rather it updates the associated objects while iterating; see Note 8 for coding implications.

Once a system is loaded, a representation of the topology (or at least the main fields) is built in-memory and can be queried. Depending on the language, a number of attributes provide object-oriented access to residues' and atoms' properties (Table 2). Some libraries (depending on the simulation software) also provide access to unit cell dimensions and physical time between frames.

Table 2 around here

6.2 Frame selection

Listing 9 shows the syntax to retrieve the number of atoms (system size), the length of a trajectory, and the actual values of coordinates. Coordinates are usually stored as the underlying language's matrix objects. Later sections will show how "slice" operators extract of a subset of atoms or frames.

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

Bio3D
```r
library(bio3d)
tp <- read.pdb(pdb)
tp$xyz <- read.dcd(dcd)
```

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

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

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

Listing 8: Loading topologies and trajectories. R code uses a different variable name not to overwrite the built-in transpose operator `t()`.

VMD
```tcl
# 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
```

Bio3D
```r
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))
```

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

# Atoms by 3
t.atoms.positions

# Unit cell
t.atoms.dimensions
```

MDTraj
```python
# 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,:]
```

HTMD
```python
t.numFrames
t.numAtoms

# Atoms by 3 by frames
t.coords

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

Listing 9: Accessing system sizes and coordinates.

6.3 Atom selection

The ability to select atoms on the base of their characteristics (identifiers, residues numbers, or chemical properties) is central to analysis. Most libraries implement atom selection languages (ASL), strings which can be applied to a trajectory frame and ultimately evaluate to a boolean value per each atom, indicating whether the selection includes the atom or not (Listing 10). For system-specific examples, let's show how to extract the O_η atom of ASIC1's Y72 residue, and the four atoms comprising the χ_1 dihedral of W288 (involved in acid-dependent gating [25]).

It is important to note some variations in the ASL syntaxes, summarized in Table 3. Of note, atom selection objects (returned by the `atomselect` command) are central in VMD, as they are used to read and modify most of a system's properties; its extensive ASL has keywords that select on primary sequence, PDB fields, steric context, geometry, polarity and so on.

VMD

```
set y72_oeta [atomselect top "resid 72
↪  and name OH and chain 0"]
set w288_chi1 [atomselect top "resid 288
↪  and name N CA CB CG and chain 0"]

# Access the "occupancy" property
# of a single atom
$y72_oeta get occupancy
```

MDAnalysis

```
y72_oeta = t.select_atoms("resid 72 and
↪  name OH and segid 0")
w288_chi1 = t.select_atoms("resid 288
↪  and name N CA CB CG and segid 0")

y72_oeta.occupancies
y72_oeta[0].occupancy   # also
```

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]
```

Bio3D

```
pdb <- tp$atom
y72_oeta <- pdb[pdb$resno == 72   &
                pdb$elety == "OH" &
                pdb$chain == "0" , ]

w288_chi1 <- atom.select(tp,
         elety=c("N","CA","CB","CG"),
         resno=288, chain="0")

y72_oeta$o
```

MDTraj

```
y72_oeta = t.topology.select("residue 72
↪  and name OH and chainid 0")
w288_chi1 = t.topology.select("residue
↪  288 and name N CA CB CG and chainid
↪  0")

t.atom_slice(y72_oeta).topology.\
    atom(0).element
```

Listing 10: Selection of Tyr72's O_η atom and the four atoms defining the Trp288's χ_1 dihedral in the first protein subunit *via* atom selection languages.

Table 3 around here

6.4 Filtering and writing

It is often useful to filter out atoms not of interest (say, water molecules) either to speed-up calculations, or to produce input files for further programs (e.g., docking software). Listing 11 shows as an example the syntax used for filtering the trimer's backbone atoms and writing the first frame to a PDB file.

VMD

```
set bb [atomselect top backbone]

# Write backbone frame 0
animate write pdb bb_frame0.pdb
↪  beg 0 end 0 sel $bb
```

MDAnalysis

```
bb = t.select_atoms("backbone")

with mda.Writer("bb_frame0.pdb") as w:
    t.trajectory[0]
    w.write(bb)

