MolTwister

Release 1.4.0

Richard Olsen

Sep 13, 2023

CONTENTS

CHAPTER

ONE

ABOUT MOLTWISTER

MolTwister is an open source software package. It can aid in the construction of molecular systems used as input to molecular dynamics simulations. However, its primary purpose is to provide a collection of analysis tools for postprocessing of trajectories and data from molecular dynamics simulations. This includes calculation of density profiles, velocity auto correlation functions, radial distribution functions, dihedral distributions and more.

The software runs in a command line window and will act as an extension to the ordinary command line window, running in the background. Hence, an additional set of commands are available for creating and analysing molecular systems from the command line. Commands not recognized by MolTwister is sent for processing as a regular command (e.g., vim can be executed directly from within MolTwister). In addition to the extended command line, a 3D View window will display the molecular system being edited or created.

The command structure in MolTwister follows a simple structure. In the source code, each command is implemented as its own class that inherits from a base command class. A similar structure is also possible with sub-commands. From within the command class, access is given to the entrire state of the molecular system, including atomic positions, masses and force-field parameters. Hence, to implement a new command, copy a similar command class to the one that is to be created, rename it, register it in the command pool and implement its behavior. The class also contains the command documentation, thus providing a complete command implementation.

CHAPTER

TWO

HOW TO CITE MOLTWISTER

MolTwister can be cited using the following article:

```
Olsen, R. (2023), Comp. Phys. Commun., 291, pp. 108822. doi: 10.1016/j.cpc.2023.108822.
```

If using the charge balancing algorithm, which is invoked by a call to the `calculate qbal` command, it is possible to cite the paper where this procedure is introduced:

```
Olsen, R. et al. (2016), J. Phys. Chem. C, 120(51), pp. 29264-29271. doi: 10.1021/acs.
↪jpcc.6b10043.
```

CHAPTER

THREE

HOW TO INSTALL MOLTWISTER

3.1 Obtaining MolTwister

The first step is to download MolTwister. Goto the MolTwister homepage and select Download. You will be directed to the GitHub repository of MolTwister. The download can be done in two ways

1. Perform a git clone of the repository
2. Download the zip file of the repository

3.2 Git

Depending on the method used to download MolTwister, it may be necessary to install Git. On Ubuntu this is done by

```
sudo apt install git
```

On MacOS, Git can be installed using binary installers or through repositories, such as Homebrew. For example,

```
brew install git
```

3.3 GCC, G++ and CLang

MolTwister is written in C++. Hence, if GCC and G++ has not been installed, they must be installed. On Ubuntu, this can be done by

```
sudo apt install gcc
sudo apt install g++
```

On MacOS, MolTwister will be compiled using CLang. If CLang is not installed. make sure it is installed. This can be done by installing XCode.

3.4 CMake

To compile MolTwister, version 3.11, or greater, of CMake must be installed. On ubuntu, this can be done by

```
sudo apt install cmake
```

It is also possible to simplify the compilation process by installing cmake-gui:

```
sudo apt install cmake-gui
```

3.5 The GNU Readline Library

MolTwister uses the GNU Readline Library. The source code and binary files, in addition to installation instructions, are available at The GNU Readline Library home page. However, it is also possible (e.g., on Ubuntu) to install the development version through a repository:

```
sudo apt install libreadline-dev
```

The library is also avialable on repositories for MacOS and can be installed using Homebrew or MacPorts. For example,

```
brew install readline
```

3.6 Python/C

Python/C comes with Python and is used by MolTwister. On Ubuntu, this can be installed by

```
sudo apt install python3
sudo apt install libpython3-dev
```

Note that the Python3 package usually comes pre-installed on newer versions of Ubuntu. For MacOS, the Python3 package can be downloaded from python.org. It may also be necessary to install XCode to obtain the appropriate development libraries to compile Python/C specific code.

3.7 OpenGL

MolTwister uses OpenGL. Hence, the OpenGL development packages must be installed. On Ubuntu, this can be done by

```
sudo apt install libopengl-dev
sudo apt-get install libglu1-mesa-dev freeglut3-dev mesa-common-dev
```

If XCode is installed on MacOS, then development libraries for both OpenGL and the GLUT libraries should already be installed.

3.8 Compiling MolTwister

To compile MolTwister from the command line, goto the MolTwister source folder (where the CMakeLists.txt file resides). Then, type

```
cmake CMakeLists.txt -DINCLUDE_CUDA_COMMANDS=OFF
make
```

3.9 CUDA

To compile with GPU acceleration for the MD simulator in MolTwister, the latest CUDA development toolkit must be installed, together with appropriate drivers for the graphics card. To compile with cuda, use

```
cmake CMakeLists.txt -DINCLUDE_CUDA_COMMANDS=ON
make
```

There also exists a command line switch to enable debug code. This is achieved by adding

```
-DCUDA_DBG_INFO=ON
```

to the cmake call above.

3.10 Using QtCreator

It is recommended to use QtCreator as an IDE for the MolTwister project (since this has been tested extensively, both on MacOS and Ubuntu). The project is opened by choosing "Open Project" and selecting the CMakeLists.txt file. The first time the project is opened, the IDE will ask to configure the project. In that case, select "Desktop Qt x.y.z ..." (for MacOS choose the clang 64bit compiler). A file called CMakeLists.txt.user will be created. If it becomes necessary to redo the configuration process, simply delete this file. To perform the cmake configuration right click the project and select "Run CMake" (usually happend automatically the first time the project is configured). After this, it is possible to right click and select "Build" or "Rebuild".

On the menu that by default appears at the left hand side of the IDE window, there are a few options ("Welcome", "Edit", "Debug", "Projects" and "Help"). Click on "Projects" and "Build" (under "Build & Run"). There, it is possible to select and deselect the INCLUDE_CUDA_COMMANDS and CUDA_DBG_INFO options. Activate the selection by clicking "Apply Configuration Changes". Then, "Run CMake" and "Rebuild".

On MacOS, it is important to goto Projects > Buid & Run > Run, from the left hand menu bar and tick "Run in Terminal".

3.11 Installation of MolTwister

If MolTwister was compiled from the command line, make sure the working folder is the folder containing the source code (same folder as pulled from Git and the same folder from where `make` was called). In case QtCreater was used to build MolTwister, goto the build output folder created by QtCreator. Usually, the output folder is created one level below the source folder. However, this is configurable through the QtCreator user interface. Then execute

```
sudo make install
```

This will install MolTwister as 'moltwister' under /usr/local/bin.

It should now be possible to execute the `moltwister` command from any folder from within the command prompt, to start MolTwister.

CHAPTER

FOUR

TUTORIALS

4.1 Using the help system

Run MolTwister and type 'help', followe by enter. This will display a list of available commands. The 'load' command is an example of a command without subcommands. Type 'help load' to see the help text for this command. The 'calculate' command is an example of a command with subcommands. Type 'help calculate' to see the list of available subcommands. To see the help text for the 'vacf' subcommand, type 'help calculate vacf'.

The commands are described by using angle brackets, <>, to denote placeholders for parameter values. I.e., the angle brackets are not part of the commands, unless explicitly specified. The parameter values must always be in the order given in the help text. Any word not given inside the angle brackets are to be included, as is, when writing the command. Square brackets, `[]`, are not to be included in the command, unless explicitly specified. Parameters inside square brackets are considered as optional.

Go through various commands in the help system and find examples of the above notations.

Note that the commands are also listed under the 'List of commands' section in this online documentation.

4.2 Creating a water molecule PDB

Open MolTwister and execute the following sequence of commands:

```
add atom H1 at 0 0 0
add atom O from atom 0 dist 0.96
add atom H2 from bond 0 1 angle 104.5 0.0 dist 0.96
genbonds
```

This will create a hydrogen atom called H1 (where the leading H yields a hydrogen atom) at (x, y, z)=(0, 0, 0), with index 0. Then an oxygen atom, called O, is added a distance 0.96 Angstrom away from the atom with index 0. Finally, a hydrogen atom, called H2, is added from the bond defined between the atoms with index 0 (H1) and index 1 (O), at an angle of 104.5 degrees and a dihedral angle of 0.0 degrees, a distance 0.96 Angstrom from O. The bonds are generated by a call to `genbonds`.

We can measure the distances and angles to check that everything is as we expect. Type in the following commands:

```
measure bondlength id 0 1
measure bondlength id 1
measure angle id 0 1 2
```

Note that a physical bond does not have to be present to use the measure command. For example, the distance between H1 (index 0) and H2 (index 2) can be found by:

```
measure bondlength id 0 2
```

To create the PDB file content, write:

```
print pdb
```

However, to write this to a file, the pipe symbol can be used, as follows:

```
print pdb > my_water.pdb
```

You can now check that the PDB file is there by using the `ll` command (or the usual `ls` command). To chech the contents it is possible to use `vim my_water.pdb`, if Vim is installed.

4.3 Creating liquid water

Open MolTwister and make sure that you have the my_water.pdb file created in the above tutorial (or a similar PDB file). Execute the following commands:

```
load my_water.pdb
sel all
copy sel 10 10 10 3.0 3.0 3.0
autoscale
genbonds
```

This will load the PDB file into MolTwister, select all atoms of the water molecule and then copy that selection 10 times in the x, y and z directions with a distance of 3 Angstrom in each of the directions. Finally, `autoscale` is used to zoom in on the entire system, followed by the creation of bonds for each of the water molecules.

4.4 Creating a MolTwister script

Create a file called gen_my_water_mol.script and write the following contents:

```
exec("add atom H1 at 0 0 0");
exec("add atom O from atom 0 dist 0.96");
exec("add atom H2 from bond 0 1 angle 104.5 0.0 dist 0.96");
exec("genbonds");
```

Now, run the following command in MolTwister:

```
load gen_my_water_mol.script
```

This will create a water molecule, by calling each of the commands given in the file, in sequence.

4.5 Creating a MolTwister Python script

Create a Python script called gen_cube.py that contains (requires Python3):

```
import moltwister as mt

for i in range(0, 10):
    for j in range(0, 10):
        for k in range (0, 10):
            x = i*3.0
            y = j*4.0
            z = k*1.5
            mt.mt_exec(f"add atom O at {x} {y} {z}")

mt.mt_exec("autoscale")
```

Then, execute the following command in MolTwister:

```
load gen_cube.py
```

4.6 Water simulation, thermalization, RDF, MSD, VACF, and VDOS

4.6.1 Introduction

In this tutorial we will create a single water molecule, then make copies of this to create a water phase. Subsequently, we assign force field parameters, perform an energy optimization, and then a simulation.

Once the simulation is done, we will plot the temperature as a function of time to determine the point of thermalization. This is followed by calculating

- the radial distribution function,
- the mean square displacement and self-diffusion,
- the velocity auto correlation function,
- and the vibraional density of states.

4.6.2 Create a water molecule

Invoke

```
add atom Hw1 at 0 0 0
```

to add a hydrogen atom called Hw1. Then add the oxygen, Ow, by

```
add atom Ow at 0.9572 0 0
```

where the O-H bond lenght is 0.9572 Å. Finally, we want to add an atom at a 120 degree angle. To learn how this is done, invoke

```
help
help atom
```

Looking at this, we see that

```
add atom Hw2 from bond 0 1 angle 104.52 0 dist 0.9572
```

does what we want. We should also set appropriate boundaries to properly view the atom. Lets create a boundary that extends 10 Å in each direction from the coordinate origin by

```
mod userdefpbc -10 10 -10 10 -10 10
set userdefpbc on
```

and then lets make sure we view the entire boundary by invoking

```
autoscale
genbonds
```

This can be stored for safe keeping to a pdb and xyz file by

```
print pdb > single_water.pdb
print xyz > single_water.xyz
```

4.6.3 Assigning force field parameters

We can now assign force field parameters to the newly created water molecule. We could enter the commands directly, but we may need to assign the force fields several times during a study. Hence, we instead create a file called 'water_ff.script' using our favourite text editor (e.g., from within MolTwister you can call vim or nano). The file should contain

```
mod mass name Hw1 to 1.00784;
mod mass name Hw2 to 1.00784;
mod mass name Ow to 15.999;

add mdbond Ow Hw1 Harm all_r_less_than 1.1 1882.8 0.9572;
add mdbond Ow Hw2 Harm all_r_less_than 1.1 1882.8 0.9572;
add mdangle Hw1 Ow Hw2 Harm all_r_less_than 1.1 230.12 104.52;

add mdnonbonded Ow Ow LJ 0.426768 3.188;
```

which defines TIP3P water molecules.

4.6.4 Clear all contents

Many times we want to perform subtasks in MolTwister, where we end up with a script, pdb files, or similar that can be reloaded later. After creating these, we often want to delete everything we did from the MolTwister memory. This is done in two steps. First we delete what we see in the 3D view by

```
sel all
del sel
```

Subsequently, we delete all force field parameters by

```
del ff
```

On some systems, it is also possible to open several instances of MolTwister, which can be useful in some situations, but here we will proceede with clearing all contents.

4.6.5 Investigate the force field

Note that for the below Python scripts to work, MatplotLib must be installed on the system, which can be done by invoking

```
pip install matplotlib
```

Python must be installed for MolTwister to work, so this should already be present on the system.

Now load the required data into the system again by

```
load single_water.pdb
load water_ff.script
```

We can output data to show plots of the assigned force field potentials, but first we need to know the indices of the assigned force fields, which can be done by invoking

```
list ff
```

Invoke

```
help moldyn
```

to investigate the possible plots that can be made (i.e., bondforceprofile, angleforceprofile, dihedralforceprofile, and nonbondforceprofile). We can for example look at the Lennard-Jones potential between the water oxygens, which is the non-bonded potential at index 0. This can be done by

```
moldyn ff nonbondforceprofile 0 2.9 10.0 100 > OwOw_LJ.dat
```

where the result is stored in OwOw_LJ.dat file. This can be plotted using the following Python script, with name 'plot_non_bonded.py':

```
import matplotlib.pyplot as plt

listX = []
listY = []
f = open("OwOw_LJ.dat", "r")
startOfList = False
for line in f:
    if startOfList:
        s = line.split()
        if len(s) >= 3:
            listX.append(float(s[0])) # Extract r in Å
            listY.append(float(s[2])) # Extract U in kJ/mol

    if "Potential" in line:
        startOfList = True

fig, ax = plt.subplots()
ax.plot(listX, listY)

ax.set(xlabel='r [Å]', ylabel='U [kJ/mol]', title='Ow-Ow Lennard-Jones potential')
plt.show()
```

To plot the Lennard-Jones potential, invoke

```
python plot_non_bonded.py
```

or

```
python3 plot_non_bonded.py
```

depending on your system.

4.6.6 Create liquid water

We now want to use the single_water.pdb file that we created and construct a water phase containing 512 water molecules, randomly placed, without collision. Lets first calculate the approximate box size if the density is 997 kg/m^3, with two hydrogen masses of 1.0078 and one oxygen mass of 15.999:

```
calculate volumefromdensity 997 1.0078,1.0078,15.999 512
```

This yields a side length of approximately 25 Å for a cubic simulation box.

We should now make sure we start with a clean MolTwister:

```
sel all
del sel
del ff
```

First we load the water molecule:

```
load single_water.pdb
```

and say 'del' to delete anything previously there. Subsequently, select all atoms and make 512 copies that are placed randomly within a cubic box with 25 Å side lengths and a minimum distance between molecular atoms of 1.5 Å:

```
sel all
copy sel random 25 25 25 512 1.5
```

Check if we have managed to create 512 molecules by selecting all water oxygens (Ow) and counting them:

```
sel none
sel atomname Ow
measure count sel
```

The MD simulator only accepts systems with geometric centers placed at the origin. Hence, we need to displace the system to this position by invoking

```
mod atompos all geomcent 0 0 0
autoscale
```

We now save the system to a PDB file by

```
print pdb > water.pdb
```

4.6.7 Set up an appropriate periodic boundary condition (PBC)

We now want to set up appropriate periodic boundary conditions (PBC) for the simulation. We know that all atoms are within a box of 25 Å from the above construction, but we can also measure the PBC by

```
measure pbc
```

Then, we here add a sufficient amount on each side of the PBC to make sure that molecules do not come to close to the molecules on the other side of the PBC. Thus, we invoke

```
mod userdefpbc -12.5 12.5 -12.5 12.5 -12.5 12.5
```

to set a 25 by 25 by 25 Å boundary box and we invoke

```
set userdefpbc on
```

to activate it. It can be smart to create a file called 'pbc.script' and insert the two last commands, just so that we do not have to manually adjust the PBC every time, and instead just write

```
load pbc.script
```

when needed.

4.6.8 Perform energy optimization

The molecules were placed randomly in the liquid phase that we created, so it is a good idea to minimize forces between the atoms by performing an energy optimization. We will do this by using the default maximum number of steps for the optimization and the default learning rate, but we will set that the algorithm will stop if we reach an energy change of less than 10 kJ/mol.

To see the default values, invoke

```
moldyn cfg get
```

To set the accuracy of 10 kJ/mol, invoke

```
moldyn cfg set optaccuracy 10
```

We now start from a fresh MolTwister:

```
sel all
del sel
del ff
```

Next, invoke

```
load water.pdb
load pbc.script
load water_ff.script
```

Then, start the energy optimization by

```
moldyn cfg set outstride 10
moldyn optimizeenergy
```

and wait until it terminates.

This will produce a file called 'traj.dcd' and 'out.txt'.

By creating a Python script called 'plot_energy.py', containing

```
import matplotlib.pyplot as plt

listX = []
listY = []
f = open("out.txt", "r")
startOfList = False
for line in f:
    if startOfList and "Searching" not in line:
        s = line.split()
        if len(s) >= 6:
            listX.append(float(s[0])) # Extract time step column
            listY.append(float(s[5])) # Extract energy column

    if "Step," not in line and "Step" in line:
        startOfList = True

fig, ax = plt.subplots()
ax.plot(listX, listY)

ax.set(xlabel='Step', ylabel='U [kJ/mol]', title='Energy optimization')
plt.show()
```

we can view the energy plot of the optimization process by

```
python plot_energy.py
```

If we are happy with the results, we can export the last frame of the optimization process to a PDB file. First load the trajectory by

```
load traj.dcd
```

where you can view the optimization process by clicking the 3D view and using the left and right arrow keys to step through. You can also enable fog to better the depth visualization by

```
set fog on
```

We now export the optimizated frame (frame 499 in our case) by

```
print pdb frame 499 > initial.pdb
```

We will use this as our starting frame for the simulations.

If you wish, rename the traj.dcd and out.txt files from the energy optimization for safe keeping. They will be deleted in a later step.

4.6.9 Perform molecular dynamics simulation

We start again by clearing MolTwister memory:

```
sel all
del sel
del ff
```

Now remove any left-over traj.dcd and out.txt files.