# Also bb.write() for single frames
```

HTMD

```
bb = t.copy()
bb.filter("backbone")
bb.dropFrames(keep=0)
bb.write("bb_frame0.pdb")
```

Bio3D

```
bb <- atom.select(tp, "backbone")
tp_bb <- trim(tp, bb)

## Select frame 1 (i.e. 0) only
bb_ref <- tp_bb
bb_ref$xyz <- trim(bb_ref$xyz, 1)

write.pdb(bb_ref, "bb_frame0.pdb")
```

MDTraj

```
bb = t.topology.select("backbone")
t_bb = t.atom_slice(bb)   # Subset

# Select frame 0 (also [0])
bb_ref = t_bb.slice(0)

# Write to file
bb_ref.save("bb_frame0.pdb")
```

Listing 11: Trajectory filtering and writing.

6.5 Basic geometry

Once coordinates are extracted from the trajectory object, they can be manipulated with the language's native operators. All libraries provide operators to compute derived quantities such as distances, angles, torsions, hydrogen bonds, surface-accessible areas, contacts, etc. As noted above, the analysis features of each library are extensive, and even a partial list would be prohibitively long.

Listing 12 shows the programming style followed in each system to compute two representative quantities, i.e. (a) the center of mass and (b) the W288 χ_1 dihedral, either for a single frame or over the whole trajectory. Care must be taken to check whether operators account for periodic boundary conditions.

6.6 Alignment and RMSD

Minimum-root mean square deviation (RMSD) alignments are a frequent operation in MD analysis, enabling the geometrical comparison between selected

```
VMD
# Center of mass
$bb frame 0
measure center $bb weight mass

# W288, chi 1, first frame
measure dihed [$w288_chi1 get index]

# All frames
measure dihed [$w288_chi1 get index]
↪  first 0 last 40
```

```
MDAnalysis
# Self-explanatory
bb.center_of_mass()

# Current frame
w288_chi1.dihedral.value()

# All frames (iterator)
[w288_chi1.dihedral.value()
   for f in t.trajectory]
```

```
HTMD
# Center of geometry: add
↪  "weights=bb.masses" if available in
↪  topology
np.average(bb.coords, axis=0).T

# First frame
htmd.molecule.util.dihedralAngle(
             t.coords[w288_chi1,:,0])

# All frames
htmd.molecule.util.dihedralAngle(
             t.coords[w288_chi1,:,:])
```

```
Bio3D
# Center of mass
com(tp_bb)

# Torsion, first frame
tmp <- tp$xyz[1, w288_chi1$xyz]
torsion.xyz(c(t(tmp)))

# All frames (reshape as 1D vector)
tmp <- tp$xyz[ , w288_chi1$xyz]
torsion.xyz(c(t(tmp)))
```

```
MDTraj
# Self-explanatory
mdt.compute_center_of_mass(bb_ref)

# First frame (to degrees)
mdt.compute_dihedrals(t[0],
    [w288_chi1])*180.0/np.pi

# All frames
mdt.compute_dihedrals(t,
    [w288_chi1])*180.0/np.pi
```

Listing 12: Geometry operations.

portions of two structures. The typical formulation proceeds in three steps: (a) it searches for the proper rigid transformation that minimizes the RMSD distance between a set of *alignment* atoms of a trajectory and the corresponding set in a reference configuration (superposition step); (b) it applies the transformation to the trajectory (alignment); and (c) optionally returns the root-mean square of the distances between a set of aligned *measurement* (or displacement) atoms and the corresponding ones in the reference (RMSD computation proper; see Note 9).

Listing 13 summarizes how the steps can be performed; the alignment and RMSD computation steps are coded separately to make them explicit. Once the alignment is performed, the modified trajectory can be further processed or saved back as seen in Section 6.4 (or with the `AlignTraj` method in the case of MDAnalysis). VMD code uses the `rmsdOf` convenience function for readability (see Note 10).

7 Conclusion

A wealth of libraries has been developed to ease structural biology-oriented manipulation of MD trajectories with general-purpose programming languages. This chapter tried to provide MD users – and beginning students in particular – with a "Rosetta stone" showing languages and constructs side-by-side. It is also hoped that this effort promotes the integration of methods and interchange of data between MD communities.

8 Notes

1. The number of libraries dealing with aspects of MD analysis is extensive. Table 4 provides a partial list of packages further to the ones examined in this chapter. The selection made in this review is therefore to a large degree arbitrary, and the balance may change as technologies evolve. I apologize for the necessary omissions.

   Table 4 around here

2. Some tension exists between tackling analysis tasks in a fully general purpose environment, *versus* using simplified "shells" optimized for specific MD-related analysis operations. The advantage of the former approach is its generality, as the algorithm may use the same spectrum of operations allowed to native code; however, general-purpose compiled languages (usually Fortran, C and C++) are hard to master, arguably error-prone, and make for verbose source codes. Structural and trajectory manipulation libraries make MD-related operations somewhat simpler and more robust, but programming challenges remain. Conversely, interactive MD-specific applications (either graphical or command-line) restrict the analysis tasks to the domain-specific ones which have been pre-programmed.

VMD

```
# Convenience functions from
# github.com/tonigi/vmd_extensions
source ~/VMDextensions.tcl

# t: trajectory; r: reference;
# alg: alignment; meas: measurement
set meas_t [atomselect top protein]
set meas_r [atomselect top protein
            frame 0]
set alg_t [atomselect top backbone]
set alg_r [atomselect top backbone
           frame 0]

set rmsd_traj [rmsdOf $meas_t $meas_r
                      $alg_t  $alg_r]
```

MDAnalysis

```
from MDAnalysis.analysis.rms import RMSD
R = RMSD(atomgroup=t,
         reference=t,
         select="backbone",  # align set
         groupselections=["protein"])
R.run()

# Measures found in column 4 and on
rmsd_traj = R.rmsd[:,3]
```

HTMD

```
meas_r = t.copy()
meas_r.dropFrames(keep=0)
meas_t = t.copy()

meas_t.align("backbone",meas_r)

meas_set = meas_t.atomselect("protein")
rmsd_traj = htmd.molecule.util.molRMSD(
    meas_t,meas_r,meas_set,meas_set)
```

Bio3D

```
meas.r <- trim(tp$xyz,1)
meas.t <- tp$xyz
alg.set <- bb$xyz

fitted <- fit.xyz(fixed = meas.r,
                  mobile = meas.t,
                  fixed.inds = alg.set,
                  mobile.inds = alg.set)

meas.set <- atom.select(tp,
                        "protein")$xyz
rmsd.traj <- rmsd(a = meas.r,
                  b = fitted,
                  a.inds = meas.set,
                  b.inds = meas.set)
```

MDTraj

```
fitted = t[:]             # Copy

# Align
alg_set = t.topology.select("backbone")
fitted.superpose(reference=t,
                 frame=0,
                 atom_indices=alg_set)

meas_set = t.topology.select("protein")
meas_r = t.atom_slice(meas_set)
meas_t = fitted.atom_slice(meas_set)