We reload the initial system, the PBC, and the force field parameters:

```
load initial.pdb
load pbc.script
load water_ff.script
```

We configure to run 100 000 time steps at 0.5 fs, yielding a total of 50 ps, and we set the stride to 10 frames:

```
moldyn cfg set timestep 0.5
moldyn cfg set timesteps 100000
moldyn cfg set outstride 10
```

Now we can run the simulation as

```
moldyn run
```

On a relatively new CPU this simulation will take a few hours and produce a traj.dcd trajectory file and an out.txt output file.

If the simulation fails, such as producing NaN (Not a Number), then it is possible to try reducing the time step. For example, from 0.5 to 0.2 fs to gain better equilibration that way.

Once the simulation is done, we can load the trajectory file by

```
load traj.dcd
```

and analyze the simulation by using the left and right keyboard arrows. Invoke help to see how to adjust the number of steps per keypress, as well as how to show/hide bonds across boundaries, and switch between perspective and orthographic view.

4.6.10 Plot temperature as function of simulation time

The temperature as function of time step is available in 'out.txt' and we will use Python to plot the results. Create the file 'plot_temp.py' and enter the following script:

```
import matplotlib.pyplot as plt

listX = []
listY = []
f = open("out.txt", "r")
startOfList = False
for line in f:
    if startOfList and "Searching" not in line:
        s = line.split()
        if len(s) >= 3:
```

(continues on next page)

(continued from previous page)

```
            listX.append(float(s[0])) # Extract time step column
            listY.append(float(s[2])) # Extract temperature column

    if "Timestep," not in line and "Timestep" in line:
        startOfList = True

fig, ax = plt.subplots()
ax.plot(listX, listY)

ax.set(xlabel='Step', ylabel='T [K]', title='Temperature')
plt.show()
```

Now invoke

```
python plot_temp.py
```

which should show clearly the thermalization period of the system. This will be important later to determine at which points in the simulation we find thermalized data to perform calculations on. In this tutorial we will assume that the system was thermalized after 8000 time steps, which corresponds to frame index 800 in the DCD file, since we output every 10 timestep.

4.6.11 Calculate and plot a pair correlation function

We now want to calculate the pair correlation function, or radial distribution function (RDF), between oxygen atoms of water (Ow). We will use the data between frame 800 and 9999, which is the last frame of the DCD file, and create a plot of 100 points between 0.4 and 12 Å. This can be done by first making sure that we have loaded the system (i.e., the PDB file in our case, not the entire trajectory) and then invoking

```
calculate paircorrelation traj.dcd 800 9999 Ow Ow 100 0.4 12.0 > pair_corr.dat
```

which can be viewed using the following python script

```
import matplotlib.pyplot as plt

listX = []
listY = []
f = open("pair_corr.dat", "r")
startOfList = False
for line in f:
    if startOfList:
        s = line.split()
        if len(s) >= 4:
            listX.append(float(s[0])) # Extract r in Å
            listY.append(float(s[3])) # Extract RDF

    if "-----------" in line:
        startOfList = True

fig, ax = plt.subplots()
ax.plot(listX, listY)
```

(continues on next page)

(continued from previous page)

```
ax.set(xlabel='r [Å]', ylabel='RDF', title='Radial distribution function')
plt.show()
```

4.6.12 Calculate and plot mean square displacement

We want to calculate the mean square displacement (MSD). However, to use this command we first need to add residue names to our PDB file. We can now start with a clean MolTwister:

```
sel all
del sel
del ff
```

Then, load the system

```
load initial.pdb
```

We assign the residue names to each atom name by

```
mod resname name Ow to wat
mod resname name Hw1 to wat
mod resname name Hw2 to wat
```

where the water residue name was set to 'wat'. We now generate a new PDB file containing residue names by

```
print pdb > initial.pdb
```

Since the mean square displacement attempts to trace the molecules movements over time we need to unwrap the molecules accross the periodic boundaries before we can start the calculation. This is done by invoking

```
dcd unwrap traj.dcd
```

which will produce a file called 'traj_mtunwrap.dcd'.

To test the unwraped system, clear the MolTwister memory by

```
sel all
del sel
del ff
```

and then invoke

```
load initial.pdb
load pbc.script
load water_ff.script
load traj_mtunwrap.dcd
```

to load the system into the 3D viewer. In the 3D viewer, hit Alt+O to switch into orthographic view and then use the arrow keys to scroll though the frames, checking that the molecules move appropriately outside the periodic boundaries.

Once this is done we can calulate the mean square displacement by

```
calculate msd traj_mtunwrap.dcd 800 9999 resname wat > msd.dat
```

Note that the 'traj_mtunwrap.dcd' file does *not* need to be loaded into MolTwister memory in order to run the above command.

which can be plotted using the following Python script:

```
import matplotlib.pyplot as plt

listX = []
listY = []
f = open("msd.dat", "r")
startOfList = False
for line in f:
    if startOfList:
        s = line.split()
        if len(s) >= 2:
            listX.append(float(s[0])) # Extract step
            listY.append(float(s[1])) # Extract MSD

    if "Index" in line:
        startOfList = True

fig, ax = plt.subplots()
ax.plot(listX, listY)

ax.set(xlabel='step', ylabel='M [Å^2]', title='Mean square displacement')
plt.show()
```

4.6.13 Calculate and plot velocity auto correlation function

We wish to calculate the velocity auto correlation function (VACF). We then need to specify the time between each frame in fs, which is 5 in our case (i.e., 0.5 fs time step and output every 10 steps), as well as the number of time steps to include in the plot, which we will chose to be 100. The velocity auto correlation function should use the unwrapped system, as was also the case for the mean square displacement. First select the atoms to be included by

```
sel all
```

and then do the calculation by invoking

```
calculate vacf traj_mtunwrap.dcd 800 9999 5 100 sel > vacf.dat
```

and then plot using the below python script:

```
import matplotlib.pyplot as plt

listX = []
listY = []
f = open("vacf.dat", "r")
startOfList = False
for line in f:
    if startOfList:
        s = line.split()
        if len(s) >= 3:
            listX.append(float(s[0])) # Extract t in fs
```

(continues on next page)

(continued from previous page)

```
            listY.append(float(s[2])) # Extract VACF

    if "vacf" in line:
        startOfList = True

fig, ax = plt.subplots()
ax.plot(listX, listY)

ax.set(xlabel='t [fs]', ylabel='nVACF', title='Normalized auto correlation function')
plt.show()
```

4.6.14 Calculate and plot vibrational density of states

The vibrarional density of states (VDOS) can be calculated both using the unwrapped and wrapped systems, where we in our case will use the unwrapped one, since we have it available. We specify the starting frame for the calculation to 800, as well as the size of the fast Fourier transform (FFT) being applied, which is also the required number of available frames after the starting frame. We specify 10, which results in a 2^10 = 1024 FFT. We start the calculation by

```
sel all
calculate vdos traj_mtunwrap.dcd 800 10 0.5 sel > vdos.dat
```

We can plot the result using the following Python script:

```
import matplotlib.pyplot as plt

listX = []
listY = []
f = open("vdos.dat", "r")
startOfList = False
for line in f:
    if startOfList:
        s = line.split()
        if len(s) >= 6:
            listX.append(float(s[4])) # Extract wavelength, lambda, in nm
            listY.append(float(s[5])) # Extract VDOS

    if "inf" in line:
        startOfList = True

fig, ax = plt.subplots()
ax.plot(listX, listY)

plt.xlim([1780,8000])
plt.ylim([0,0.00014])
ax.fill_between(listX, listY, alpha=0.2)
ax.set(xlabel=r'$\lambda$ [nm]', ylabel='VDOS', title='Vibrational density of states')
plt.show()
```

CHAPTER

FIVE

IMPLEMENT NEW COMMAND

5.1 Command without subcommand

There are few steps to creating a new command in MolTwister. These are as follows.

Create a class that subclasses from CCmd. An example is shown below for the header file, which in this case should be called MolTwisterCmdMyCmd.h.

```
#pragma once
#include <string>
#include "MolTwisterCmd.h"

class CCmdMyCmd : public CCmd
{
public:
    CCmdMyCmd() = delete;
    CCmdMyCmd(CMolTwisterState* state) : CCmd(state) { state_ = state; }

public:
    void execute(std::string commandLine);
    std::string getCmd() const { return "mycmd"; }
    std::string getTopLevHelpString() const { return std::string("This command does what␣
↪I want"); }
    std::string getHelpString() const;

protected:
    virtual void onAddKeywords();
};
```

The corresponding implementation file should be called MolTwisterCmdMyCmd.cpp and would have the structure shown below.

```
#include "MolTwisterCmdMyCmd.h"

void CCmdMyCmd::onAddKeywords()
{
    addKeyword("mycmd");
    addKeyword("dothis")
    addKeyword("dothat")
}
```

(continues on next page)

(continued from previous page)

```
std::string CCmdMyCmd::getHelpString() const
{
    std::string text;

    text+= "\tUsage: mycmd <selection>\r\n";
    text+= "\r\n";
    text+= "\tThis command has two options that can be chosen by\r\n";
    text+= "\tsetting <selection>=dothis or <selection>=dothat.";

    return text;
}

void CCmdMyCmd::execute(std::string commandLine)
{
    int arg = 1

    std::string text = CASCIIUtility::getWord(commandLine, arg++);
    if(text == "dothis")
    {
        // Do this!
    }
    if(text == "dothat")
    {
        // Do that!
    }
}
```

Both these files should be stored under the Cmd folder. Once the command class has been created it must be added into the CMakeLists.txt file. This is done by adding entries into the MT_SOURCE_FILES variable:

```
        .
        .
        .
    ${CMAKE_CURRENT_LIST_DIR}/Cmd/MolTwisterCmdVarlist.h
    ${CMAKE_CURRENT_LIST_DIR}/Cmd/MolTwisterCmdVarlist.cpp
    ${CMAKE_CURRENT_LIST_DIR}/Cmd/MolTwisterCmdMolDyn.h
    ${CMAKE_CURRENT_LIST_DIR}/Cmd/MolTwisterCmdMolDyn.cpp
    ${CMAKE_CURRENT_LIST_DIR}/Cmd/MolTwisterCmdMyCmd.h
    ${CMAKE_CURRENT_LIST_DIR}/Cmd/MolTwisterCmdMyCmd.cpp
        .
        .
        .
```

Note that the CCmd class contains the state_ attribute, which is a pointer to CMolTwisterState. This contains information about the entire state of the molecular system. From this it is also possible to manipulate these states (such as adding, deleting and modifying atoms, as well as atomic force-fields). For example, an important public attribute of the CMolTwisterState class is atoms_, which is a vector of pointers to all atoms defined in the system. It is helpful to study other commands in the system to get familliar with how this class works. Another useful class to know about is the CDCDFile class, located under the Utilities folder. This can be used to access single records within a DCD file. This is also extensively used within other commands.

After the command had been created and added to the build process, it needs to be instantiated and registered. This is done in the CMolTwisterCommandPool class. To add the command, include the header file and add a registration of

the command in the implementation file:

```
#include "MolTwisterCommandPool.h"

#include "Cmd/MolTwisterCmdCd.h"
    .
    .
    .
#include "Cmd/MolTwisterCmdMyCmd.h"

void CMolTwisterCommandPool::generateCmdList(CMolTwisterState* mtState, std::vector
↪<std::shared_ptr<CCmd>>& cmdList)
{
    cmdList.emplace_back(std::make_shared<CCmdCd>(mtState));
        .
        .
        .
    cmdList.emplace_back(std::make_shared<CCmdMycmd>(mtState));

    for(size_t i=0; i<cmdList.size(); i++) cmdList[i]->init();
}
```

Now MolTwister should show the command under 'help' and it should be possible to execute the command. Autocomplete will be available for all keywords added with the addKeyword() method.

5.2 Command with subcommand

It is also possible to create commands where each sub-command of the main command is its own class. The 'measure' and 'calculate' commands are of this kind. In that case the main command header file is implemented in the following manner.

```
#pragma once
#include <string>
#include "MolTwisterCmd.h"

class CCmdMyCmd : public CCmd
{
public:
    CCmdMyCmd() = delete;
    CCmdMyCmd(CMolTwisterState* state) : CCmd(state) { state_ = state; }

public:
    virtual void execute(std::string commandLine);
    virtual std::string getCmd() const { return "mycmd"; }
    std::string getTopLevHelpString() const { return std::string("his command does what␣
↪I want"); }
    std::string getHelpString() const;
    std::string getHelpString(std::string subCommand) const;

protected:
    virtual void onAddKeywords();
    virtual void onRegisterSubCommands();
```

(continues on next page)

(continued from previous page)

```
};
```

As can be seen, compared to before, the getHelpString() has a second override and onRegisterSubCommands() is overriden. The implementation file for the main command looks as follows.

```
#include <stdio.h>
#include "MolTwisterCmdMyCmd.h"

#include "MolTwisterCmdMyCmd/CmdMySub1.h"
#include "MolTwisterCmdMyCmd/CmdMySub2.h"

void CCmdMyCmd::onAddKeywords()
{
    addKeyword("mycmd");
    addKeywords(parser_->getCmdLineKeywords());
}

void CCmdMyCmd::onRegisterSubCommands()
{
    parser_->purge();
    parser_->registerCmd(std::make_shared<CCmdMySub1>(state_, stdOut_));
    parser_->registerCmd(std::make_shared<CCmdMySub2>(state_, stdOut_));
}

std::string CCmdMyCmd::getHelpString() const
{
    return parser_->genHelpText(getCmd());
}

std::string CCmdMyCmd::getHelpString(std::string subCommand) const
{
    return parser_->genHelpText(getCmd(), subCommand);
}

void CCmdMyCmd::execute(std::string commandLine)
{
    std::string lastError = parser_->executeCmd(commandLine);
    if(!lastError.empty())
    {
        printf("%s\r\n", lastError.data());
        return;
    }

    // requestUpdate() is only needed if any of the commands require
    // updating of the 3D view contents!
    if(state_->view3D_)
    {
        if(state_->atoms_.size() == 1) state_->view3D_->requestUpdate(true);
        else                           state_->view3D_->requestUpdate(false);
    }
}
```

The header and implementation files for the main command should be stored under the same folder as for a command

without subcommands. Each subcommand class should be implemented in its own header and implementation file pair and stored under a folder with the same name as the files containing the main command. In this case that would be a folder with name MolTwisterCmdMyCmd.

The header file of a subcommand has the following structure

```
#pragma once
#include "../Tools/MolTwisterCmdEntry.h"

class CCmdMySub1 : public CCmdEntry
{
public:
    CCmdMySub1() = delete;
    CCmdMySub1(CMolTwisterState* state, FILE* stdOut) : CCmdEntry(state, stdOut) { }
    virtual ~CCmdMySub1() = default;

public:
    std::string getCmd();
    std::vector<std::string> getCmdLineKeywords();
    std::vector<std::string> getCmdHelpLines();
    std::string getCmdFreetextHelp();
    std::string execute(std::vector<std::string> arguments);

private:
    std::string lastError_;
};
```

where the file name of the header file would be CmdMySub1.h. The corresponding implementation file, CmdMySub1.cpp, has the structure shown below.

```
#include "CmdMySub1.h"
#include "../../Utilities/ASCIIUtility.h"

std::string CCmdMySub1::getCmd()
{
    return "mysub1";
}

std::vector<std::string> CCmdMySub1::getCmdLineKeywords()
{
    return { "mysub1", "dothis", "dothat" };
}

std::vector<std::string> CCmdMySub1::getCmdHelpLines()
{
    return {
                "mysub1 <selection>"
            };
}

std::string CCmdMySub1::getCmdFreetextHelp()
{
    std::string text;
```

(continues on next page)

(continued from previous page)

```
    text+= "\tThis subcommand has two options that can be chosen by\r\n";
    text+= "\tsetting <selection>=dothis or <selection>=dothat.";

    return text;
}

std::string CCmdMySub1::execute(std::vector<std::string> arguments)
{
    lastError_ = "";
    size_t arg = 0;

    std::string text = CASCIIUtility::getArg(arguments, arg++);
    if(text == "dothis")
    {
        // Do this!
    }
    else if(text == "dothat")
    {
        // Do that!
    }
    else
    {
        lastError_ = "Error: unknown selection!";
        return lastError_;
    }

    return lastError_;
}
```

To add more subcommands, add a new subcommand class for each, and register them (e.g., CCmdMySub2, which was registered in the example main command of this section).

Note that several versions of the subcommand can be registered in getCmdHelpLines(), separated by commas, where the available versions are determined by the implementation in execute().