d = meas_t.xyz-meas_r.xyz[0]
rmsd_traj = 10 * np.sqrt( np.mean(
        np.sum(d**2,axis=2),axis=1))
```

Listing 13: RMSD-based alignments.

The dilemma is to a large extent solved by high-level interpreted scripting languages such as the ones examined in this chapter, which provide access to a wide variety of libraries, terse syntaxes, and fast prototype-execute cycles. Unsurprisingly, interpreters for scripting languages are now embedded in most molecular analysis environments.

3. Python variables *can* indeed be strongly typed when the interpreter is used in combination with packages such as Cython [26] or Numba [27]. Both packages transparently compile Python code, annotated with static types, into optimized native code.

4. The detailed understanding of variable visibility and namespace partitioning rules is of particular relevance for development projects more complex than one-off scripts. Although outside of the scope of this chapter, mastering them is highly desirable because of the increase in productivity and code quality it affords.

5. Some resources providing realistic worked-out examples:
   VMD – VMD's Tutorials, available at `www.ks.uiuc.edu/Training/Tutorials`.
   Bio3D – Executable demos (`pdb`, `md` and `pca`) and tutorials (called *vignettes* in the context of R), installed with the package and also available at `thegrantlab.org/bio3d`.
   MDAnalysis – The *tutorial* section available at `mdanalysis.org`.
   MDTraj – The *examples* section available at `mdtraj.org` in the form of IPython notebooks.
   HTMD – The *Introduction to HTMD* section of the User Guide, at `htmd.org`.

6. Each package's documentation specifies in which *Conda channel* it is found. Many contributed packages, including MDTraj and MDAnalysis, are found in the `conda-forge` channel, which provides an automated building and distribution pipeline. The typical command line is `conda install -c <channel> <packagename>[=<version>]`.

7. Listing 14 provides scripts which download the data files used in this tutorial and set the corresponding file name variables. The files are simulation trajectories of the acid sensing ion channel 1 trimer (PDB: 2QTS [24]), embedded in a POPC membrane located by the OPS algorithm [28] and simulated for 40 ns with the CHARMM36 forcefield [29]. They are available from the PlayMolecule repository of pre-equilibrated OPM membrane proteins [22].

8. MDAnalysis' *out of core* design enables the analysis of trajectories much larger than the memory physically available; however, care must be taken because iterating over frames in a `Universe` changes the objects derived from it. This is evident e.g. in Listing 11 (MDAnalysis panel), where access to trajectory frame – 0 in the example – also updates the `bb` object about to be written. (The same occurs in Listing 12 for the `w288_chi1`

VMD/TCL

```tcl
set code 2qts
set url "http://www.playmolecule.org/static/apps/OPM/data/$code/equil_charmm/"
set pdb structure.filtered.pdb
set xtc traj.filtered.xtc

vmd_mol_urlload $url/$pdb $pdb
vmd_mol_urlload $url/$xtc $xtc
```

R

```r
code <- "2qts"
url <- sprintf("http://www.playmolecule.org/static/apps/OPM/data/%s/equil_charmm/", code)
pdb <- "structure.filtered.pdb"
xtc <- "traj.filtered.xtc"
dcd <- "traj.filtered.dcd"

download.file(file.path(url, pdb), pdb)
download.file(file.path(url, xtc), xtc)

## Convert to DCD format for Bio3D
system(sprintf("catdcd -o %s -xtc %s", dcd, xtc))
```

Python

```python
code = "2qts"
url = f"http://www.playmolecule.org/static/apps/OPM/data/{code}/equil_charmm/"
pdb = "structure.filtered.pdb"
xtc = "traj.filtered.xtc"

import numpy as np
from urllib.request import urlretrieve
urlretrieve(url+pdb, pdb)
urlretrieve(url+xtc, xtc)
```

Listing 14: Initialization code.

object.) This is perhaps less surprising recalling that in general Python objects are references, not values.

9. The term “RMSD calculation” may be ambiguous; in particular, it may indicate the result of the calculation either before or after applying the optimal-rotation operator. This chapter, for the sake of clarity and generality, presents two-step procedures in which the alignment and (unaligned) RMSD calculation steps are explicitly separated. In some software packages, “RMSD” functions perform the alignment implicitly.

10. The `rmsdOf` function is a shorthand operator part of the *VMD Extensions Functions* library, available at `tonigi.github.io/vmd_extensions`.

Acknowledgments

I gratefully acknowledge the constructive comments received from Prof. B. Grant, Prof. O. Beckstein, M. Linke, and J. Barnoud. I also acknowledge Acellera Ltd. and the Dipartimento di Medicina e Chirurgia of the Università degli Studi dell’Insubria for funding, as well as CINECA awards under the ISCRA initiative for the availability of high performance computing resources and support.

References

[1] Pronk, S., Páll, S., Schulz, R., Larsson, P., Bjelkmar, P., Apostolov, R., Shirts, M.R., Smith, J.C., Kasson, P.M., Spoel, D.v.d., Hess, B., Lindahl, E.: GROMACS 4.5: a high-throughput and highly parallel open source molecular simulation toolkit. Bioinformatics **29**(7), 845–854 (2013). DOI 10.1093/bioinformatics/btt055

[2] Roe, D.R., Cheatham, T.E.: PTRAJ and CPPTRAJ: Software for Processing and Analysis of Molecular Dynamics Trajectory Data. Journal of Chemical Theory and Computation **9**(7), 3084–3095 (2013). DOI 10.1021/ct400341p

[3] Tribello, G.A., Bonomi, M., Branduardi, D., Camilloni, C., Bussi, G.: Plumed 2: New feathers for an old bird. Computer Physics Communications **185**(2), 604–613 (2014). DOI 10.1016/j.cpc.2013.09.018

[4] Giorgino, T.: How to differentiate collective variables in free energy codes: Computer-algebra code generation and automatic differentiation. Computer Physics Communications **228**, 258–263 (2018). DOI 10.1016/j.cpc.2018.02.017. 2-s2.0-85043302064

[5] Giorgino, T.: PLUMED-GUI: An environment for the interactive development of molecular dynamics analysis and biasing scripts. Computer Physics Communications **185**(3), 1109–1114 (2014). DOI 10.1016/j.cpc.2013.11.019

[6] Humphrey, W., Dalke, A., Schulten, K.: VMD: Visual molecular dynamics. Journal of Molecular Graphics **14**(1), 33–38 (1996). DOI 10.1016/0263-7855(96)00018-5

[7] R Development Core Team: R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria (2008). ISBN 3-900051-07-0

[8] Huber, W., Carey, V.J., Gentleman, R., Anders, S., Carlson, M., Carvalho, B.S., Bravo, H.C., Davis, S., Gatto, L., Girke, T., Gottardo, R., Hahne, F., Hansen, K.D., Irizarry, R.A., Lawrence, M., Love, M.I., MacDonald, J., Obenchain, V., Oleś, A.K., Pagès, H., Reyes, A., Shannon, P., Smyth, G.K., Tenenbaum, D., Waldron, L., Morgan, M.: Orchestrating high-throughput genomic analysis with Bioconductor. Nature Methods **12**(2), 115–121 (2015). DOI 10.1038/nmeth.3252

[9] Grant, B.J., Rodrigues, A.P.C., ElSawy, K.M., McCammon, J.A., Caves, L.S.D.: Bio3d: an R package for the comparative analysis of protein structures. Bioinformatics **22**(21), 2695–2696 (2006). DOI 10.1093/bioinformatics/btl461