CHAPTER

SIX

LIST OF COMMANDS

6.1 cd

Change directory (MolTwister implementation)

```
Usage: cd <destination directory>

Changes directory to <destination directory>. It is possible to specify
directory names containing spaces using the '\' character. For example,
'cd /Users/John\ Doe/' would change to the directory '/Users/John Doe'
To change directory to MolTwister shortcut N, type 'cd [N]' and hit enter.
Shortcuts can be edited directly by accessing 'MolTwister.shortcuts',
located under the home directoy of your computer. The contents of this
file is structured as follows:

        #Default
        <default directory at startup>
        #n
        <shortcut 1>
             .
             .
             .
        <shortcut n>

where n is the number of shortcuts available in the file.
```

6.2 lsm

Show current directory contents

```
Usage: lsm

Shows the contents of the current directory in the same way as the 'ls'
command in Linux, but will also show the directory shortcuts that are
available to the MolTwister 'cd' command. Go to the help pages for the
MolTwister 'cd' command for a further explanation on shortcuts. However,
the 'lsm' command does not accept any of theswitches available for the
Linux 'ls' command.
```

6.3 ll

Shortcut for the Unix 'ls -lha' command

```
Usage: ll

This is a shortcut for the 'ls -lha' command (only applicable on Unix based systems).
```

6.4 load

Load data into MolTwister

```
Usage: load <data type> <file path>

Load data into MolTwister. The allowed data types are:

xyz :        XYZ-coordinate files (including XYZ files containing
             several frames). Bonds can be ignored using 'ignore' (
             see genbonds). Use 'frame <frame>' to select frame to use
             as basis for generating the bonds. Default is frame zero.
             The order of the additional arguments is 'ignore', 'frame'.
dcd :        DCD-coordinate files. Bonds can be ignored using 'ignore' (
             see genbonds). Use 'frame <frame>' to select frame to use
             as basis for generating the bonds. Default is frame zero.
             Use 'stride <stride>' to only load every <stride> frame.
             The order of the additional arguments is 'ignore', 'frame', 'stride'.
xtc :        XTC-coordinate files. Bonds can be ignored using 'ignore' (
             see genbonds). Use 'frame <frame>' to select frame to use
             as basis for generating the bonds. Default is frame zero.
             Use 'stride <stride>' to only load every <stride> frame.
             The order of the additional arguments is 'ignore', 'frame', 'stride'.
pdb :        PDB-coordinate files. Bonds can be ignored using 'ignore'.
             To avoid query about delete ('del'), append ('app') or cancel ('can'),
             use 'noquery <response>' with the appropriate response.
mtt :        MolTwister trajectory file.
script :     MolTwister script containing a sequence of MolTwister commands,
             each formated as: exec("<MolTwister command>");. For example,
             exec("add atom C at 0 0 0"); would add a C atom at (0, 0, 0).
             It is also possible to leave out the exec() function, as well
             as the quotes to achieve the same effect. For example,
                    add atom C at 0 0 0;
             is equivalent to
                    exec("add atom C at 0 0 0");
masscharge : Load charges and masses into the current molecular structure. In
             the file each line defines a charge/mass in two possible ways:
                   * ID <ID, e.g. H, C, C2, or similar> <partial charge> <mass>
                   * AtInd <Index of atom> <partial charge> <mass>
qepos :      XYZ-position files (*.pos) from QuantumEspresso. Note that in
             this case <file path> = <*.pos file> <input file>

             If no data type is selected MolTwister will try to recognize
```

(continues on next page)

(continued from previous page)

```
             the file format from the file extension.
python :     Execute a python script. How to access MolTwister from a Python
             is documented under the 'mtpython' command. However, when loading
             a script, only the contents of 'mtpython {<contents>}' is to be
             included in the Python script.
```

6.5 genbonds

Generate bonds between atoms

```
Usage: genbonds [minr <minimum R>] [pbcdetect] [verbose] [atomicunwrap]

Generate bonds between atoms. A bond is defined as any bond length R that
satisfies 'minimum R' < r < r1+r2+0.4. In addition, the number of bonds that
are connected to C, N, P, S atoms are restricted to 4. The default minimum
bond length limit is 0.8AA. The parmeters inside square brackets are optional.
If the verbose keyword is used, then each detected bond will be displayed as
they are detected.

Using the pbcdetect keyword results in bonds across periodic images defined
by the current PBCs (i.e. Periodic Boundary Conditions). 'atomicunwrap'
results in molecules diveded by the PBCs being wrapped to one side of the
periodic boundary that divide the molecule.

To ignore bonds to atoms use 'ignore' followed by a comma separated list of
atoms (without any space).
```

6.6 list

List atomic properties

```
Usage: list <filter>

Lists atomic properties based on the specified filter. The allowed filters
are:

      * all                    :   List all
      * mol <N>                :   List atoms in molecule N
      * ff                     :   List only force-field parameters
      * latexff [longtable]    :   Make LaTeX force-field tables. 'longtable'
                                   creates tables across several pages
```

6.7 autoscale

Autoscale the 3D view window

```
Usage: autoscale

Autoscale the 3D view by searching for the boundary of all atoms inside the
3D view and subsequently adjusting the viewing frustum to these boundaries.
The camera is placed along the x-axis and will point towards the center of
the located atomic boundaries.
```

6.8 del

Delete atoms, bonds, etc.

```
Usage: del <specifier>

Delete atoms, bonds, angles and dihedrals. The allowed specifiers are:

        * atom <N>              :   Delete atom index <N>
        * atomname <name>       :   Delete atoms with name <name>
        * sel                   :   Delete selected atoms
        * bond <N1> <N2>        :   Delete bond <N1>-<N2>
        * bonds <1> <2>         :   Delete bonds <1>-<2>, where <1>, <2> are atom names.
                                    If <2>=* then all bonds to <1> are deleted
        * mdnonbonded <index>   :   The 'list ff' command yields list of indices
        * mdbond <index>        :   The 'list ff' command yields list of indices
        * mdangle <index>       :   The 'list ff' command yields list of indices
        * mddihedral <index>    :   The 'list ff' command yields list of indices
        * ff                    :   Delete force-field (m, Q, mobility, non-bonded, bonds,
↪ angles etc.)
```

6.9 copy

Copy atoms, molecules, etc.

```
Usage: copy <specifier>

Make several copies of atoms, or collection of atoms. The allowed specifiers are:

        * atoms <nx> <ny> <nz> <dx> <dy> <dz> <index1> ... <indexN>         : Copy <ni>␣
↪times
          with distance <di> between each copy, where i is x,y or z.
        * names <nx> <ny> <nz> <dx> <dy> <dz> <name1> ... <nameN>           : Copy based␣
↪on names.
        * names random <len x> <len y> <len z> <n> <dr> <name1> ... <nameN> : Copy based␣
↪on names.
        * all <nx> <ny> <nz> <dx> <dy> <dz>                                 : Copy all.
        * all random <len x> <len y> <len z> <n> <dr>                       : Copy all.
```

(continues on next page)

(continued from previous page)

```
        * sel <nx> <ny> <nz> <dx> <dy> <dz>                                    : Copy only␣
↪selected.
        * sel random <len x> <len y> <len z> <n> <dr>                          : Copy only␣
↪selected.
        * resnames <nx> <ny> <nz> <dx> <dy> <dz> <resname1> ... <resnameN>  : Copy based␣
↪on resnames.
        * resnames random <len x> <len y> <len z> <n> <dr> <resname1> ... <resnameN> :␣
↪Copy based on resnames.

If the random keyword is used, then <n> copies of the atomic set are randomly placed and␣
↪oriented within a box
with size <len x>, <len y>, and <len z>, starting from the first copy, such that the␣
↪atoms of the sets
never touch, with a least distance <dr>.
```

6.10 add

Add atom to system

```
Overview of commands of the form 'add <sub command>':

add atom <ID> [atomlabel <name> <x-displ> <y-displ> <z-displ> <r>, <g>, <b>] at <x> <y>
↪<z> [cubecpy <nx> <ny> <nz> <dx> <dy> <dz>, spherecpy <N> <R> [random]]
add atom <ID> [atomlabel <name> <x-displ> <y-displ> <z-displ> <r>, <g>, <b>] from atom
↪<n> [bondlabel <name> <x-displ> <y-displ> <z-displ> <r>, <g>, <b>] dist <d> [cubecpy...
↪]
add atom <ID> [atomlabel <name> <x-displ> <y-displ> <z-displ> <r>, <g>, <b>] from atom
↪<n> [bondlabel <name> <x-displ> <y-displ> <z-displ> <r>, <g>, <b>] dir <dx> <dy> <dz>␣
↪dist <d> [cubecpy...]
add atom <ID> [atomlabel <name> <x-displ> <y-displ> <z-displ> <r>, <g>, <b>] from bond
↪<n1> <n2> [bondlabel <name> <x-displ> <y-displ> <z-displ> <r>, <g>, <b>] angle <angle>
↪<dih> dist <d> [cubecpy...]
add atom <ID> [atomlabel <name> <x-displ> <y-displ> <z-displ> <r>, <g>, <b>] from angle
↪<n1> <n2> <n3> [bondlabel <name> <x-displ> <y-displ> <z-displ> <r>, <g>, <b>] angle
↪<angle> <dih> dist <d> [cubecpy...]
add atoms <ID> sel <dx> <dy> <dz>
add mdnonbonded <ID1> <ID2> <FF-type> <parameters for given FF-type>
add mdbond <ID1> <ID2> <FF-type> [all_r_less_than <radius>, mol_r_less_than <radius>,␣
↪only_visible_bonds, only_14_bonds] <parameters for given FF-type>
add mdangle <ID1> <ID2> <ID3> <FF-type> [all_r_less_than <radius>, mol_r_less_than
↪<radius>, only_visible_bonds] <parameters for given FF-type>
add mddihedral <ID1> <ID2> <ID3> <ID4> <FF-type> [all_r_less_than <radius>, mol_r_less_
↪than <radius>, only_visible_bonds] <parameters for given FF-type>

To get more information about a <sub command>, type 'help add <sub command>
```

6.10.1 atom

```
Overview of commands of the form 'add atom':

add atom <ID> [atomlabel <name> <x-displ> <y-displ> <z-displ> <r>, <g>, <b>] at <x> <y>
↪<z> [cubecpy <nx> <ny> <nz> <dx> <dy> <dz>, spherecpy <N> <R> [random]]
add atom <ID> [atomlabel <name> <x-displ> <y-displ> <z-displ> <r>, <g>, <b>] from atom
↪<n> [bondlabel <name> <x-displ> <y-displ> <z-displ> <r>, <g>, <b>] dist <d> [cubecpy...
↪]
add atom <ID> [atomlabel <name> <x-displ> <y-displ> <z-displ> <r>, <g>, <b>] from atom
↪<n> [bondlabel <name> <x-displ> <y-displ> <z-displ> <r>, <g>, <b>] dir <dx> <dy> <dz>␣
↪dist <d> [cubecpy...]
add atom <ID> [atomlabel <name> <x-displ> <y-displ> <z-displ> <r>, <g>, <b>] from bond
↪<n1> <n2> [bondlabel <name> <x-displ> <y-displ> <z-displ> <r>, <g>, <b>] angle <angle>
↪<dih> dist <d> [cubecpy...]
add atom <ID> [atomlabel <name> <x-displ> <y-displ> <z-displ> <r>, <g>, <b>] from angle
↪<n1> <n2> <n3> [bondlabel <name> <x-displ> <y-displ> <z-displ> <r>, <g>, <b>] angle
↪<angle> <dih> dist <d> [cubecpy...]

In general, ID could for example be C, H, or O (i.e. carbon, hydrogen, or
oxygen). It is also possible to give names such as C1 or C2. As long as
the name contains a C it is recognized as a carbon atom. Similarly, any
name containing O will be recognized as oxygen, etc.

The 'atomlabel' keyword can be used to add a label with any <name>, without white
characters, that can be displayed a displacement (<x-disp>, <y-disp>, <z-displ>) away
from the atomic position, with the color {<r>, <g>, <b>}. Similarly, a label can be
added relative to each bond center by applying the 'bondlabel' keyword.

When atoms are added they attain a new index starting at index 0. The list
of atomic indices are obtained through the 'list' command.
Versions of the add command:

 1) Add atom at the coordinate (<x>, <y>, <z>). Grid copy makes <na> copies separated
    by a distance <da>, where a is x, y, and z. Sphere-copy places <N> atoms at surface
    of a sphere of radius R, approx. equidistant., unless 'random' is specified.

 2) Add atom a distance <d> in the y-direction from the atom with index <n>.

 3) Add atom a distance <d> in the (<dx>, <dy>, <dz>)-direction from the atom
    with index <n>.

 4) Add atom with index n3, such that the angle <n1>-<n2>-<n3> is <angle>
    degrees, with the <n2>-<n3> bond being rotated <dih> around the <n1>-<n2>
    bond (automatically using the x-, y-, or z- axis as reference). The bond
    length <n2>-<n3> is set to <d>.

 5) Add atom with index n4, such that the angle <n2>-<n3>-<n4> is <angle>
    degrees, with the <n1>-<n2>-<n3>-<n4> dihedral set to <dih> degrees.
    The bond length <n3>-<n4> is set to <d>.
```

6.10.2 atoms

```
Overview of commands of the form 'add atoms':

add atoms <ID> sel <dx> <dy> <dz>

In general, ID could for example be C, H, or O (i.e. carbon, hydrogen, or
oxygen). It is also possible to give names such as C1 or C2. As long as
the name contains a C it is recognized as a carbon atom. Similarly, any
name containing O will be recognized as oxygen, etc.

When atoms are added they attain a new index starting at index 0. The list
of atomic indices are obtained through the 'list' command.

The above command will add one atom of type <ID> a distance <dx> <dy> <dz> away
from each selected atom.
```

6.10.3 mdnonbonded

```
Overview of commands of the form 'add mdnonbonded':

add mdnonbonded <ID1> <ID2> <FF-type> <parameters for given FF-type>

ID1 and ID2 identifiy the atom types where a non-bonded interaction is to
be defined (e.g., H, O, C, O5). The force field type, FF-type, is idefined by
a string. The possible strings are listed below, together with how 'parameters
for given FF-type' is defined for each string.

Possible <FF-type> <parameters for given FF-type> to apply:
* LJ <Lennard-Jones epsilon> <Lennard-Jones sigma> [alpha <Lennard-Jones alpha>]
* LJ1208 <Lennard-Jones epsilon> <Lennard-Jones sigma> [alpha <Lennard-Jones alpha>]
* Buck <Buckingham A> <Buckingham rho> <Buckingham C>
* LJBuck 0 <Lennard-Jones epsilon> <Lennard-Jones sigma>
* LJBuck 1 <Buckingham A> <Buckingham rho> <Buckingham C>
```

6.10.4 mdbond

```
Overview of commands of the form 'add mdbond':

add mdbond <ID1> <ID2> <FF-type> [all_r_less_than <radius>, mol_r_less_than <radius>,␣
↪only_visible_bonds, only_14_bonds] <parameters for given FF-type>

ID1 and ID2 identifiy the atom types where a bond interaction is to be
defined (e.g., H, O, C, O5). The force field type, FF-type, is idefined by
a string. The possible strings are listed below, together with how 'parameters
for given FF-type' is defined for each string.

Possible bond definitions to apply:
* all_r_less_than <r> - yields bonds even between atoms not close enough to define a␣
↪molecule (if the r-criteria is satisfied)
```

(continues on next page)

(continued from previous page)

```
* mol_r_less_than <r> - yields bonds only between atoms close enough to define a␣
↪molecule (if the r-criteria is satisfied)
* only_visible_bonds - yields bonds only where bonds are visible in the 3D view
* only_14_bonds - yields bonds only between 1-4 interactions

Possible <FF-type> <parameters for given FF-type> to apply:
* Harm <Harmonic k> <Harmonic r_0>
* Morse <Morse D> <Morse alpha> <Morse r_0>
* LJC <Lennard-Jones epsilon> <Lennard-Jones sigma> <scale>
```

6.10.5 mdangle

```
Overview of commands of the form 'add mdangle':

add mdangle <ID1> <ID2> <ID3> <FF-type> [all_r_less_than <radius>, mol_r_less_than
↪<radius>, only_visible_bonds] <parameters for given FF-type>

ID1, ID2 and ID3 identifiy the atom types where an angular interaction is to be
defined (e.g., H, O, C, O5). The force field type, FF-type, is idefined by
a string. The possible strings are listed below, together with how 'parameters
for given FF-type' is defined for eaah string.

Possible bond definitions to apply:
* all_r_less_than <r> - yields bonds even between atoms not close enough to define a␣
↪molecule (if the r-criteria is satisfied)
* mol_r_less_than <r> - yields bonds only between atoms close enough to define a␣
↪molecule (if the r-criteria is satisfied)
* only_visible_bonds - yields bonds only where bonds are visible in the 3D view

Possible <FF-type> <parameters for given FF-type> to apply:
* Harm <Harmonic k> <Harmonic theta0(deg)>
* Class2 <Ea.theta0> <Ea.K2> <Ea.K3> <Ea.K4> <Ebb.M> <Ebb.r1> <Ebb.r2> <Eba.N1> <Eba.N2>
↪<Eba.r1> <Eba.r2>
```

6.10.6 mddihedral

```
Overview of commands of the form 'add mddihedral':

add mddihedral <ID1> <ID2> <ID3> <ID4> <FF-type> [all_r_less_than <radius>, mol_r_less_
↪than <radius>, only_visible_bonds] <parameters for given FF-type>

ID1, ID2, ID3 and ID4 identifiy the atom types where a dihedral interaction is to be
defined (e.g., H, O, C, O5). The force field type, FF-type, is idefined by
a string. The possible strings are listed below, together with how 'parameters
for given FF-type' is defined for eaah string.

Possible bond definitions to apply:
* all_r_less_than <r> - yields bonds even between atoms not close enough to define a␣
↪molecule (if the r-criteria is satisfied)
```

(continues on next page)

(continued from previous page)

```
* mol_r_less_than <r> - yields bonds only between atoms close enough to define a␣
↪molecule (if the r-criteria is satisfied)
* only_visible_bonds - yields bonds only where bonds are visible in the 3D view

Possible <FF-type> <parameters for given FF-type> to apply:
* Fourier4t <V1> <V2> <V3> <V4>
* Harm <Harmonic K> <Harmonic D> <Harmonic N>
```

6.11 var

Define a variable

```
Usage: var <type+value> <variable name>

Define a variable that, for example, can be used (in some situations)
instead of atomic indices. <type+value> can be one of the following:

  * atom <N>                      : Define atomic variable with index <N>
  * bond <N1> <N2>                : Define bond variable between index <N1>
                                    and index <N2>
  * angle <N1> <N2> <N3>          : Define angle variable between index <N1>,
                                    <N2> and <N3>
  * dihedral <N1> <N2> <N3> <N4> : Define dihedral variable between index
                                    <N1>, <N2>, <N3> and <N4>

Defining variables can be useful for commands such as 'mod', 'measure',
and 'gauss9', to avoid having to remember atomic indices.
```

6.12 varlist

List all variables defined by 'var'

```
Usage: varlist

List all variables that have been defined using the 'var command.
The variable names, their type and their content will be listed.
```

6.13 mod

Modify settings, parameters and states

```
Overview of commands of the form 'mod <sub command>':

mod angle id <atom index 1> <atom index 2> <atom index 3> to <angle>
mod angle var <variable name> to <angle>
mod atomname <atom ID to change> <atom ID to change to> [within <domain>]
```

(continues on next page)

(continued from previous page)

```
mod atompos id <atom index> to <x> <y> <z>
mod atompos id <atom index> by <x> <y> <z>
mod atompos id <atom index> geomcent <x> <y> <z>
mod atompos id <atom index> flip <flip axis>
mod atompos var <variable name> to <x> <y> <z>
mod atompos var <variable name> by <x> <y> <z>
mod atompos var <variable name> geomcent <x> <y> <z>
mod atompos var <variable name> flip <flip axis>
mod atompos all to <x> <y> <z>
mod atompos all by <x> <y> <z>
mod atompos all geomcent <x> <y> <z>
mod atompos all flip <flip axis>
mod atompos sel to <x> <y> <z>
mod atompos sel by <x> <y> <z>
mod atompos sel geomcent <x> <y> <z>
mod atompos sel flip <flip axis>
mod bondlength id <atom index 1> <atom index 2> to <dist>
mod bondlength var <variable name> to <dist>
mod bondtype id <atom index 1> <atom index 2> to <type>
mod bondtype var <variable name> to <type>
mod charge id <atom index> to <partial charge>
mod charge var <variable name> to <partial charge>
mod charge name <atom ID> to <partial charge>
mod dihedral id <atom index 1> <atom index 2> <atom index 3> <atom index 4> to <angle>
mod dihedral var <variable name> to <angle>
mod mass id <atom index> to <atomic mass>
mod mass var <variable name> to <atomic mass>
mod mass name <atom ID> to <atomic mass>
mod mobility id <atom index> to <mobility>
mod mobility var <variable name> to <mobility>
mod mobility name <atom ID> to <mobility>
mod mobility sel to <mobility>
mod resname id <atom index> to <resname>
mod resname var <variable name> to <resname>
mod resname name <atom ID> to <resname>
mod resname resname <resname> to <resname>
mod resname molecule <molecule index> to <resname>
mod resname atomnames <atom ID1> <atom ID2> ... <atom IDn> to <resname>
mod rotatesel <angle> <pos. of rotation vector> <rotation vector>
mod sigma id <atom index> to <sigma>
mod sigma var <variable name> to <sigma>
mod defaultatomprops cpkcolor name <atom id> to <R> <G> <B>
mod defaultatomprops vdwr name <atom id> to <r>
mod defaultatomprops covalentr name <atom id> to <r>
mod defaultatomprops reset
mod userdefpbc <x low> <x high> <y low> <y high> <z low> <z high>

To get more information about a <sub command>, type 'help mod <sub command>
```

6.13.1 angle

```
Overview of commands of the form 'mod angle':

mod angle id <atom index 1> <atom index 2> <atom index 3> to <angle>
mod angle var <variable name> to <angle>

Modifies the angle, given by
* the three atom indices <atom index 1> to <atom index 3>
* the atom indices contained in the variable <variable name>
to the angle given by <angle>, which is given in degrees.
```

6.13.2 atomname

```
Overview of commands of the form 'mod atomname':

mod atomname <atom ID to change> <atom ID to change to> [within <domain>]

Renames atom ID (e.g., H, O, C7) from <atom ID to change> to <atom ID to change to>,␣
↪either
for all atoms of the loaded system, if 'within' is not specified, or for the atoms␣
↪within the
given <domain>.

The <domain> parameter consists of a string without space, where several
conditions are specified, separated by boolean operators. The boolean
operators that can be applied are:
* not: !
* and: &
* and: *
* or: |
* or: +
* xor: ^
Each condition is on x, y or z and can be of the following types
* equal: =
* not equal: !=
* greater than: >
* greater than or equal: >=
* less than: <
* less than or equal: <=
For example,
* x>5
* x>5&x<10
* (x>5&x<10)|(x>20&x<23)
* !(x>5&x<10)
```

6.13.3 atompos

```
Overview of commands of the form 'mod atompos':

mod atompos id <atom index> to <x> <y> <z>
mod atompos id <atom index> by <x> <y> <z>
mod atompos id <atom index> geomcent <x> <y> <z>
mod atompos id <atom index> flip <flip axis>
mod atompos var <variable name> to <x> <y> <z>
mod atompos var <variable name> by <x> <y> <z>
mod atompos var <variable name> geomcent <x> <y> <z>
mod atompos var <variable name> flip <flip axis>
mod atompos all to <x> <y> <z>
mod atompos all by <x> <y> <z>
mod atompos all geomcent <x> <y> <z>
mod atompos all flip <flip axis>
mod atompos sel to <x> <y> <z>
mod atompos sel by <x> <y> <z>
mod atompos sel geomcent <x> <y> <z>
mod atompos sel flip <flip axis>

Modifies the atomic positions of one or more atoms. The atoms to be moved can be␣
↪specified by
* id <atom index>
* var <variable name>, where variable name contains atom index
* all, which signifies all loaded or created atoms
* sel, which signifies all selected atoms
The relative motion of the translations can be one of the following
* to <x> <y> <z>, which will move all chosen atoms to (<x>, <y>, <z>)
* by <x> <y> <z>, which will translate all chosen atoms by <x>, <y> and <z> along the x-,
↪ y-
                  and z-axis, respectively
* geomcent <x> <y> <z>, which will translate all chosen atoms such that geometric center␣
↪of
                       their bounding box is positioned at (<x>, <y>, <z>)
* flip <flip axis>, where <flip axis> is either x, y, or z, which will flip all the␣
↪chosen
                    atoms are mirrored across the selected axis around the bounding box␣
↪center
                    of the chosen atoms
```

6.13.4 bondlength

```
Overview of commands of the form 'mod bondlength':

mod bondlength id <atom index 1> <atom index 2> to <dist>
mod bondlength var <variable name> to <dist>

Modifies the distance between two atoms, typically a bond, given by
* the two atom indices <atom index 1> and <atom index 3>
* the atom indices contained in the variable <variable name>
to the distance given by <dist>.
```

6.13.5 bondtype

```
Overview of commands of the form 'mod bondtype':

mod bondtype id <atom index 1> <atom index 2> to <type>
mod bondtype var <variable name> to <type>

Modifies the bond type between two atoms, given by
* the two atom indices <atom index 1> and <atom index 3>
* the atom indices contained in the variable <variable name>
to the type given by <type>, which can be 'single' or 'double'.
```

6.13.6 charge

```
Overview of commands of the form 'mod charge':

mod charge id <atom index> to <partial charge>
mod charge var <variable name> to <partial charge>
mod charge name <atom ID> to <partial charge>

Modifies the charge of atoms given by
* the atom index, <atom index>
* the atom index contained in the variable <variable name>
* all atoms with atom ID = <atom ID> (e.g., O, H, C7)
to the charge given by <partial charge>.
```

6.13.7 dihedral

```
Overview of commands of the form 'mod dihedral':

mod dihedral id <atom index 1> <atom index 2> <atom index 3> <atom index 4> to <angle>
mod dihedral var <variable name> to <angle>

Modifies the dihedral angle, given by
* the three atom indices <atom index 1> to <atom index 4>
* the atom indices contained in the variable <variable name>
to the dihedral angle given by <angle>, which is given in degrees.
```

6.13.8 mass

```
Overview of commands of the form 'mod mass':

mod mass id <atom index> to <atomic mass>
mod mass var <variable name> to <atomic mass>
mod mass name <atom ID> to <atomic mass>

Modifies the atomic mass of atoms given by
* the atom index, <atom index>
```

(continues on next page)

(continued from previous page)

```
* the atom index contained in the variable <variable name>
* all atoms with atom ID = <atom ID> (e.g., O, H, C7)
to the mass given by <atomic mass>.
```

6.13.9 mobility

```
Overview of commands of the form 'mod mobility':

mod mobility id <atom index> to <mobility>
mod mobility var <variable name> to <mobility>
mod mobility name <atom ID> to <mobility>
mod mobility sel to <mobility>

Modifies the atomic mobility of atoms given by
* the atom index, <atom index>
* the atom index contained in the variable <variable name>
* all atoms with atom ID = <atom ID> (e.g., O, H, C7)
* all atoms contained in the current visible selection ('sel')
to the mobility given by <mobility>, which can be
either 'mobile' (e.g., included in MD integrations)
or 'fixed' (e.g., not included in MD integrations).
```

6.13.10 resname

```
Overview of commands of the form 'mod resname':

mod resname id <atom index> to <resname>
mod resname var <variable name> to <resname>
mod resname name <atom ID> to <resname>
mod resname resname <resname> to <resname>
mod resname molecule <molecule index> to <resname>
mod resname atomnames <atom ID1> <atom ID2> ... <atom IDn> to <resname>

Modifies the resname of atoms given by
* the atom index, <atom index>
* the atom index contained in the variable <variable name>
* all atoms with atom ID = <atom ID> (e.g., O, H, C7)
* all atoms with resname = <resname>
* all atoms within molecular index = <molecule index>
* all atoms with atom IDs, <atom ID1> through <atom IDn>
to the resname given by <resname>.
```

6.13.11 rotatesel

```
Overview of commands of the form 'mod rotatesel':

mod rotatesel <angle> <pos. of rotation vector> <rotation vector>

Rotates the current selection <angle> degrees around the vector <rotation vector>,
which is positioned at <pos. of rotation vector>. The position and vector are formatted
as three numbers, <x> <y> <z>, separated by space.
```

6.13.12 sigma

```
Overview of commands of the form 'mod sigma':

mod sigma id <atom index> to <sigma>
mod sigma var <variable name> to <sigma>

Changes the van der Waals radius, denoted as sigma, to the value stated by <sigma>.
This is not to be confued with the Lennard-Jones sigma for the assigned force fields.
The van der Waals radius is used in calculations such as for iso surfaces describing
a surface plot surrounding a molecule or atomic cluster.

The atom index to receive the new value of sigma is identified by
* directly assigning the atom index, <atom index>
* a variable, <variable name>, containing the atom index
```

6.13.13 defaultatomprops

```
Overview of commands of the form 'mod defaultatomprops':

mod defaultatomprops cpkcolor name <atom id> to <R> <G> <B>
mod defaultatomprops vdwr name <atom id> to <r>
mod defaultatomprops covalentr name <atom id> to <r>
mod defaultatomprops reset

Changes the default CPK color, van der Waals radius, and covalent radius, to the value
stated by <R>, <G>, <B>, and <r>. Red, green and blue (i.e., <R>, <G>, <B>) are values
between zero and one, while the radius is in units of AA. Note that this only changes
the values for the currently opened MolTwister instance and will not persist the next
time MolTwister opens. Hence, if the new default values are used repeatedly, it is
recommended to create a script containing the new values (see 'load' for more information
on scripts).
```

6.13.14 userdefpbc

```
Overview of commands of the form 'mod userdefpbc':

mod userdefpbc <x low> <x high> <y low> <y high> <z low> <z high>

Sets a user defined periodic boundary condition (PBC). Note that the
user defined PBC will not automatically be applied, but needs to be
activated through the 'set' command.
```

6.14 set

Set various properties of MolTwister and the loaded systems

```
Usage: set projection <projection>
       set fullscreen <fullscreen>
       set userdefpbc <userdefpbc>
       set bondacrosspbc <bondacrosspbc>
       set redrawlimit <num atoms>
       set fog <fog>
       set usevdwradius <usevdwradius>
       set vdwscalefactor <vdwscalefactor>
       set labels <labels>
       set labelsfontsize <fontsize>
       set backgroundcolor <r> <g> <b>
       set backgroundcolor default

This command will set various properties of MolTwister or its loaded systems.

The <projection> can be either 'perspective' or 'ortho'.

The <fullscreen> parameter can be either 'on' or 'off'.

The <userdefpbc> parameter can be either 'on' or 'off'. If 'on', the user defined
periodic boundary conditions (PBC) set by the 'mod userdefpbc' command is applied.
If 'off', the PBC will be collected elsewhere.

The <bondacrosspbc> parameter can be either 'on' or 'off'. If 'on', bonds acrsss
PBCs will be visible in the 3D view.

The 'redrawlimit' sub-command will specify that, during rotation / panning / scaling
of the scene, scenes with more than <num atoms> atoms will only be displayed with the
axis system. All atoms will be redrawn once the corresponding mouse button is released.

The <fog> can be either 'on' or 'off'.

The <usevdwradius> can be either 'on' or 'off'. If 'on' the atoms are drawn using the
built in van der Waals radius multiplied by <vdwscalefactor> (default 1).

The <labels> can be either 'on' or 'off'. If 'on', the atom and bond labels defined in␣
↪the
```

(continues on next page)

(continued from previous page)

```
'add atom' command will be displayed.

The <fontsize> can be either 10, 12 or 18.

The color values <r> <g> <b> (red, green and blue) must be between 0 and 1.
```

6.15 get

Get various properties of MolTwister and the loaded systems

```
Usage: get <args>

Retrieve various properties of the system. <args> can be:

       * atomtypes                 :   List all atom types present in system
       * mdinconsistency           :   Check if any inconsistencies in MD force-field␣
↪can be found
       * bondinfo <ind 1> <ind 2>  :   Get a list of bonds connected to atom indices 1␣
↪and 2
       * userdefpbc                :   Get user defined PBCs
       * defaultatomprops          :   Default atom properties, such as van derWaals␣
↪radius and CPK color
       * gpuinfo                   :   Check if CUDA is available and return info
```

6.16 sel

Create a selection of atoms

```
Usage: sel atom <atom index>
       sel atomname <atom ID (e.g., O, H, C7)> [within <condition>]
       sel all [within <domain>]
       sel none

This command will create an atomic selection, where the selected atoms will be␣
↪highlighted
in the 3D view. The first time a selection is done, the selection is done. The second␣
↪time
the same selection is done, the selection is unselected, except for 'sel all' and 'sel␣
↪none'.

The <domain> parameter consists of a string without space, where several
conditions are specified, separated by boolean operators. The boolean
operators that can be applied are:
* not: !
* and: &
* and: *
* or: |
* or: +
```

(continues on next page)

(continued from previous page)

```
* xor: ^
Each condition is on x, y or z and can be of the following types
* equal: =
* not equal: !=
* greater than: >
* greater than or equal: >=
* less than: <
* less than or equal: <=
For example,
* x>5
* x>5&x<10
* (x>5&x<10)|(x>20&x<23)
* !(x>5&x<10)
```

6.17 print

Print information (e.g., atom pos. in different file formats)

```
Usage: print <content>

Prints, to screen, or file if piped ('>'), the specified contents, where <contents> can␣
↪be any of the following
* xyz [currentframe] [atomicunits]: atomic IDs and positions in the XYZ-file format
* xyz [frame <frame index>] [atomicunits]: atomic IDs and positions in the XYZ-file␣
↪format
* pdb [currentframe]: atomic IDs and positions in the PDB-file format (Protein Data Bank␣
↪file format)
* pdb [frame <frame index>]: atomic IDs and positions in the PDB-file format (Protein␣
↪Data Bank file format)
* mtt [bondsfromff] [bondacrosspbc] [currentframe] [binary]: atomic IDs and positions in␣
↪the MolTwister file format
* mtt [bondsfromff] [bondacrosspbc] [frame <frame index>] [binary]: atomic IDs and␣
↪positions in the MolTwister file format
* version: version of the software
* mixff resname <resname 1> resname <resname 2> <mixing-rule>: finds a mixed set of␣
↪force field parameters between atoms in
                                                                    <resname 1> and <resname␣
↪2> and outputs a list of mixed
                                                                    parameters in the form of␣
↪MolTwister exec() commands, that
                                                                    constitutes a script that␣
↪sets up the mixing force field
* mixff atomnames <list of atom IDs 1> atomnames <list of atom IDs 2> <mixing-rule>:␣
↪finds a mixed set of force field parameters
                                                                                          ␣
↪between atoms in <list of atom IDs 1> and

↪<list of atom IDs 2> and outputs a list
                                                                                        of␣
↪mixed parameters in the form of MolTwister
```

(continues on next page)

(continued from previous page)

```
                                                                                ␣
↪exec() commands, that constitutes a script
                                                                                ␣
↪that sets up the mixing force field

If 'atomicunits' is not specified, then the positions are given in Aangstroms. If
↪'bondsfromff' is specified,
bonds are taken from the given force field bonds, otherwise bonds are calculated based␣
↪on atomic separations
and will be identical to the bonds visible in the 3D view. If 'bondacrosspbc' is not␣
↪specified, then the bonds
will not stretch across the periodic boundary conditions (PBC). The 'binary' keyword␣
↪will invoke a binary
version of the file format, if available. If neither 'currentframe' or 'frame' is␣
↪specified, the default frame
to print is the frame at index 0. The <mixing-rule> parameter depends on the force␣
↪fields to be mixed. In
some cases it is not possible to choose the rule (the parameter will be ignored), but in␣
↪others it is either
'aritmetic' or 'geometric'. Note that the <list of atom IDs X> consists of a comma␣
↪separated list of atom IDs
(e.g., H, O, C7) without any space.
```

6.18 measure

Perform a measurement

```
Overview of commands of the form 'measure <sub command>':

measure angle id <atom index 1> <atom index 2> <atom index 3>
measure angle var <variable name>
measure atompos id <atom index>
measure atompos var <variable name>
measure atompos sel
measure bondcount <atom ID>
measure bondlength id <atom index 1> <atom index 2>
measure bondlength var <variable name>
measure bondlengthdyn <DCD filename> <frame from> <frame to> <atom indices list (each␣
↪pair defines a bond)>
measure bondsep id <atom index 1> <atom index 2>
measure bondsep var <variable name>
measure center sel
measure charge sel
measure charge tot
measure coulombenergy single [1to4coeffs <coeff 1> <coeff 2> <coeff 3> <coeff 4>]
measure coulombenergy dihedralrot <dihedral> rot <start angle> <end angle> <angular step␣
↪size> [1to4coeffs <coeff 1> <coeff 2> <coeff 3> <coeff 4>]
measure coulombenergy anglerot <angle> rot <start angle> <end angle> <angular step size>␣
↪[1to4coeffs <coeff 1> <coeff 2> <coeff 3> <coeff 4>]
measure coulombenergy bondstretch <bond> stretch <start dist> <end dist> <dist step size>
```

(continues on next page)

(continued from previous page)

```
↪ [1to4coeffs <coeff 1> <coeff 2> <coeff 3> <coeff 4>]
measure coulombenergy allframes [1to4coeffs <coeff 1> <coeff 2> <coeff 3> <coeff 4>]
measure coulombpotential single at <x> <y> <z>
measure coulombpotential dihedralrot at <x> <y> <z> <dihedral> rot <angle start> <angle␣
↪end> <angular step size>
measure count sel
measure count tot
measure dihedral <dihedral>
measure overlap <within> sel
measure pbc [<frame index>]
measure radiusofgyration sel

To get more information about a <sub command>, type 'help measure <sub command>
```

6.18.1 angle

```
Overview of commands of the form 'measure angle':

measure angle id <atom index 1> <atom index 2> <atom index 3>
measure angle var <variable name>

Measures the angle between three atoms (in degrees) by
* defining three atom indices directly by using the 'id' keyword
* obtaining atom indices from a variable by using the 'var' keyword

Output:
Theta(<atom index 1>, <atom index 2>, <atom index 3>) = <angle in degrees>
```

6.18.2 atompos

```
Overview of commands of the form 'measure atompos':

measure atompos id <atom index>
measure atompos var <variable name>
measure atompos sel

Measures the position of a single atom by
* defining the atom index directly by using the 'id' keyword
* obtaining atom indiex from a variable by using the 'var' keyword
* obtaining multiple atom indices from a selection by using the 'sel' keyword

Output (single atom measurement):
t(<x>, <y>, <z>)
Output (multiple atom measurement):
1. [<atom index>, <atom ID>, <resname>]: (<x>, <y>, <z>)
2. [<atom index>, <atom ID>, <resname>]: (<x>, <y>, <z>)
     .
     .
     .
```

(continues on next page)

(continued from previous page)

```
N. [<atom index>, <atom ID>, <resname>]: (<x>, <y>, <z>)
where N is the number of selected atoms.
```