[10] Scherer, M.K., Trendelkamp-Schroer, B., Paul, F., Pérez-Hernández, G., Hoffmann, M., Plattner, N., Wehmeyer, C., Prinz, J.H., Noé, F.: PyEMMA 2: A Software Package for Estimation, Validation, and Analysis of Markov Models. Journal of Chemical Theory and Computation **11**(11), 5525–5542 (2015). DOI 10.1021/acs.jctc.5b00743

[11] Eastman, P., Swails, J., Chodera, J.D., McGibbon, R.T., Zhao, Y., Beauchamp, K.A., Wang, L.P., Simmonett, A.C., Harrigan, M.P., Stern, C.D., Wiewiora, R.P., Brooks, B.R., Pande, V.S.: OpenMM 7: Rapid development of high performance algorithms for molecular dynamics. PLOS Computational Biology **13**(7), e1005,659 (2017). DOI 10.1371/journal.pcbi.1005659

[12] Harrigan, M.P., Sultan, M.M., Hernández, C.X., Husic, B.E., Eastman, P., Schwantes, C.R., Beauchamp, K.A., McGibbon, R.T., Pande, V.S.: MSMBuilder: Statistical Models for Biomolecular Dynamics. Biophysical Journal **112**(1), 10–15 (2017). DOI 10.1016/j.bpj.2016.10.042

[13] Giorgino, T.: Computing 1-D atomic densities in macromolecular simulations: The density profile tool for VMD. Computer Physics Communications **185**(1), 317–322 (2014). DOI 10.1016/j.cpc.2013.08.022

[14] Fernandes, H.S., Ramos, M.J., Cerqueira, N.M.F.S.A.: molUP: A VMD plugin to handle QM and ONIOM calculations using the gaussian software. Journal of Computational Chemistry **39**(19), 1344–1353 (2018). DOI 10.1002/jcc.25189

[15] Giorgino, T., Laio, A., Rodriguez, A.: METAGUI 3: A graphical user interface for choosing the collective variables in molecular dynamics simulations. Computer Physics Communications **217**, 204–209 (2017). DOI 10.1016/j.cpc.2017.04.009

[16] Guixà-González, R., Rodriguez-Espigares, I., Ramírez-Anguita, J.M., Carrió-Gaspar, P., Martinez-Seara, H., Giorgino, T., Selent, J.: MEMBPLUGIN: studying membrane complexity in VMD. Bioinformatics **30**(10), 1478–1480 (2014). DOI 10.1093/bioinformatics/btu037

[17] Gowers, R.J., Linke, M., Barnoud, J., Reddy, T.J.E., Melo, M.N., Seyler, S.L., Domański, J., Dotson, D.L., Buchoux, S., Kenney, I.M., Beckstein, O.: MDAnalysis: A Python Package for the Rapid Analysis of Molecular Dynamics Simulations. pp. 98–105 (2016)

[18] Michaud-Agrawal, N., Denning, E.J., Woolf, T.B., Beckstein, O.: MDAnalysis: a toolkit for the analysis of molecular dynamics simulations. Journal of Computational Chemistry **32**(10), 2319–2327 (2011). DOI 10.1002/jcc.21787

[19] McGibbon, R.T., Beauchamp, K.A., Harrigan, M.P., Klein, C., Swails, J.M., Hernández, C.X., Schwantes, C.R., Wang, L.P., Lane, T.J., Pande, V.S.: MDTraj: A Modern Open Library for the Analysis of Molecular Dynamics Trajectories. Biophysical Journal **109**(8), 1528–1532 (2015). DOI 10.1016/j.bpj.2015.08.015

[20] Doerr, S., Harvey, M.J., Noé, F., De Fabritiis, G.: HTMD: High-Throughput Molecular Dynamics for Molecular Discovery. Journal of Chemical Theory and Computation **12**(4), 1845–1852 (2016). DOI 10.1021/acs.jctc.6b00049

[21] Martinez-Rosell, G., Giorgino, T., De Fabritiis, G.: PlayMolecule ProteinPrepare: A Web Application for Protein Preparation for Molecular Dynamics Simulations. Journal of Chemical Information and Modeling (2017). DOI 10.1021/acs.jcim.7b00190

[22] Doerr, S., Giorgino, T., Martinez-Rosell, G., Damas, J.M., De Fabritiis, G.: High-throughput automated preparation and simulation of membrane proteins with HTMD. Journal of Chemical Theory and Computation (2017). DOI 10.1021/acs.jctc.7b00480

[23] Harvey, M.J., Giupponi, G., De Fabritiis, G.: ACEMD: Accelerating Biomolecular Dynamics in the Microsecond Time Scale. Journal of Chemical Theory and Computation **5**(6), 1632–1639 (2009). DOI 10.1021/ct9000685