6.18.3 bondcount

```
Overview of commands of the form 'measure bondcount':

measure bondcount <atom ID>

Counts bonds connected to a given atom type, specified by <atom ID> (e.g., H, O, C7), and
presents a histogram that shows the number of atoms (with atom ID) with 0, 1, 2, .., >=9
bond connections.

Output:
1. n<=0 n=1 n=2 n=3 n=4 n=5 n=6 n=7 n=8 n>=9
2. <count n<=0> <count n=1> <count n=2> <count n=3> <count n=4> <count n=5> <count n=6>
↪<count n=7> <count n=8> <count n>=9>
```

6.18.4 bondlength

```
Overview of commands of the form 'measure bondlength':

measure bondlength id <atom index 1> <atom index 2>
measure bondlength var <variable name>

Measures the length between two atoms by
* defining two atom indices directly by using the 'id' keyword
* obtaining atom indices from a variable by using the 'var' keyword

Output:
R(<atom index 1>, <atom index 2>) = <distance>
```

6.18.5 bondlengthdyn

```
Overview of commands of the form 'measure bondlengthdyn':

measure bondlengthdyn <DCD filename> <frame from> <frame to> <atom indices list (each␣
↪pair defines a bond)>

Measures distances between atom pairs defined in <atom indices list>, which is comma
separated with no spaces alowed. E.g., <atom indices list>=0,3,5,6 will yield a
measurement of distances between atoms 0 and 3, as well as betwen atoms 5 and 6. More
distances are added by extending the list. The distances are measured from the DCD
file, <DCD filename>, for each frame between <frame from> and <frame to>. Thus, a
distance plot, as function of time, is provided for each selected sistance to measure.

Output:
```

(continues on next page)

(continued from previous page)

```
1. <distance 1> <distance 2> ... <distance n>
2. <distance 1> <distance 2> ... <distance n>
         .
         .
         .
N. <distance 1> <distance 2> ... <distance n>
where N is the number of selected frames from the DCD file and n is the number of pairs
of atoms in <atom indices list>, where the order is the same as in <atom indices list>.
```

6.18.6 bondsep

```
Overview of commands of the form 'measure bondsep':

measure bondsep id <atom index 1> <atom index 2>
measure bondsep var <variable name>

Measures the number of bondsbetween two atoms by
* defining two atom indices directly by using the 'id' keyword
* obtaining atom indices from a variable by using the 'var' keyword

Output:
* BondSep(<atom index 1>, <atom index 2>) = <distance>
or
* BondSep(<atom index 1>, <atom index 2>) > 4
if the number of bonds exceeds 4.
```

6.18.7 center

```
Overview of commands of the form 'measure center':

measure center sel

Measures the geometric center of the collection of atoms defined by
* the visual selection of atoms, by using the 'sel' keyword

Output:
(<x_center>, <y_center>, <z_center>)
```

6.18.8 charge

```
Overview of commands of the form 'measure charge':

measure charge sel
measure charge tot

Measures the total charge of the collection of atoms defined by
* the visual selection of atoms, by using the 'sel' keyword
```

(continues on next page)

(continued from previous page)

```
* all the atoms, by using the 'tot' keyword
If 'sel' or 'tot' is not specified, 'tot' is assumed.

Output:
Qsum = <summed charge>
```

6.18.9 coulombenergy

```
Overview of commands of the form 'measure coulombenergy':

measure coulombenergy single [1to4coeffs <coeff 1> <coeff 2> <coeff 3> <coeff 4>]
measure coulombenergy dihedralrot <dihedral> rot <start angle> <end angle> <angular step␣
↪size> [1to4coeffs <coeff 1> <coeff 2> <coeff 3> <coeff 4>]
measure coulombenergy anglerot <angle> rot <start angle> <end angle> <angular step size>␣
↪[1to4coeffs <coeff 1> <coeff 2> <coeff 3> <coeff 4>]
measure coulombenergy bondstretch <bond> stretch <start dist> <end dist> <dist step size>
↪ [1to4coeffs <coeff 1> <coeff 2> <coeff 3> <coeff 4>]
measure coulombenergy allframes [1to4coeffs <coeff 1> <coeff 2> <coeff 3> <coeff 4>]

Measures the Coulomb energy using one of the following choices
* 'single':      sum Coulomb energies between all atoms, N, by (i=0..N, j=i+1..N)
                 with output 'Etot = <energy> kJ/mol'.
* 'dihedralrot': performs a 'single' for dihedral angles, specified by the
                 <dihedral>, <start angle>, <end angle>, <angular step size>
                 parameters. The output is a two column space sepatated list
                 with header 'Angle [deg] Etot [kJ/mol]'.
* 'anglerot':    performs a 'single' for molecular angles, specified by the
                 <angle>, <start angle>, <end angle>, <angular step size>
                 parameters. The output is a two column space sepatated list
                 with header 'Angle [deg] Etot [kJ/mol]'.
* 'bondstretch': performs a 'single' for bond stretching, specified by the
                 <bond>, <start dist>, <end dist>, <dist step size>
                 parameters. The output is a two column space sepatated list
                 with header 'Dist [AA] Etot [kJ/mol]'.
* 'allframes':   performs a 'single' for all the loaded frames and produces
                 a two column output of the form 'Frame Etot [kJ/mol].'

The <dihedral> parameter specifies the dihedral to rotate and can be formatted
as shown below.
* id <atom index 1> <atom index 2> <atom index 3> <atom index 4>
* var <variable name>

The <angle> parameter specifies the angle to rotate and can be formatted as
shown below.
* id <atom index 1> <atom index 2> <atom index 3>
* var <variable name>

The <bond> parameter specifies the bond to stretch and can be formatted as
shown below.
* id <atom index 1> <atom index 2>
```

(continues on next page)

(continued from previous page)

```
* var <variable name>

The '1to4coeffs' keyword can be applied to assign different weights to the
Coulomb energy calculations that are separated with 1, 2, 3 and 4 bonds,
respectively, by assigning values to <coeff 1> through <coeff 4>.
```

6.18.10 coulombpotential

```
Overview of commands of the form 'measure coulombpotential':

measure coulombpotential single at <x> <y> <z>
measure coulombpotential dihedralrot at <x> <y> <z> <dihedral> rot <angle start> <angle␣
↪end> <angular step size>

Measures the Coulomb potential, at a given point, of a test charge, +1,
using one of the following choices
* 'single':      find coulomb energy at <x> <y> <z>, with output 'Utot =
                  <energy> kJ/C'.
* 'dihedralrot': find coulomb energy at <x> <y> <z> for several dihedral
                 rotations between <angle start> and <angle end> with steps
                 <angular step size> (in degrees), where the rotation is
                 applied to the specified dihedral, <dihedral>.

The <dihedral> parameter specifies the dihedral to rotate and can be formatted
as shown below.
* id <atom index 1> <atom index 2> <atom index 3> <atom index 4>
* var <variable name>
```

6.18.11 count

```
Overview of commands of the form 'measure count':

measure count sel
measure count tot

Counts number of atoms. Either the total count can be measured, using 'tot',
or only the selected atoms can be measured, using 'sel'. If 'tot' or 'sel' are
not specified, the default behavior is 'tot'.

Output:
Nsum = <atom count>
```

6.18.12 dihedral

```
Overview of commands of the form 'measure dihedral':

measure dihedral <dihedral>

Measures the dihedral angle, given by the <dihedral> parameter. This
parameter specifies the dihedral to rotate and can be formatted
as shown below.
* id <atom index 1> <atom index 2> <atom index 3> <atom index 4>
* var <variable name>

Output:
Phi(<dih. index 1>, <dih. index 2>, <dih. index 3>, <dih. index 4>) = <angle>
where <angle> is given in degrees.
```

6.18.13 overlap

```
Overview of commands of the form 'measure overlap':

measure overlap <within> sel

Counts number of overlaping atoms within the current frame by counting the
number of atoms that are closer than the distance <within>. The count is
done for all atoms. Specifying the 'sel' keyword will select one atom of
every overlapping pair of atoms.

Output:
1. Total number of overlaps within <within> = N
2.
3. Atom i  Atom j  Type i  Type j  r_ij
4. ------------------------------------------------------------------------
5. <index i> <index j> <atom ID i> <atom ID j> <distance from i to j>
6. <index i> <index j> <atom ID i> <atom ID j> <distance from i to j>
       .
       .
       .
N+4. <index i> <index j> <atom ID i> <atom ID j> <distance from i to j>
where N is the number of identified overlaps.
```

6.18.14 pbc

```
Overview of commands of the form 'measure pbc':

measure pbc [<frame index>]

Measures the current periodic bounary conditions of frame <frame index>.
The frame index is, by default, the current frame index (i.e., the visible
frame), in case <frame index> is omitted.
```

(continues on next page)

(continued from previous page)

```
Output:
1. x = [<x low>, <x high>]
2. x = [<y low>, <y high>]
3. x = [<z low>, <z high>]
If the PBC is user defined, this will be notified by the message 'User
defined PBC!', in addition to the above.
```

6.18.15 radiusofgyration

```
Overview of commands of the form 'measure radiusofgyration':

measure radiusofgyration sel

Measures the radius of gyration for the collection of atoms defined by
* the visual selection of atoms, by using the 'sel' keyword
If 'sel' is not specified, 'sel' is assumed.

Note that atomic masses must be loaded for this command to work!

Output:
Rgyr = <radius of gyration>
```

6.19 calculate

Perform a calculation

```
Overview of commands of the form 'calculate <sub command>':

calculate com <selection>
calculate densitymap <DCD filename> <stride> <span> <last frame to load> <num bins>
↪<atoms (comma sep, no space)> <direction>
calculate densityprofile <DCD filename> <frame from> <frame to> <atoms (comma sep, no␣
↪space)> <vec. from> <vec. to> <num bins> [within simbox <vec. low> <vec. high>, within␣
↪cylinder <radius>]
calculate dihedraldistr <DCD filename> <frame from> <frame to> <num bins> indices <index␣
↪1> <index 2> <index 3> <index 4>
calculate dihedraldistr <DCD filename> <frame from> <frame to> <num bins> atomtypes
↪<atom ID 1> <atom ID 2> <atom ID 3> <atom ID 4> <bond cutoff r> [ignorepbc]
calculate dihedraldistrcom <DCD filename> <frame from> <frame to> <num bins> <num COM␣
↪bins> <COM dir> <COM range> indices <index 1> <index 2> <index 3> <index 4>␣
↪[ignorepbc] indices <M> <COM mol index 1> ... <COM mol index M>
calculate dihedraldistrcom <DCD filename> <frame from> <frame to> <num bins> <num COM␣
↪bins> <COM dir> <COM range> indices <index 1> <index 2> <index 3> <index 4>␣
↪[ignorepbc] atomtypes <M> <COM mol atom ID 1> ... <COM mol atom ID M>
calculate dihedraldistrcom <DCD filename> <frame from> <frame to> <num bins> <num COM␣
↪bins> <COM dir> <COM range> atomtypes <atom ID 1> <atom ID 2> <atom ID 3> <atom ID 4>
↪<bond cutoff r> [ignorepbc] indices <M> <COM mol index 1> ... <COM mol index M>
calculate dihedraldistrcom <DCD filename> <frame from> <frame to> <num bins> <num COM␣
```

(continues on next page)

(continued from previous page)

```
↪bins> <COM dir> <COM range> atomtypes <atom ID 1> <atom ID 2> <atom ID 3> <atom ID 4>
↪<bond cutoff r> [ignorepbc] atomtypes <M> <COM mol atom ID 1> ... <COM mol atom ID M>
calculate dipmom <DCD filename> <frame index> <list of atomic IDs (comma sep., no space)>
↪ [chargedmol]
calculate dipmomperturbcharge <DCD filename> <frame index> <list of atomic IDs>
↪<positive charges> <negative charges> <num perturbations> <delta Q> [chargedmol]
calculate dipmomperturbzpos <DCD filename> <frame index> <list of atomic IDs> <atoms to␣
↪displace> <delta Z> <num perturbations> [chargedmol]
calculate dipmomperturbzposexchange <DCD filename> <frame index> <list of atomic IDs>
↪<positive charges> <negative charges> <delta Z pos. charges> <delta Z neg. charges>␣
↪[chargedmol]
calculate dipmomprofile <DCD filename> <frame from> <frame to> <molecule resname> <x-dir.
↪vec> <y-dir.vec> <z-dir.vec> <num. bins> <min. dist.> <max. dist.> [chargedmol]␣
↪[forcecomplete <num. atoms in complete molecule>]
calculate distprobabilitycom <DCD filename> <frame from> <frame to> <num dist. bins>
↪<start dist. range> <end dist. range> <num COM bins> <COM direction> <start COM range>
↪<end COM range> <atom IDs from> <atom IDs to> indices <M> <COM base index 1> ... <COM␣
↪base index M> [ignorepbc]
calculate distprobabilitycom <DCD filename> <frame from> <frame to> <num dist. bins>
↪<start dist. range> <end dist. range> <num COM bins> <COM direction> <start COM range>
↪<end COM range> <atom IDs from> <atom IDs to> atomtypes <M> <COM base ID 1> ... <COM␣
↪base ID M> [ignorepbc]
calculate energybetween nonbonded <comma sep. list of atom IDs> <FF index (zero based)>
calculate energybetween coulomb <comma sep. list of atom IDs>
calculate energybetween bond <comma sep. list of atom IDs> <FF index (zero based)>
calculate energybetween angle <comma sep. list of atom IDs> <FF index (zero based)>
calculate energybetween dihedral <comma sep. list of atom IDs> <FF index (zero based)>
calculate energyoftranslation <xs> <ys> <zs> <xe> <ye> <ze> <steps>
calculate fft <ASCII file with 1 or more columns> <index of columns with real numbers>
↪<index of column with imaginary numbers> <direction> [zeropad]
calculate forcebetween nonbonded <comma sep. list of atom IDs> <FF index (zero based)>␣
↪[numerical]
calculate forcebetween coulomb <comma sep. list of atom IDs>
calculate forcebetween bond <comma sep. list of atom IDs> <FF index (zero based)>␣
↪[numerical]
calculate forcebetween angle <comma sep. list of atom IDs> <FF index (zero based)>␣
↪[numerical]
calculate forcebetween dihedral <comma sep. list of atom IDs> <FF index (zero based)>␣
↪[numerical]
calculate hbondcount <DCD filename> <frame from> <frame to> stride <stride> <M> <h-bond␣
↪crit 1> ... <h-bond crit M> [pbcfromvisual] [searchtype <type>] [nopbc]
calculate hbondcount <DCD filename> <frame from> <frame to> <M> <h-bond crit 1> ... <h-
↪bond crit M> [pbcfromvisual] [searchtype <type>] [nopbc]
calculate loading <DCD filename> <frame from> <frame to> <atom IDs to find loading for>
↪<lower vector - loading region> <upper vector - loading region>
calculate msd <DCD filename> <frame from> <frame to> resname <resname> [usegeomcent]␣
↪[numshiftsint0 <num shifts>]
calculate paircorrelation <DCD filename> <frame from> <frame to> <atom ID 1> <atom ID 2>
↪<num bins> <min. dist> <max. dist> [ignorepbc] [ignoredistbelow <dist>]
calculate potenergymap <DCD filename> <frame from> <frame to> <atom IDs> <list of␣
↪applied force fields> <Nx> <Ny> <cutting plane> <cutting plane pos>
calculate qbal <group of atoms to modify>
```

(continues on next page)

(continued from previous page)

```
calculate vacf <DCD filenmae> <frame from> <frame to> <time step (fs)> <VACF length>␣
↪name <atom IDs (comma sep., no space)>
calculate vacf <DCD filenmae> <frame from> <frame to> <time step (fs)> <VACF length> sel
calculate vdos <DCD filename> <frame from> <num. bits> <time step (fs)> [com] name <atom␣
↪IDs (comma sep., no space)>
calculate vdos <DCD filename> <frame from> <num. bits> <time step (fs)> [com] sel
calculate volumefromdensity <target density in kg/m^3> <atomic masses in g/mol> <num.␣
↪molecules>

To get more information about a <sub command>, type 'help calculate <sub command>
```

6.19.1 com

```
Overview of commands of the form 'calculate com':

calculate com <selection>

Calculates the center of mass (COM) of the specified selection. The possible selections␣
↪are:
* sel - calculate the COM of the selected atoms.
```

6.19.2 densitymap

```
Overview of commands of the form 'calculate densitymap':

calculate densitymap <DCD filename> <stride> <span> <last frame to load> <num bins>
↪<atoms (comma sep, no space)> <direction>

Calculates density maps of a specified atomic selection. Each density map that is␣
↪generated will be
a count taken over <span> frames of the specified DCD file, centered at an index, t. The␣
↪index, t,
runs over all frames of the DCD file, up to <last frame to load>, where each index is␣
↪separated by <stride>
frames. I.e., the number of generated density maps are approximately <last frame to load>
↪ / <stride>.
Each map is generated by specifying a plane, given by <direction> in {xy, yz, zx}, for␣
↪which all atoms
are projected into. The atoms specified by the comma separated list of atom IDs (i.e.,
↪<atoms>, s.a.,
O, H, C5, etc.) are counted within every bin of the defined plane, where each side of␣
↪the plane are
divided into <num bins> equisized bins. The output is formatted as described below.

First a header, consisting of three lines are output:
1. x(t,n,val) y(t,n,val) xbin(t,n,val) ybin(t,n,val) bincnt(t,n,val) ...
2. <t>        <t>        <t>           <t>           <t>              ...
3. <Nx>       <Ny>       <num bins>^2  <num bins>^2  <num bins>^2    ...
The first line describes each column that will follow in the data section, located␣
```

(continues on next page)

(continued from previous page)

```
↪immediately below the
header. The ellipsis (...) denotes that the columns are repeated for each output frame,
↪<t>, which is
counted over <span>/2 on each side. Note that if the span is outside the available␣
↪frames of the DCD
file, then these frames are simply ignored. The number of points (i.e., the length of␣
↪the data columns
below) are given by <Nx> and <Ny>., while the total number of bins in the specified␣
↪plane is given by the
square of <num bins>. The data section is formatted as:
4. <x> <y> <x center pos. of bin containing (x,y)> <y center pos. of bin containing (x,
↪y)> <bin count of bin containing (x,y) ...
5. <x> <y> <x center pos. of bin containing (x,y)> <y center pos. of bin containing (x,
↪y)> <bin count of bin containing (x,y) ...
                                    .
                                    .
                                    .
N+3. <x> <y> <x center pos. of bin containing (x,y)> <y center pos. of bin containing (x,
↪y)> <bin count of bin containing (x,y) ...
where N is the number of selected atoms.
```