[24] Jasti, J., Furukawa, H., Gonzales, E.B., Gouaux, E.: Structure of acid-sensing ion channel 1 at 1.9 Å resolution and low pH. Nature **449**(7160), 316–323 (2007). DOI 10.1038/nature06163

[25] Sherwood, T.W., Frey, E.N., Askwith, C.C.: Structure and activity of the acid-sensing ion channels. American Journal of Physiology - Cell Physiology **303**(7), C699–C710 (2012). DOI 10.1152/ajpcell.00188.2012

[26] Behnel, S., Bradshaw, R., Citro, C., Dalcin, L., Seljebotn, D., Smith, K.: Cython: The best of both worlds. Computing in Science Engineering **13**(2), 31 –39 (2011). DOI 10.1109/MCSE.2010.118

[27] Lam, S.K., Pitrou, A., Seibert, S.: Numba: A LLVM-based Python JIT Compiler. In: Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC, LLVM '15, pp. 7:1–7:6. ACM, New York, NY, USA (2015). DOI 10.1145/2833157.2833162

[28] Lomize, M.A., Pogozheva, I.D., Joo, H., Mosberg, H.I., Lomize, A.L.: OPM database and PPM web server: resources for positioning of proteins in membranes. Nucleic Acids Research **40**(Database issue), D370–376 (2012). DOI 10.1093/nar/gkr703

[29] Best, R.B., Zhu, X., Shim, J., Lopes, P.E.M., Mittal, J., Feig, M., MacKerell, A.D.: Optimization of the Additive CHARMM All-Atom Protein Force Field Targeting Improved Sampling of the Backbone ϕ, ψ and Side-Chain $\chi1$ and $\chi2$ Dihedral Angles. Journal of Chemical Theory and Computation **8**(9), 3257–3273 (2012). DOI 10.1021/ct300400x

[30] Sanner, M.F.: Python: a programming language for software integration and development. Journal of Molecular Graphics & Modelling **17**(1), 57–61 (1999). WOS:000084162500006

[31] Schrödinger, LLC: The PyMOL molecular graphics system, version 1.8 (2015)

[32] Hildebrandt, A., Dehof, A.K., Rurainski, A., Bertsch, A., Schumann, M., Toussaint, N.C., Moll, A., Stöckel, D., Nickels, S., Mueller, S.C., Lenhof, H.P., Kohlbacher, O.: BALL - biochemical algorithms library 1.3. BMC Bioinformatics **11**, 531 (2010). DOI 10.1186/1471-2105-11-531

[33] Hinsen, K.: The molecular modeling toolkit: A new approach to molecular simulations. Journal of Computational Chemistry **21**(2), 79–85 (2000)

[34] Pettersen, E.F., Goddard, T.D., Huang, C.C., Couch, G.S., Greenblatt, D.M., Meng, E.C., Ferrin, T.E.: UCSF Chimera–a visualization system for exploratory research and analysis. Journal of Computational Chemistry **25**(13), 1605–1612 (2004). DOI 10.1002/jcc.20084

[35] Grünberg, R., Nilges, M., Leckner, J.: Biskit—A software platform for structural bioinformatics. Bioinformatics **23**(6), 769–770 (2007). DOI 10.1093/bioinformatics/btl655

[36] Romo, T.D., Grossfield, A.: LOOS: An extensible platform for the structural analysis of simulations. In: 2009 Annual International Conference of the IEEE Engineering in Medicine and Biology Society, pp. 2332–2335 (2009). DOI 10.1109/IEMBS.2009.5335065

[37] Cock, P.J.A., Antao, T., Chang, J.T., Chapman, B.A., Cox, C.J., Dalke, A., Friedberg, I., Hamelryck, T., Kauff, F., Wilczynski, B., Hoon, D., L, M.J.: Biopython: freely available Python tools for computational molecular biology and bioinformatics. Bioinformatics **25**(11), 1422–1423 (2009). DOI 10.1093/bioinformatics/btp163

[38] Biasini, M., Mariani, V., Haas, J., Scheuber, S., Schenk, A.D., Schwede, T., Philippsen, A.: OpenStructure: a flexible software framework for computational structural biology. Bioinformatics **26**(20), 2626–2628 (2010). DOI 10.1093/bioinformatics/btq481

[39] Bakan, A., Meireles, L.M., Bahar, I.: ProDy: Protein Dynamics Inferred from Theory and Experiments. Bioinformatics **27**(11), 1575–1577 (2011). DOI 10.1093/bioinformatics/btr168