6.19.3 densityprofile

```
Overview of commands of the form 'calculate densityprofile':

calculate densityprofile <DCD filename> <frame from> <frame to> <atoms (comma sep, no␣
↪space)> <vec. from> <vec. to> <num bins> [within simbox <vec. low> <vec. high>, within␣
↪cylinder <radius>]

Calculates the density profile from an atomic selection, sepcified by the list of atom␣
↪IDs,
<atoms>, s.a., O, H, C5, etc (no cammas). The atomic positions are taken from the␣
↪specified
DCD file, from <frame from> to <frame to>. The density profile is calculated along the␣
↪vector
V = <vec. to> - <vec. from>, where <vec. from> and <vec. to>, both are specified on the␣
↪form
<x> <y> <z>. The density profile is divided into <num bins> between these two points.␣
↪Optionally,
the 'within' keyword can be used to specify a cutoff for which atoms are not included␣
↪into the
density profile. If simbox is used, then atoms outside the lower corner <vec. low> and␣
↪the upper
corner <vec. high> (both of the form <x> <y> <z>), are not counted. If cylinder is used,␣
↪then all
atoms outside the cylinder of radius <radius> with its core along V are ignored. If␣
↪within is not
specified, then all atoms are counted. The output is specified below.

Note that there are three types of calculated densities from this command, these are
```

(continues on next page)

(continued from previous page)

```
* Particle density
* Mass density
* Charge density

The output is as follows, starting with a single header line,
1. abs_x abs_y abs_z rel_x rel_y rel_z #/frm m/frm Q/frm
2. <x abs> <y <bs> <z abs> <x rel> <y rel> <z rel> <particle density> <mass density>
↪<charge density>
3. <x abs> <y <bs> <z abs> <x rel> <y rel> <z rel> <particle density> <mass density>
↪<charge density>
                .
                .
                .
N+1. <x abs> <y <bs> <z abs> <x rel> <y rel> <z rel> <particle density> <mass density>
↪<charge density>
where N is the number of bins. Relative positions (rel) are relative to <vec. from>,␣
↪while absolute
positions (abs) are relative to the simulation box. Mass density is in [g/mol], while␣
↪charge density
is in units of partial charges.
```

6.19.4 dihedraldistr

```
Overview of commands of the form 'calculate dihedraldistr':

calculate dihedraldistr <DCD filename> <frame from> <frame to> <num bins> indices <index␣
↪1> <index 2> <index 3> <index 4>
calculate dihedraldistr <DCD filename> <frame from> <frame to> <num bins> atomtypes
↪<atom ID 1> <atom ID 2> <atom ID 3> <atom ID 4> <bond cutoff r> [ignorepbc]

Calculates the dihedral distribution, based on the contents of the <DCD filename> DCD␣
↪file,
from frame index <frame from> to frame index <frame to>. The number of bins to divide␣
↪the full
360 degree angle into is defined by <num bins>. It is possible to either define four atom
indices (zero based) and hence study the conformational distributions of a single␣
↪dihedral
over time, or to specify four atom IDs (e.g., H, O, C5) and study a specific type of␣
↪dihedral
over time (i.e., several dihedrals of the same type, per frame). In the latter case, one␣
↪also
need to specify the bond cutoff radius, <bond cutoff r>, which defines how long a bond␣
↪should
be before it is no longer considered a bond. By default, bonds reach across periodic␣
↪boundaries
(PBC), but it is possible to only consider bonds within the PBC by specifying 'ignorepbc
↪'. The
output from the calculation is given below.

The output starts with the following header:
```

(continues on next page)

(continued from previous page)

```
1. Num Bins: <applied number of bins>
2. From frame: <starting frame index>
3. To frame: <to frame index>
4. Bond cutoff: <applied bond cutoff>
5. Dihedral: <specified dihedral that was studied>
6. Index Angle Tot.Count Count/Frm

Following the header is the data section:
7. <bin index> <angle in degrees> <number of entries in this bin> <number of entries per␣
↪frame in this bin>
8. <bin index> <angle in degrees> <number of entries in this bin> <number of entries per␣
↪frame in this bin>
                        .
                        .
                        .
N+6. <bin index> <angle in degrees> <number of entries in this bin> <number of entries␣
↪per frame in this bin>
where N is the number of bins.
```

6.19.5 dihedraldistrcom

```
Overview of commands of the form 'calculate dihedraldistrcom':

calculate dihedraldistrcom <DCD filename> <frame from> <frame to> <num bins> <num COM␣
↪bins> <COM dir> <COM range> indices <index 1> <index 2> <index 3> <index 4>␣
↪[ignorepbc] indices <M> <COM mol index 1> ... <COM mol index M>
calculate dihedraldistrcom <DCD filename> <frame from> <frame to> <num bins> <num COM␣
↪bins> <COM dir> <COM range> indices <index 1> <index 2> <index 3> <index 4>␣
↪[ignorepbc] atomtypes <M> <COM mol atom ID 1> ... <COM mol atom ID M>
calculate dihedraldistrcom <DCD filename> <frame from> <frame to> <num bins> <num COM␣
↪bins> <COM dir> <COM range> atomtypes <atom ID 1> <atom ID 2> <atom ID 3> <atom ID 4>
↪<bond cutoff r> [ignorepbc] indices <M> <COM mol index 1> ... <COM mol index M>
calculate dihedraldistrcom <DCD filename> <frame from> <frame to> <num bins> <num COM␣
↪bins> <COM dir> <COM range> atomtypes <atom ID 1> <atom ID 2> <atom ID 3> <atom ID 4>
↪<bond cutoff r> [ignorepbc] atomtypes <M> <COM mol atom ID 1> ... <COM mol atom ID M>

Calculates the dihedral distribution, based on the contents of the <DCD filename> DCD␣
↪file,
from frame index <frame from> to frame index <frame to>. The number of bins to divide␣
↪the full
360 degree angle into is defined by <num bins>. It is possible to either define four atom
indices (zero based) and hence study the conformational distributions of a single␣
↪dihedral
over time, or to specify four atom IDs (e.g., H, O, C5) and study a specific type of␣
↪dihedral
over time (i.e., several dihedrals of the same type, per frame). In the latter case, one␣
↪also
need to specify the bond cutoff radius, <bond cutoff r>, which defines how long a bond␣
↪should
be before it is no longer considered a bond. By default, bonds reach across periodic␣
```

(continues on next page)

(continued from previous page)

```
↪boundaries
(PBC), but it is possible to only consider bonds within the PBC by specifying 'ignorepbc
↪'.

In the output from this calculation, the bins will be identified by the center of mass␣
↪(COM)
positions of the molecules containing the dihedral angles that are counted. Thus, the␣
↪COM molecule
must be identified, as well as the range of COM position that are to be binned and the␣
↪number of
such bins. This is done through either specifying the 'indices' or 'atomtypes', together␣
↪with the
number of indices or molecules, <M>, and the indices, <COM mol index i>, or atom IDs␣
↪(such as H,
O, C6), <COM mol atom ID i>. The COM bins are defined by <num COM bins> <COM range> (of␣
↪the form
<start pos> <end pos>), as well as the axis along which to perform binning <COM dir> in
↪{x, y, z}.

The output from the calculation is given below.

The output starts with the following header:
1. Num angle Bins: <applied number of bins>
2. Num COM Bins: <applied number of bins>
3. Start COM range: <applied start pos>
4. End COM range: <applied end pos>
5. COM range direction (0=x,1=y,2=z): <applied direction>
6. From frame: <starting frame index>
7. To frame: <to frame index>
8. Bond cutoff: <applied bond cutoff>
9. Dihedral: <specified dihedral that was studied>
10. COMPos Angle Tot.Count Normalized

Following the header is the data section:
11. <COM pos> <angle in degrees> <number of entries in this bin> <normalized num.␣
↪entries, max is unity>
12. <COM pos> <angle in degrees> <number of entries in this bin> <normalized num.␣
↪entries, max is unity>
                        .
                        .
                        .
Na*Nb+10. <COM pos> <angle in degrees> <number of entries in this bin> <normalized num.␣
↪entries, max is unity>
where Na is the number of COM bins and Nb is the number of angle bins (i.e., this␣
↪constitutes.
a surface plot with angle along one axis and molecular COM along the other).
```

6.19.6 dipmom

```
Overview of commands of the form 'calculate dipmom':

calculate dipmom <DCD filename> <frame index> <list of atomic IDs (comma sep., no space)>
↪ [chargedmol]

Calculates the dipole moment of the selected molecules <list of atomic IDs>. The
dipolemoment is averaged based on all the defined molecules in frame index given by
<frame index>. Either one can choose a DCD file through <DCD filename> as input, or
it is possible to use the loaded frames as input by letting <DCD filename> =
__fromloadedframes__. The dipole moment expression for neutral molecules are used as
default. By sepcifying 'chargedmol', the dipole moment expression for charged molecules
is employed. The output is defined below.

Output:
Px Py Pz
<dipole moment x-component> <dipole moment y-component> <dipole moment z-component>
```

6.19.7 dipmomperturbcharge

```
Overview of commands of the form 'calculate dipmomperturbcharge':

calculate dipmomperturbcharge <DCD filename> <frame index> <list of atomic IDs>
↪<positive charges> <negative charges> <num perturbations> <delta Q> [chargedmol]

Calculates the dipole moment of the selected molecules <list of atomic IDs> (comma
separated, no space). The dipolemoment is averaged based on all the defined molecules
in frame index given by <frame index>. The DCD file, <DCD filename>, is used as input.
The dipole moment expression for neutral molecules are used as default. By sepcifying
'chargedmol', the dipole moment expression for charged molecules is employed. The dipole
moment is perturbed through <num perturbations> steps, where the positive and negative
charges are all increased and decreased, respectively, by <delta Q> spread over the
respective groups of charges, thus preserving charge neutrality. The greatest␣
↪perturbation
is calculated first. Note that the lists of positive and negative charges are lists of
atomic IDs, such as H, O and C7 (comma separated, no space).

Output:
1. QPerturb+ QPerturb- Px Py Pz
2. <positive perturbation> <negative perturbation> <dipole moment x-component> <dipole␣
↪moment y-component> <dipole moment z-component>
3. <positive perturbation> <negative perturbation> <dipole moment x-component> <dipole␣
↪moment y-component> <dipole moment z-component>
                .
                .
                .
N+1. <positive perturbation> <negative perturbation> <dipole moment x-component> <dipole␣
↪moment y-component> <dipole moment z-component>
where N is the number of perturbations.
```

6.19.8 dipmomperturbzpos

```
Overview of commands of the form 'calculate dipmomperturbzpos':

calculate dipmomperturbzpos <DCD filename> <frame index> <list of atomic IDs> <atoms to␣
↪displace> <delta Z> <num perturbations> [chargedmol]

Calculates the dipole moment of the selected molecules <list of atomic IDs> (comma
separated, no space). The dipolemoment is averaged based on all the defined molecules
in frame index given by <frame index>. The DCD file, <DCD filename>, is used as input,
or the loaded frames if <DCD filename> = __fromloadedframes__. The dipole moment␣
↪expression
for neutral molecules are used as default. By sepcifying 'chargedmol', the dipole moment
expression for charged molecules is employed. The dipole moment is perturbed through
<num perturbations> steps, where the <atomd to displace> (comma separated, no space)
are all displaced by the z-position <delta Z>. Note that <atoms to displace> is a lists␣
↪of
atomic IDs, such as H, O and C7.

Output:
1. Step Px Py Pz Rel.z.pos
2. <step> <dipole moment x-component> <dipole moment y-component> <dipole moment z-
↪component> <Z pos displacement>
3. <step> <dipole moment x-component> <dipole moment y-component> <dipole moment z-
↪component> <Z pos displacement>
                .
                .
                .
N+1. <step> <dipole moment x-component> <dipole moment y-component> <dipole moment z-
↪component> <Z pos displacement>
where N is the number of perturbations.
```

6.19.9 dipmomperturbzposexchange

```
Overview of commands of the form 'calculate dipmomperturbzposexchange':

calculate dipmomperturbzposexchange <DCD filename> <frame index> <list of atomic IDs>
↪<positive charges> <negative charges> <delta Z pos. charges> <delta Z neg. charges>␣
↪[chargedmol]

Calculate dipole moment while incrementally increasing the number of ion-swaps to
perform. Swapping the ions will naturally keep the system charge neutral. The ion
indices to exchange are chosen at random. I.e., in the first iteration a random
ion-pair is chosen. Then, in the next iteration, the first pair remains the same
while the second pair is randomly chosen. An ion pair is constructed by first
selecting an index from the positive ions and then finding the negative ion
which is the closest to the chosen positive ion. The the dipole moment is calculated
for the selected molecules <list of atomic IDs> (comma separated, no space). The
dipolemoment is averaged based on all the defined molecules in frame index given by
<frame index>. The DCD file, <DCD filename>, is used as input, or the loaded frames if
<DCD filename> = __fromloadedframes__. The dipole moment expression for neutral molecules
```

(continues on next page)

(continued from previous page)

```
are used as default. By sepcifying 'chargedmol', the dipole moment expression for charged
molecules is employed. The lists of positive and negative charges are lists of atomic␣
↪IDs,
such as H, O and C7 (comma separated, no space), which defines groups of positive and
negative charges that can form the aforementioned ion pairs to swap (i.e., ions are␣
↪picked
from these groups). The <delta Z pos. charges> and <delta Z neg. charges> parameters
define the z-axis displacement to perform on the positive and negative charges,
respectively.

Output:
1. #Exchanges Px Py Pz Exch.index
2. <number of performed exchanges> <dipole moment x-component> <dipole moment y-
↪component> <dipole moment z-component> <Index of the positive ion>
3. <number of performed exchanges> <dipole moment x-component> <dipole moment y-
↪component> <dipole moment z-component> <Index of the positive ion>
                  .
                  .
                  .
N+1. <number of performed exchanges> <dipole moment x-component> <dipole moment y-
↪component> <dipole moment z-component> <Index of the positive ion>
N+2.              .
N+3.              .
             <N atomic system configurations, for each exchange step that has been␣
↪performed, in the xyz-file format>
M.                .
M+1.              .
where N is the number of perturbations.
```

6.19.10 dipmomprofile

```
Overview of commands of the form 'calculate dipmomprofile':

calculate dipmomprofile <DCD filename> <frame from> <frame to> <molecule resname> <x-dir.
↪vec> <y-dir.vec> <z-dir.vec> <num. bins> <min. dist.> <max. dist.> [chargedmol]␣
↪[forcecomplete <num. atoms in complete molecule>]

Calculates the dipole moment of the selected molecules from <molecule resname> (string
with resname). The dipolemoment is averaged based on all the defined molecules
in frames <frame from> to <frame to>. The DCD file, <DCD filename>, is used as input.

The dipole moment expression for neutral molecules are used as default. By sepcifying
'chargedmol', the dipole moment expression for charged molecules is employed. A profile
of the dipolemoment, for the defined molecule type, is created along the unit vector
created from the vector (<x-dir.vec>, <y-dir.vec>, <z-dir.vec>) from the distance <min.
dist.> to the distance <max. dist.>, where the distances are molecular center of mass
(COM) distances that are projected along the specified direction unit vector.The
projected COM for each molecule where the dipole moment is calculated will be binned into
<num. bins> bins between the specified minimum and maximum distances. Each bin will␣
↪contain
```

(continues on next page)

(continued from previous page)

```
an average dipole moment from all molecules belonging to that bin. By default, the␣
↪algorithm
will not check if a molecule is wrapped accross periodic boundaries, thus resulting in
incomplete molecules being processed. This can be fixed by applying the 'forcecomplete'
keyword with the number of atoms to expect in a molecule, <num. atoms in complete␣
↪molecule>.
Molecules that do not conform to the specified atom count will not be included in the
dipole moment profile.

Output:

1. From frame: <first frame to include in average>
2. To frame: <last frame to include in average>
3. Resname: <resname for selected molecules>
4. Range: [<min. dist>, <max. dist>]
5. Num. bins: <num. bins>
6. Rc projection: <UnitVec(<x-dir.vec>, <y-dir.vec>, <z-dir.vec>)>
7. Distance Count P_x P_y P_z P_abs
8. <projected COM distance for bin (center of bin)> <num bin entries> <dipole moment x-
↪component> <dipole moment y-component> <dipole moment z-component> <absolute dipole␣
↪moment>
9. <projected COM distance for bin (center of bin)> <num bin entries> <dipole moment x-
↪component> <dipole moment y-component> <dipole moment z-component> <absolute dipole␣
↪moment>
                .
                .
                .
N+7. <projected COM distance for bin (center of bin)> <num bin entries> <dipole moment x-
↪component> <dipole moment y-component> <dipole moment z-component> <absolute dipole␣
↪moment>
where N is the number of bins.
```

6.19.11 distprobabilitycom

```
Overview of commands of the form 'calculate distprobabilitycom':

calculate distprobabilitycom <DCD filename> <frame from> <frame to> <num dist. bins>
↪<start dist. range> <end dist. range> <num COM bins> <COM direction> <start COM range>
↪<end COM range> <atom IDs from> <atom IDs to> indices <M> <COM base index 1> ... <COM␣
↪base index M> [ignorepbc]
calculate distprobabilitycom <DCD filename> <frame from> <frame to> <num dist. bins>
↪<start dist. range> <end dist. range> <num COM bins> <COM direction> <start COM range>
↪<end COM range> <atom IDs from> <atom IDs to> atomtypes <M> <COM base ID 1> ... <COM␣
↪base ID M> [ignorepbc]

Calculates a two dimensional surface plot with the molecular center of mass (COM) along
one axis. The other axis is the length between atoms of the set defined by <atom IDs␣
↪from>
(comma separated, no space) and the set defined by <atom IDs to>. Each of the axis are
binned according to <num COM bins> and <num dist. bins>, respectively. The range of the
```

(continues on next page)

(continued from previous page)

```
two axis are defined by [<start COM range>, <end COM range>] and by [<start dist. range>,
<end dist. range>], respectively. The <COM direction> can be either x, y or z and defines
which of the components of the COM vector are to be binned. The set of atoms that is used
as basis for the COM calculations are defined by specifying the number of atoms, <M>,
followed by the atom indices <COM base index i>, if 'indices' is specified, or by the
atom IDs (e.g., H, O, C7) <COM base ID i>, if 'atomtypes' is specified. By default the
periodic boundary conditions (PBC) are taken into account (i.e., distances are measured
across PBC boundaries. However, PBCs can be ignored by using the 'ignorepbc' keyword.

Trajectories used as input is specified by <DCD filename>, where all frames from <frame␣
↪from>
to <frame to> are used in the averaging (i.e., contributing to counts in the 2D grid of
specified bins).

Output:
1. Num distance bins: <num dist. bins>
2. Start dist. range: <start dist. range>
3. End dist. range: <end dist. range>
4. Num COM bins: <num COM bins>
5. Start COM range: <start COM range>
6. End COM range: <end COM range>
7. COM range direction (0=x,1=y,2=z): <COM direction>
8. From frame: <frame from>
9. To frame: <frame to>
10. Dist from: <atom IDs from>
11. Dist to: <atom IDs to>
12. COMPos Dist Tot.Count Normalized
13. <COM positio binn> <distance between atoms bin> <total count in bin> <normalized␣
↪count, max set to unity>
14. <COM positio binn> <distance between atoms bin> <total count in bin> <normalized␣
↪count, max set to unity>
                .
                .
                .
Na*Nb+12. <COM positio binn> <distance between atoms bin> <total count in bin>
↪<normalized count, max set to unity>
where Na and Nb are the number of COM bins and distance bins, respectively.
```

6.19.12 energybetween

```
Overview of commands of the form 'calculate energybetween':

calculate energybetween nonbonded <comma sep. list of atom IDs> <FF index (zero based)>
calculate energybetween coulomb <comma sep. list of atom IDs>
calculate energybetween bond <comma sep. list of atom IDs> <FF index (zero based)>
calculate energybetween angle <comma sep. list of atom IDs> <FF index (zero based)>
calculate energybetween dihedral <comma sep. list of atom IDs> <FF index (zero based)>

Calculates the nonbonded, Coulomb or bond energy (in kJ/mol) betwee two atoms, based on␣
↪the
```

(continues on next page)

(continued from previous page)

```
configured force fields, or the angular energy between three atoms or the dihedral energy
between four atoms. The energies are calculated based on the atoms shown in the current␣
↪frame.
The indices of the various force fields that have been created or loaded is available␣
↪through
the 'list' command. An index into these lists, for the appropriate force field type, is␣
↪specified
through the <FF index (zero based)> parameter. The energy is calculated between the␣
↪atoms given
in <comma sep. list of atom IDs> (e.g., H, O, C7, where the list must be enetered␣
↪withour spece).
```

6.19.13 energyoftranslation

```
Overview of commands of the form 'calculate energyoftranslation':

calculate energyoftranslation <xs> <ys> <zs> <xe> <ye> <ze> <steps>

Calculates the total non-bonded energy of the system shown in the current frame for␣
↪repeated
translations of the selected atoms in the current frame. The selected atoms are␣
↪translated along
from start vector, (<xs>, <ys>, <zs>) to the end vector, (<xe> <ye> <ze>), in <step>␣
↪steps. The
non-bonded force fields must be loaded or created before running this command.

Output:
1. Position ENon-bond
2. <x> <y> <z> <total system energy (kJ/mol)>
3. <x> <y> <z> <total system energy (kJ/mol)>
                .
                .
                .
N+1. <x> <y> <z> <total system energy (kJ/mol)>
where N is the number of steps. The position (<x>, <y>, <z>) are the positions between␣
↪the
start vector and the end vector.
```

6.19.14 fft

```
Overview of commands of the form 'calculate fft':

calculate fft <ASCII file with 1 or more columns> <index of columns with real numbers>
↪<index of column with imaginary numbers> <direction> [zeropad]

Calculates the FFT (Fast Fourier Transform) of a list of complex numbers, taken from an␣
↪ASCII file
containing several coulumns of data (separated using one or more space characters). The␣
↪columns are
```

(continues on next page)

(continued from previous page)

```
chosen by their zero base indices. One or two columns can be selected, one for the real␣
↪part of the
complex numners and one for the imaginary part. In case of only real numbers, then the␣
↪imaginary
column index can be set to -1. The <direction> parameter can be either 'fwd' or 'rev',␣
↪depending on
if a forward or reverse Fourier transform is to be calculated, respectively.

By invoking 'zeropad' the input dataset is zero padded to achieve a size 2^M, where M is␣
↪an integer.

Output:
1. Re Im
2. <transformed real number> <transformed imaginary number>
3. <transformed real number> <transformed imaginary number>
            .
            .
            .
N+1. <transformed real number> <transformed imaginary number>
where N is the number of entries in the FFT transformed dataset.
```

6.19.15 forcebetween

```
Overview of commands of the form 'calculate forcebetween':

calculate forcebetween nonbonded <comma sep. list of atom IDs> <FF index (zero based)>␣
↪[numerical]
calculate forcebetween coulomb <comma sep. list of atom IDs>
calculate forcebetween bond <comma sep. list of atom IDs> <FF index (zero based)>␣
↪[numerical]
calculate forcebetween angle <comma sep. list of atom IDs> <FF index (zero based)>␣
↪[numerical]
calculate forcebetween dihedral <comma sep. list of atom IDs> <FF index (zero based)>␣
↪[numerical]

Calculates the nonbonded, Coulomb or bond force (in kJ/(mol Angstrom)) betwee two atoms,␣
↪based on
the configured force fields, or the angular force between three atoms or the dihedral␣
↪force
between four atoms. The forces are calculated based on the atoms shown in the current␣
↪frame.
The indices of the various force fields that have been created or loaded is available␣
↪through
the 'list' command. An index into these lists, for the appropriate force field type, is␣
↪specified
through the <FF index (zero based)> parameter. The force is calculated between the atoms␣
↪given
in <comma sep. list of atom IDs> (e.g., H, O, C7, where the list must be enetered␣
↪withour spece).
By sepcifying 'numerical', the forces are calculated using numerical differentiation.
```

6.19.16 hbondcount

```
Overview of commands of the form 'calculate hbondcount':

calculate hbondcount <DCD filename> <frame from> <frame to> stride <stride> <M> <h-bond␣
↪crit 1> ... <h-bond crit M> [pbcfromvisual] [searchtype <type>] [nopbc]
calculate hbondcount <DCD filename> <frame from> <frame to> <M> <h-bond crit 1> ... <h-
↪bond crit M> [pbcfromvisual] [searchtype <type>] [nopbc]

Calculates hydrogen bond statistics from frames <frame from> to <frame to> within the␣
↪DCD file specified
by <DCD filenmae>. <M> hydrogen bond criteria are specified in the <h-bond crit i>␣
↪parameters. Each such
parameter is given as a comma separated list with no spaces in-between. There are 7␣
↪entries in the comma
separated lists
* the donor atom to consider (e.g., O, O4, C7)
* the hydrogen atom to consider
* the acceptor atom to consider
* the distance criteria between donor and hydrogen
* the distance criteria between hydrogen and acceptor
* the distance criteria between donor and acceptor
* the angle criteria, donor-hydrogen-acceptor
Both distance and angular criteria can be specified as '-', which means they are ignored.
↪ It is possible to
load every <stride> frame from the DCD file, which can be useful for large DCD files. By␣
↪default, the periodic
boundary conditions (PBC) are taken from each frame in the DCD file. However, by␣
↪specifying 'pbcfromvisual'
the PBC is collected from the one active in the 3D view. Using 'nopbc' ignores PBCs. By␣
↪default, the bonds
are searched by considereing only intermolecular hydrogen bonds. By using the 'searchtype
↪' parameter, it is
possible to specify
* <type>=intermolecular: only consider intermolecular bonds
* <type>=intramolecular: only consider intramolecular bonds
* <type>=allhbonds: consider all types of bonds

Output:
1. [1] D:<donor> H:<hydrogen> A:<acceptor> d-h:<dist.crit. donor-hydrogen> h-a:<dist.
↪crit. hydrogen-acceptor> d-a:<dist.crit. donor-acceproror> d-h...a:<angle crit>
2. [2] D:<donor> H:<hydrogen> A:<acceptor> d-h:<dist.crit. donor-hydrogen> h-a:<dist.
↪crit. hydrogen-acceptor> d-a:<dist.crit. donor-acceproror> d-h...a:<angle crit>
            .
            .
            .
N. [N] D:<donor> H:<hydrogen> A:<acceptor> d-h:<dist.crit. donor-hydrogen> h-a:<dist.
↪crit. hydrogen-acceptor> d-a:<dist.crit. donor-acceproror> d-h...a:<angle crit>
N+1. Index Tot.count Num.Donors Num.Accept. Num.Conn.Don. Num.Conn.Acc. Frame FirstDon.␣
↪FirstHydr. FirstAcc
N+2. <index> <tot.count> <num. donors> <num. acceptors> <num h-bonds conn to donors>
↪<num h-bonds conn to acceptors> <frame index> <first donor in frame> <first hydrogen␣
↪in frame> <first acceptor in frame>
```

(continues on next page)

(continued from previous page)

```
N+3. <index> <tot.count> <num. donors> <num. acceptors> <num h-bonds conn to donors>
↪<num h-bonds conn to acceptors> <frame index> <first donor in frame> <first hydrogen␣
↪in frame> <first acceptor in frame>
            .
            .
            .
N+K+1. <index> <tot.count> <num. donors> <num. acceptors> <num h-bonds conn to donors>
↪<num h-bonds conn to acceptors> <frame index> <first donor in frame> <first hydrogen␣
↪in frame> <first acceptor in frame>
where N is the number of hydrogen bond criteria and K is the number of analyzed frames.
```

6.19.17 loading

```
Overview of commands of the form 'calculate loading':

calculate loading <DCD filename> <frame from> <frame to> <atom IDs to find loading for>
↪<lower vector - loading region> <upper vector - loading region>

Calculates the loading curve from <frame from> to <frame to> of the atoms specified in
<atom IDs to find loading for> within the DCD file given by <DCD filename>. For each
frame, which is one point on the curve, the number of atoms from the list of atom IDs
(comma separated list without spaces, e.g., H,O,C7) that are within the loadin region
are counted. The loading region is specified by a lower and upper vector, which both
are entered as three numbers, <x> <y> <z>, separated by a space.

Output:
1. <list of atoms for which loading is calculated>
2. <loading frame 1>
3. <loading frame 2>
      .
      .
      .
N+1. <loading frame N>
where N is the number of included frames.
```

6.19.18 msd

```
Overview of commands of the form 'calculate msd':

calculate msd <DCD filename> <frame from> <frame to> resname <resname> [usegeomcent]␣
↪[numshiftsint0 <num shifts>]

Calculates the mean square displacement (MSD) of the molecules that has the resname
↪<resname>
between frames <frame from> to <frame to>. By default, the MSD is calculated based on the
center of mass of the tracked molecules (which requires masses of the atoms to be␣
↪loaded).
However, by using the 'usegeomcent' keyword, all masses are set to unity, thus resulting
in the geometric center of the molecules being used as basis for the MSD.
```

(continues on next page)

(continued from previous page)

```
To enable better MSD averaging, the 'numshiftsint0' keyword kan be used to specify <num
shifts> number of shifts to apply to the starting frame, t0. Hence, if <num shifts>=0, no
shifts are applied, if <num shifts>=10, then the MSD is calculated 10 times but with the
starting frame being moved 0, 1, 2 and up to 10 steps. The average MSD is calculated␣
↪between
all the shifted calculations.

Output:
1. Num. shifts in t0 = <num shifts>
2. From frame = <frame from>
3. To frame = <frame to>
4. Index MSD
5. <index> <MSD>
6. <index> <MSD>
          .
          .
          .
N+4. <index> <MSD>
where N is the number of points in the MSD curve.
```

6.19.19 paircorrelation

```
Overview of commands of the form 'calculate paircorrelation':

calculate paircorrelation <DCD filename> <frame from> <frame to> <atom ID 1> <atom ID 2>
↪<num bins> <min. dist> <max. dist> [ignorepbc] [ignoredistbelow <dist>]

Calculates the pair correlation function, averaged over frames <frame from> to <frame to>
↪,
taken from the DCD file <DCD filename>. Pair correlation is calculated between atoms with
ID <atom ID 1> and <atom ID 2> (e.g., H, O, C7). The pair correlation function is␣
↪calculated
for distances between <min. dist> and <max. dist>, divided into <num bins>. By default␣
↪periodic
boundary conditions (PBCs) are used, thus letting the distance measurements go across␣
↪these
boundaries. To prevent distance measurements across PBCs, use the 'ignorepbc' keyword.␣
↪If it
is desirable to only include distances above a distance criteria, <dist>, in the pair␣
↪correlation
function, the 'ignoredistbelow' keyword can be applied.

Ouptut:
1. From frame: <frame from>
2. To frame: <frame to>
3. Pair: <atom ID1>-<atom ID2>
4. Range: [<min. dist>, <max. dist>]
5. Num. bins: <num bins>
6. Ignore dist. below: <dist>
```

(continues on next page)

(continued from previous page)

```
7. Use PBC: yes/no
8.
9. Distance Count PairCorr RDF IdGas.RDF
10. <distance> <num. atoms in bin> <pair correlation value> <radial distribution␣
↪function (RDF) value> <ideal gas RDF value>
11. <distance> <num. atoms in bin> <pair correlation value> <radial distribution␣
↪function (RDF) value> <ideal gas RDF value>
        .
        .
        .
N+9. <distance> <num. atoms in bin> <pair correlation value> <radial distribution␣
↪function (RDF) value> <ideal gas RDF value>
where N is the number of applied bins.
```

6.19.20 potenergymap

```
Overview of commands of the form 'calculate potenergymap':

calculate potenergymap <DCD filename> <frame from> <frame to> <atom IDs> <list of␣
↪applied force fields> <Nx> <Ny> <cutting plane> <cutting plane pos>

Calculates a 2D potential energy map by utilizing a test particle of charge +1, averaged␣
↪over
frames from <frame from> to <frame to> in the DCD file specified by <DCD filename>. Only␣
↪the
atoms listed in <atom IDs> (e.g., H, O, C7) yield a contribution to the map. The <atom␣
↪IDs>
list is to be comma separated without any spaces. The <list of applied forcefields> can␣
↪be
* coulomb
This list is also to be comma separated, without any spaces. <Nx> and <Ny> defines the␣
↪number
of bins to include in the 2D map. The 2D map is calculated for a <cutting plane> which␣
↪can be
* XY
* YZ
* ZX
where the cutting plane position is <cutting plane pos>. If XY, then the cutting plane␣
↪position
is along Z, if YZ, then it is along X and if ZX, then it is along Y.

Output:
1. PlanePrinted : <cutting plane>(<0 if XY, 1 if YZ and 2 if ZX)
2. Selection : <atom ID 1> <atom ID 2> ... <atom ID n>
3. Size : (<Nx>, <Ny>)
4. DensityData={{<point x>, <point y>, <pot. E>}, {<point x>, <point y>, <pot. E>}, ...,
↪{<point x>, <point y>, <pot. E>}};
where <point x> and <point y> are points in the defined cutting plane.
```

6.19.21 qbal

```
Overview of commands of the form 'calculate qbal':

calculate qbal <group of atoms to modify>

Given a loaded dataset, this command will calculate scaling factors for the atoms within␣
↪the given set of
atoms to modify. The <group of atoms to modify> parameter can be one of the following:
* sel - the group of atoms to modify is the current visual selection
The calculated scaling factors for the charges are such that, if applied to the␣
↪corresponding charges, the
total charge of the loaded system is charge balanced. Moreover, the scaling factors are␣
↪minimized, such that
they are as small as possible (given the usual L2-norm).

More detailed information about calculating the scaling factors is given in Refs. 1 and␣
↪2 (see below).

Output:
1. Charge 'Multiply with' 'Old charge' 'New charge' 'Change in %'
2. <charge type> <correction factor> <old charge value> <charge value after␣
↪multiplication with correction factor> <change in % from old charge>
3. <charge type> <correction factor> <old charge value> <charge value after␣
↪multiplication with correction factor> <change in % from old charge>
         .
         .
         .
N+1. <charge type> <correction factor> <old charge value> <charge value after␣
↪multiplication with correction factor> <change in % from old charge>
where N is the number of charge types (e.g., H, O, C8) within the group of atoms to␣
↪modify.


[1] Olsen, R. and Kvamme, B. (2019) 'Effects of glycol on adsorption dynamics of␣
↪idealized water droplets on LTA-3A zeolite surfaces', AIChE Journal, 65(5), p. e16567.␣
↪doi: 10.1002/aic.16567.
[2] Olsen, R. et al. (2016) 'Effects of Sodium Chloride on Acidic Nanoscale Pores␣
↪Between Steel and Cement', The Journal of Physical Chemistry C, 120(51), pp.␣
↪29264-29271. doi: 10.1021/acs.jpcc.6b10043.
```

6.19.22 vacf

```
Overview of commands of the form 'calculate vacf':

calculate vacf <DCD filenmae> <frame from> <frame to> <time step (fs)> <VACF length>␣
↪name <atom IDs (comma sep., no space)>
calculate vacf <DCD filenmae> <frame from> <frame to> <time step (fs)> <VACF length> sel

Calculates the velocity auto correlation function (VACF) for the DCD file, <DCD filename>
↪,
```

(continues on next page)

(continued from previous page)

```
between the start frame, <frame from>, to the frame <frame to>. The time step, <time␣
↪step>,
given in units of femtoseconds (fs), is used to obtain numerically calculated velocities
from pairs of atomic position originating from two adjacent time frames. The <VACF␣
↪length>
parameter is the length of the VACF, in number of time steps, and must be smaller or␣
↪equal
to D = <frame to> - <frame from>. If <VACF length> < D, then a VACF is calculated for␣
↪each
starting time t0 from t0=<frame from> to t0=<frame to> (where indices outside the number
of frames are ignored). All collected VACF are then averaged. The VACF is only calculated
for a given selection of atoms. Either, based on the atom 'name', where a list, <atom␣
↪IDs>,
is supplied (e.g., H, O, C7), or based on the visual selection of atoms, achieved through
the 'sel' keyword.

Output:
1. t vacf norm(vacf)
2. <VACF time point in fs> <VACF value> <VACF value, normalized to the first value>
3. <VACF time point in fs> <VACF value> <VACF value, normalized to the first value>
       .
       .
       .
N+1. <VACF time point in fs> <VACF value> <VACF value, normalized to the first value>
where N is the number of points selected in the VACF (<VACF length>).
```

6.19.23 vdos

```
Overview of commands of the form 'calculate vdos':

calculate vdos <DCD filename> <frame from> <num. bits> <time step (fs)> [com] name <atom␣
↪IDs (comma sep., no space)>
calculate vdos <DCD filename> <frame from> <num. bits> <time step (fs)> [com] sel

Calculates the vibrational density of states (VDOS) for the DCD file, <DCD filename>,
between the start frame, <frame from>, to the frame <frame from>+2^<num. bits>. The time
step, <time step>, given in units of femtoseconds (fs), is used to obtain numerically
calculated velocities that are needed within the calculations. VDOS is only calculated␣
↪for a
given selection of atoms. Either, based on the atom 'name', where a list, <atom IDs>,
is supplied (e.g., H, O, C7), or based on the visual selection of atoms, achieved through
the 'sel' keyword. If the 'com' keyword is selected, the velocity of the center of mass␣
↪of
the selected atoms is calculated, otherwise the average velocity of the selected atoms␣
↪is used.

Output:
1. omega[rad/fs] vdos_o freq[x10^15Haz] vdos_f lambda[nm] vdos_l k[cm^-1] vdos_k
2. <angular frequency> <VDOS value> <frequency> <VDOS value> <wavelength> <VDOS value>
↪<wavenumber> <VDOS value>
```