[40] Münz, M., Biggin, P.C.: JGromacs: a Java package for analyzing protein simulations. Journal of Chemical Information and Modeling **52**(1), 255–259 (2012). DOI 10.1021/ci200289s

[41] Hirsh, L., Piovesan, D., Giollo, M., Ferrari, C., Tosatto, S.C.E.: The Victor C++ library for protein representation and advanced manipulation. Bioinformatics **31**(7), 1138–1140 (2015). DOI 10.1093/bioinformatics/btu773

[42] Yesylevskyy, S.O.: Pteros 2.0: Evolution of the fast parallel molecular analysis library for C++ and Python. Journal of Computational Chemistry **36**(19), 1480–1488 (2015). DOI 10.1002/jcc.23943

Tables

Software	**Version**	**Language**	**Reference**	**Pub. date**	**URL**
VMD	1.9.3	TCL	[6]	1996	`www.ks.uiuc.edu/Research/vmd`
Bio3D	2.3	R	[9]	2006	`thegrantlab.org/bio3d`
MDAnalysis	0.17.0	Python	[17]	2011	`www.mdanalysis.org`
MDTraj	1.9.1	Python	[19]	2015	`www.mdtraj.org`
HTMD	1.14	Python	[20]	2016	`www.htmd.org`

Table 1: Libraries presented in this chapter (sorted by first publication date). Python-based ones were used with Python version 3.6.5, from the Conda distribution of Anaconda, Inc.

PDB field	**VMD, HTMD**	**Bio3D**	**MDAnalysis***	**MDTraj**	**Description (PDB 3.3 standard)**
`ATOM`		type			Record name.
serial	serial	eleno		A.serial	Atom serial number.
name	name	elety	names	A.name	Atom name.
altLoc	altloc	alt	altLocs		Alternate location indicator.
resName	resname	resid	resnames	R.name	Residue name.
chainID	chain	chain		R.chain.index	Chain identifier.
resSeq	resid	resno	resids	R.resSeq	Residue sequence number.
iCode	insertion	insert	icodes		Code for insertion of residues.
x	x	x			Orthogonal coordinates for X in Angstroms.
y	y	y			Orthogonal coordinates for Y in Angstroms.
z	z	z			Orthogonal coordinates for Z in Angstroms.
occupancy	occupancy	o	occupancies		Occupancy.
tempFactor	beta	b	tempfactors		Temperature factor.
segID†	segname	segid	segids	R.segment_id	Segment identifier.
element	element	elesy		A.element	Element symbol.
charge	charge	charge			Charge on the atom.

Table 2: Approximate correspondence between fields in PDB, VMD atom selection objects, Bio3D's `atom` data frame, properties of MDAnalysis' *AtomGroup* objects and MDTraj object model properties (R: an instance of a *Residue* object; A: an instance of an *Atom* object). Notes: (*) Equivalent properties are also present in *Residue* and *Atom* instances. (†) No longer part of the PDB 3.3 format version, but used e.g. for defining molecules in system building.

VMD-like*	**MDTraj**	**Description**
name	name	Atom name
index	index	Atom index (0-based)
mass	mass	Element atomic mass (Dalton)
resname	resname	Three-letter residue code
residue	resid	Residue index (0-based)
resid	residue	Residue sequence record
	rescode	One-letter residue code
element	type	Chemical symbol
type		Forcefield atom type
	chainid	Chain index (0-based)
chain		Chain identifier
segid	segment_ id	Segment identifier

Table 3: Correspondences between keywords in the atom selection languages. Computed attributes (e.g. "backbone") are omitted. Note: (*) VMD, MDAnalysis and HTMD.

Name	**Language**	**Pub. date**	**Reference**
MGLTools/PMV	Python	1999	[30]
PyMOL	Python	ca. 2000	[31]
BALL	C++	2000	[32]
MMTK	Python	2000	[33]
UCSF Chimera	Python	2003	[34]
Biskit	Python	2007	[35]
LOOS	C++	2009	[36]
BioPython	Python	2009	[37]
OpenStructure	C++, Python	2010	[38]
ProDy	Python	2011	[39]
JGromacs	Java	2012	[40]
Victor	C++	2015	[41]
Pteros	C++, Python	2015	[42]

Table 4: A selection of MD-oriented analysis libraries and toolkits.