(continues on next page)

(continued from previous page)

```
3. <angular frequency> <VDOS value> <frequency> <VDOS value> <wavelength> <VDOS value>
↪<wavenumber> <VDOS value>
       .
       .
       .
N+1. <angular frequency> <VDOS value> <frequency> <VDOS value> <wavelength> <VDOS value>
↪<wavenumber> <VDOS value>
where N is the length of the VDOS function.
```

6.19.24 volumefromdensity

```
Overview of commands of the form 'calculate volumefromdensity':

calculate volumefromdensity <target density in kg/m^3> <atomic masses in g/mol> <num.␣
↪molecules>

Given a target density in kg/m^3, a list of atomic masses (comma separated with no␣
↪space) for
a single atom in the system, as well as the number of such molecules in the system, the␣
↪required
volume in Angstrom^3 is calculated, in addition to the required side length required if␣
↪the system
was cubic.
```

6.20 gauss9

Operations related to the Gaussian9 SW package

```
Usage: gauss9 dihedralrot id <atom index 1> <atom index 2> <atom index 3> <atom index41>␣
↪rot <start angle> <end angle> <angular step size> qmspec <base set> <charge> <spin␣
↪multiplicity> [relaxed]
       gauss9 dihedralrot var <variable name> rot <start angle> <end angle> <angular␣
↪step size> qmspec <base set> <charge> <spin multiplicity> [relaxed]
       gauss9 anglerot id <atom index 1> <atom index 2> <atom index 3> rot <start angle>
↪<end angle> <angular step size> qmspec <base set> <charge> <spin multiplicity>␣
↪[relaxed]
       gauss9 anglerot var <variable name> rot <start angle> <end angle> <angular step␣
↪size> qmspec <base set> <charge> <spin multiplicity> [relaxed]
       gauss9 bondstretch id <atom index 1> <atom index 2> stretch <start dist> <end␣
↪dist> <dist step size> qmspec <base set> <charge> <spin multiplicity> [relaxed]
       gauss9 bondstretch var <variable name> stretch <start dist> <end dist> <dist step␣
↪size> qmspec <base set> <charge> <spin multiplicity> [relaxed]
       gauss9 genxyzfrominput <gaussian input file name>
       gauss9 genoptefromoutput <gaussian output file name>
       gauss9 genoptxyzfromoutput <gaussian output file name>
       gauss9 genxyzfromoutput <gaussian output file name>

This command can execute several operations that are useful for the Gaussian9 software␣
```

(continues on next page)

(continued from previous page)

```
↪package. These are listed in the following:
* dihedralrot: will generate a Gaussian9 input script that executes electronic structure␣
↪calculations on the loaded system
               for each rotation of a dihedral, either specified through 4 atomic␣
↪indices or through the same indices stored
               in a variable.
* anglerot:    will generate a Gaussian9 input script that executes electronic structure␣
↪calculations on the loaded system
               for each rotation of an angle, either specified through 3 atomic indices␣
↪or through the same indices stored
               in a variable.
* bondstretch: will generate a Gaussian9 input script that executes electronic structure␣
↪calculations on the loaded system
               for each atomic distance separation, either specified through 2 atomic␣
↪indices or through the same indices
               stored in a variable.
* genxyzfrominput: Extracts the atom names and coordinates from a Gaussian9 input file␣
↪and outputs a XYZ-file formatted text.
* genoptefromoutput: Extracts the optimization energies from a Gaussian9 output file and␣
↪produces a list of these.
* genoptxyzfromoutput: Extracts the optimized structures from a Gaussian9 output file␣
↪and produces XYZ-formatted text outputs.
* genxyzfromoutput: Extracts the input structure from a Gaussian9 output file and␣
↪produces XYZ-formatted text outputs.
```

6.21 lammps

Operations related to the LAMMPS SW package

```
Usage: lammps genff
       lammps gendata [bondacrosspbc]

This command provides features that relate to version 5 Sep 2014 version of the LAMMPS MD
simulator, as well as other versions that support the same input / output structure.

The 'genff' sub-command will create a LAMMPS text that sets up the specified force field.
This should be piped to a file (i.e.,genff > <ff file>).

The 'gendata' sub-command will create a LAMMPS text that sets up all atomic positions,␣
↪atomic IDs,
bond definitions, angle definitions, etc. This should be piped to a file (i.e.,gendata >
<data file>). To make sure bonds are created across PBCs, use the 'bondacrosspbc'␣
↪keyword.

The generated input files can now be used as a basis for creating the final input files.␣
↪The
<ff file> and <data file> should be used as input to the main input file for LAMMPS.
```

6.22 hoomdblue

Operations related to the HOOMD-blue SW package

```
Usage: hoomdblue genff
       hoomdblue gendata [bondacrosspbc]
       hoomdblue genrun <data file> <ff file> <temperature (K)> <time step (fs)>␣
↪[enable12and13]
       hoomdblue atmtolj <value in atm units>
       hoomdblue fstolj <value in fs units>
       hoomdblue Ktolj <value in K units>

This command provides features that relate to version 1.3.1 of the HOOMD-blue MD␣
↪simulator,
as well as other versions that support the same input / output structure. The 'atmtolj',
'fstolj' and 'Ktolj' sub-commands are used to convert values given in atm, fs and K,
respectiviely, to the Lennard-Jones units used by the sub-commands that generate inpu␣
↪files
for HOOMD-blue.

The 'genff' sub-command will create a HOOMD-blue Python text that sets up the specified␣
↪force
field. This should be piped to a Python file (i.e.,genff > <ff file>.py).

The 'gendata' sub-command will create an XML text that sets up all atomic positions,␣
↪atomic IDs,
bond definitions, angle definitions, etc. This should be piped to a XML file (i.e.,
↪gendata >
<data file>.xml). To make sure bonds are created across PBCs, use the 'bondacrosspbc'␣
↪keyword.

Once 'genff' and 'gendata' has generated the <ff file>.py and <data file>.xml, the
↪'genrun'
sub-command can be applied to generate a Python run script for HOOMD-blue. This should␣
↪be piped
to a Python file (i.e., genrun <data file> <ff file> <temp> <step> > run.py). To enable␣
↪1-2 and
1-3 interactions in HOOMD-blue, use the 'enable12and13' keyword.

The generated input files can now be used as a basis for creating the final input files.
```

6.23 mmol

*Apply operations to *.mmol type files*

```
Usage: mmol toscript [noself] [mixgeom] <molfile 1> <molfile 2> ... <molfile N> >
↪<script file to generate>

This command imports a series of *.mmol files, which are molecular definition
files defined within the MDynaMix MD software package, and converts them to a
series of MolTwister exec(), script instructions that will set up the force
```

(continues on next page)

(continued from previous page)

```
fields defined within the *.mmol files. The list of input molfiles (*.mmol)
is terminated by the pipe symbol, '>'.

Self interactions can be exculded in the MolTwister instructions by appyling the
'noself' keyword. By default, mixing of short range interaction parameters is done
by using arithmetic mixing rules. However, by employing the 'mixgeom' keyword,
geometric mixing rules will be applied.
```

6.24 dcd

Perform a DCD file operation

```
Overview of commands of the form 'dcd <sub command>':

dcd atomicunwrap <DCD filename> [atomnames <atom ID list>] [pbcfromvisual]
dcd header <DCD filename>
dcd numcoordinates <DCD filename> <record index>
dcd numrecords <DCD filename>
dcd readcoordinate <DCD filename> <record index> <coordinate index>
dcd readrecord <DCD filename> <record index>
dcd unwrap <DCD filename> [pbcfromvisual]
dcd wrap <DCD filename> [pbcfromvisual]
dcd fromxtc <XTC filename> <DCD filename> <Num. time steps>

To get more information about a <sub command>, type 'help dcd <sub command>
```

6.24.1 atomicunwrap

```
Overview of commands of the form 'dcd atomicunwrap':

dcd atomicunwrap <DCD filename> [atomnames <atom ID list>] [pbcfromvisual]

Loads a DCD file, <DCD filename>, and performs an atomic unwrap across the
periodic boundaries (i.e., PBC). The PBC is taken from the DCD file by default.
However, by specifying the 'pbcfromvisual' keyword, the PBC can be taken from
the applied visual PBC. It is also possible to select a subsection of the atoms
stored in the original DCD file by applying the 'atomnames' keyword with a list
of atom IDs (e.g., H, O, C7), <atom ID list>, which is comma separated with no
space. The output is a DCD file with the name
* <DCD filename (without extension)>_mtatunwrap.dcd

Note that an 'unwrap' will move the atomic coordinates across the PBC if it
is discovered that the coordinate crosses it between two time steps (thus leading
to a complete unfolding of the PBC. However, an 'atomic unwrap' will unwrap
frame-by-frame, based on the molecular definitions. Hence, only molecules that
wrap across the PBC will be unwrapped so as to be un-divided, but outside the
PBC. Hence. an atomic unwrap requires only one frame, while unwrap needs more.
```

6.24.2 header

```
Overview of commands of the form 'dcd header':

dcd header <DCD filename>

Prints the DCD file header from <DCD filename>.

Output:
ID = <ID>
NSets = <number of sets>
InitStep = <init step>
WrtFreq = <Write frequency>
TimeStep = <time step>
DescriptA = <descript A>
DescriptB = <descript B>
NAtoms = <number of atoms>
```

6.24.3 numcoordinates

```
Overview of commands of the form 'dcd numcoordinates':

dcd numcoordinates <DCD filename> <record index>

Prints the number of coordinates in record <record index>
of the DCD file, <DCD filename>.

Output:
Coordinate count = <coordinate count>
```

6.24.4 numrecords

```
Overview of commands of the form 'dcd numrecords':

dcd numrecords <DCD filename>

Prints the number of recoords within the DCD file, <DCD filename>.

Output:
Num records = <record count>
```

6.24.5 readcoordinate

```
Overview of commands of the form 'dcd readcoordinate':

dcd readcoordinate <DCD filename> <record index> <coordinate index>

Prints the coordinate at record index <record index>, coordinate index
<coordinate index> within the DCD file, <DCD filename>.

Output:
Coordinate = (<x>, <y>, <z>)
```

6.24.6 readrecord

```
Overview of commands of the form 'dcd readrecord':

dcd readrecord <DCD filename> <record index>

Prints all the coordinates at record index <record index>
within the DCD file, <DCD filename>.

Output:
{{<x1>, <y1>, <z1>}, {<x2>, <y2>, <z2>}, ..., {<xN>, <yN>, <zN>}}
where N is the number of coordinates within the given record.
```

6.24.7 unwrap

```
Overview of commands of the form 'dcd unwrap':

dcd unwrap <DCD filename> [pbcfromvisual]

Loads a DCD file, <DCD filename>, and performs an unwrap across the periodic
boundaries (i.e., PBC). The PBC is taken from the DCD file by default. However,
by specifying the 'pbcfromvisual' keyword, the PBC can be taken from the applied
visual PBC. The output is a DCD file with the name
* <DCD filename (without extension)>_mtunwrap.dcd

Note that an 'unwrap' will move the atomic coordinates across the PBC if it
is discovered that the coordinate crosses it between two time steps (thus leading
to a complete unfolding of the PBC. However, an 'atomic unwrap' will unwrap
frame-by-frame, based on the molecular definitions. Hence, only molecules that
wrap across the PBC will be unwrapped so as to be un-divided, but outside the
PBC. Hence. an atomic unwrap requires only one frame, while unwrap needs more.
```

6.24.8 wrap

```
Overview of commands of the form 'dcd wrap':

dcd wrap <DCD filename> [pbcfromvisual]

Loads a DCD file, <DCD filename>, and performs a wrap across the periodic boundaries
(i.e., PBC). The PBC is taken from the DCD file by default. However, by specifying
the 'pbcfromvisual' keyword, the PBC can be taken from the applied visual PBC.
The output is a DCD file with the name
* <DCD filename (without extension)>_mtwrap.dcd
```

6.24.9 fromxtc

```
Overview of commands of the form 'dcd fromxtc':

dcd fromxtc <XTC filename> <DCD filename> <Num. time steps>

Converts an XTC file, <XTC filename>, to a DCD file with name <DCD filename>.
The number of time steps is not available in the XTC header. Hence, this must
be given in the <Num. time steps> argument (i.e., the total number of time
steps in the simulation).
```

6.25 mtpython

Runs Python script input, with MolTwister spesific functions

```
Usage: mtpython {<Python line of code>}

Any Python line of code can be executed, one by one. For example, the sequence:

mtpython {a = 5}
mtpython {b = 20}
mtpython {print("Output: %f" % (a+b))}

would produce the output 'Output: 25.000000'. Note that Python 'import' will load
the specified library and will be available for the next use of 'mtpython'.

By importing the 'moltwister' library (e.g., import moltwister as mt), several
Python functions will be made available that can query / manipulate the state of
MolTwister. These are as follows

        exec(<moltwister command>) : result as string
        Executes a moltwister command. For example, exec("list all"), which
        will return result of 'list all'.

        get_num_atoms() : result as int
        Returns the number of atoms presently loaded or added.
```

(continues on next page)

(continued from previous page)

```
        get_atom_pos(atomIndex:integer, axisIndex:integer) : result as integer
        Returns the atom position of a given atom index and a given axis (0=x, 1=y, 2=z).

        get_atom_type(atomIndex:integer) : result as string
        Returns the atom type of the atom at the given atom index.

        get_atom_mass(atomIndex:integer) : result as integer
        Returns the assigned atomic mass of the atom at the given atom index.

        get_atom_charge(atomIndex:integer) : result as integer
        Returns the assigned atomic charge of the atom at the given atom index.

        get_atom_resname(atomIndex:integer) : result as string
        Returns the assigned resname of the atom at the given atom index.

        get_atom_molindex(atomIndex:integer) : result as integer
        Returns the assigned molecular index of the atom at the given atom index.

        is_atom_sel(atomIndex:integer) : result as boolean
        Returns true if the atom at the given atom index is selected, else it returns␣
↪false.

        create_xyz_file(filePath:string) : no result
        Create an empty XYZ file.

        append_to_xyz_file(filePath:string, boxSizeX:float, boxSizeY:float,␣
↪boxSizeZ:float, convertToAU:bool, atomCoordinates:list) : no result
        Append list of [atomTypeString, x, y, z]-lists to XYZ file.

        create_dcd_file(filePath:string, numTimeSteps:int, stride:int, timeStep:float,␣
↪numAtoms:int) : no result
        Create a DCD file with given header information.

        append_to_dcd_file(filePath:string, boxSizeX:float, boxSizeY:float,␣
↪boxSizeZ:float, atomCoordinates:list) : no result
        Append list of [x, y, z]-lists to DCD file.

        begin_progress(progBarDescription:string) : no result
        Shows initial progress bar (in the command line shell) with given text.

        update_progress(step:integer, totalSteps:integer) : no result
        Updates progress bar according to given step information.

        end_progress() : no result
        Finishes progress bar, showing as 100 percent complete.

An example sequence could be:

mtpython {import moltwister as mt}
mtpython {print("Num atoms: %f" % mt.get_num_atoms())}
mtpython {print("%s" % mt.exec("list all"))}
```

(continues on next page)

(continued from previous page)

```
Note that all the above examples are written in the language of Python version 3.0
and may not be compatible with earlier versions of Python.

It is also possible to write a python script and then load this script using
the 'load python <name of script>' command, where only the <Python line of code>
parts of the mtpython commands are included within the script.
```

6.26 moldyn

Invoke a molecular dynamics simulation

```
Overview of commands of the form 'moldyn <sub command>':

moldyn optimizeenergy [cpu]
moldyn run [cpu]
moldyn ff bondforceprofile <ff index> <profile start r> <profile end f> <num points in␣
↪profile>
moldyn ff angleforceprofile <ff index> <profile start theta> <profile end theta> <num␣
↪points in profile>
moldyn ff dihedralforceprofile <ff index> <profile start theta> <profile end theta> <num␣
↪points in profile>
moldyn ff nonbondforceprofile <ff index> <profile start r> <profile end f> <num points␣
↪in profile>
moldyn cfg get
moldyn cfg set <parameter> <value>

To get more information about a <sub command>, type 'help moldyn <sub command>
```

6.26.1 optimizeenergy

```
Overview of commands of the form 'moldyn optimizeenergy':

moldyn optimizeenergy [cpu]

This command will run an energy optimization. If the 'cpu' keyword is included, the
optimization will be forced to be executed on the CPU. If not, an attempt will be made␣
↪to run the
optimization on GPU. If this does not succeed (e.g., if the software is not compiled to␣
↪run on GPU)
the optimization will be executed on CPU.
```

6.26.2 run

```
Overview of commands of the form 'moldyn run':

moldyn run [cpu]

This command will run a molecular dynamics (MD) simulation. If the 'cpu' keyword is␣
↪included, the
simulation will be forced to be executed on the CPU. If not, an attempt will be made to␣
↪run the
simulation on GPU. If this does not succeed (e.g., if the software is not compiled to␣
↪run on GPU)
the simulation will be executed on CPU.
```

6.26.3 ff

```
Overview of commands of the form 'moldyn ff':

moldyn ff bondforceprofile <ff index> <profile start r> <profile end f> <num points in␣
↪profile>
moldyn ff angleforceprofile <ff index> <profile start theta> <profile end theta> <num␣
↪points in profile>
moldyn ff dihedralforceprofile <ff index> <profile start theta> <profile end theta> <num␣
↪points in profile>
moldyn ff nonbondforceprofile <ff index> <profile start r> <profile end f> <num points␣
↪in profile>

The molecular dynamics algorithm will first create a list of points based on the force-
↪field
potential expressions. This list of points will be interpolated during the simulations to
calculate the forces that need to be appiled to each atom in the system.

This command will output the list of points that make up both the potential (in kJ/mol)␣
↪and
the force (in kJ/(mol*AA)). Depending on the type of potential we query (bond, angle,␣
↪dihedral
or non-boned), the start and end points to output must be specified, together with the␣
↪desired
number of points. The units for bond and non-bonded profiles are in Angstrom, while for␣
↪angle
and dihedral profiles they are in degrees. The force-field index, <ff index>, can be␣
↪found
through the 'list ff' command.
```

6.26.4 cfg

```
Overview of commands of the form 'moldyn cfg':

moldyn cfg get
moldyn cfg set <parameter> <value>

This command can be used both to set and get configurations for the molecular
dynamics (MD) simulator. If 'get' is called, the available parameters are listed
together with their current values. The first word in each listed line (followed
by '=') is the value to be handed to <parameter> when calling 'set'. In the list
obtained from 'get' it has also been made clear how <value> for the corresponding
parameter should be formatted.
```

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search