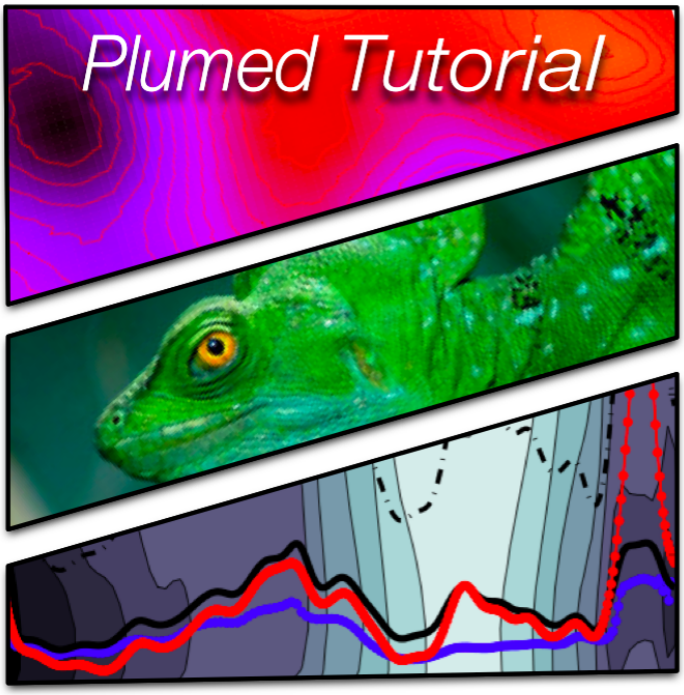
PLUMED Tutorial

A portable plugin for free-energy calculations with molecular dynamics



CECAM, Lausanne, Switzerland
September 28, 2010 - October 1, 2010

This document and the relative computer exercises have been written by:

Massimiliano Bonomi
Davide Branduardi
Giovanni Bussi
Francesco Gervasio
Alessandro Laio
Fabio Pietrucci

Tutorial website:
`http://sites.google.com/site/plumedtutorial2010/`

PLUMED website:
`http://www.plumed-code.org`

PLUMED users Google group:
`plumed-users@googlegroups.com`

PLUMED reference article:
M. Bonomi, D. Branduardi, G. Bussi, C. Camilloni, D. Provasi,
P. Raiteri, D. Donadio, F. Marinelli, F. Pietrucci, R.A. Broglia
and M. Parrinello, PLUMED: a portable plugin for free-energy calculations with m
Comp. Phys. Comm. 2009 vol. 180 (10) pp. 1961-1972.

Contents

Chapter 1

Compilation

1.1 PLUMED compilation

Here in the following we will show a bunch of codes that work with plumed. They have been chosen because they are all free codes and can be downloaded straight from the web without any license or agreement. FIrst we will show a classical code and secondly an ab-initio code. First some useful informations. You log in as user `tutoXX` (which stands for `tuto01`,`tuto02` etc...). You will be assigned a node, say `nodeYYY`. All the tests should be run in the node. Therefore you have open a terminal and access the node you have been assigned

```
ssh -X tutoXX@nodeYYY
```

and you are on the node. Change to the scratch directory

```
cd /scratch
```

If the directory `tutoXX` does not exist create it and enter it.

```
mkdir tutoXX
cd tutoXX
```

This is the place where you will run all the calculations. You have a maximum of **4 processors per terminal**. You will see later how to use them. All the exercises will be provided daily and you may find them into:

```
/nfs_home/tutoadmin/PLUMED/EXERCISES_FINAL
```

which are accessible read-only. You will find the directories

```
1_compiling  2_basics  3_biasing ...
```

and you will be told time to time which directory you should copy into your `/scratch/tutoXX` directory for the exercise as indicated by the tutor. Now copy the directory into your scratch

```
cd /scratch/tutoXX
cp -r /nfs_home/tutoadmin/PLUMED/EXERCISES_FINAL/1_compiling .
```

The documentation (a file `tutorial.pdf` with the practical lessons and the slides) are put in

```
/nfs_home/tutoadmin/PLUMED/DOCUMENTATION
```

and updated daily (and bugs solved: help us in improving it!). Check it from time to time when the tutor tells you.

Now let'us go back to the compiling: first enter the `1_compiling/plumed` directory and untar `PLUMED`

```
cd 1_compiling/plumed
tar -zxvf PLUMED-1.2.1.tar.gz
cd ../
```

1.1.1 Compile PLUMED with GROMACS

In this section we will discuss how to compile plumed in the CECAM machine. Our "reference" code is GROMACS (version 4.0.7) but other versions works in identical way. Please note that, as the number of codes supported by PLUMED is getting larger, we're trying at least to discontinue the old version of the code (version 3.3 in GROMACS).

What you need to compile a GROMACS version 4.0.7 on the ottokar cluster at cecam is similar to what you generally need on a standard Linux machine.

- c/c++ compilers are available (gcc/g++ are fine. Intel compilers on Linux boxes are also free for academic). We assume this is already installed on the machine as many Linux suites allow to download them directly as rpm packages.
- FFTW fast fourier libraries http://www.fftw.org/. 3.2.2 version is working nicely for most of cases. We assume this to be already installed on the system.
- Message Passing Interface (MPI) is installed (Openmpi or MPICH are freely available from http://www.open-mpi.org/). In several Linux distros you can download it as rpm packages so we assume this to be available on your system.
- GROMACS package (4.0.7) for this example (see http://www.gromacs.org).
- PLUMED distribution is available (probably at this stage you will have a 1.2.1 version) at the usual website http://www.plumed-code.org.

The installation procedure closely follows the GROMACS installation with the exception that, between configuration step

and compilation, you have to apply `PLUMED` patch. Let us revise step-by-step the procedure you have to carry out and let us assume that we work with a bash shell so that we set up variables accordingly.

First let's start by installing the serial version (double precision) and then we will proceed in installing the parallel version of mdrun. First we move to the installation directory of the exercises. We should have everything we need. We will make use of the Intel compilers here. Generally, neglecting the specific compiler variables, the default should be gcc. On `ottokar` cluster and on many HPC machines most of the useful environment variables are automatically provided via the `module` command.

```
myuser> module load intel-cc/10.1.015
myuser> module load fftw/3.2.2_intel-10.1.015
```

and we can conveniently set up some variables that we will use and reuse at runtime

```
myuser> fftw=/opt/fftw-3.2.2/intel-10.1.015
myuser> mycc=icc
myuser> myld=icc
myuser> mycxx=icpc
myuser> cd gromacs
myuser> mydir=‘pwd‘
```

Now we can unpack the GROMACS code and enter it. We also create a directory `install_dir` where all the final installation will be placed. It is rather convenient that you create your custom installation directory so you don't affect the installation of the other users (probably you'll be prevented to do this in any case, unless you do the installation as root user).

```
myuser> tar -zxvf gromacs-4.0.7.tar.gz
myuser> cd gromacs-4.0.7
myuser> mkdir install_dir
```

Now we proceed with the full configuration:

```
myuser>./configure LDFLAGS=-L${fftw}/lib
         CPPFLAGS=-I${fftw}/include
         --prefix=$mydir/gromacs-4.0.7/install_dir
         CC=$mycc LD=$myld
         CXX=$mycxx --with-fft=fftw3
         --disable-float --program-suffix=_d
```

Let us analyze what is done in this command. It first tells the `configure` utility where to get the fftw through the `LDFLAGS` and `CPPFLAGS`. Then it tells where the final installation should be done(`--prefix=$mydir/gromacs-4.0.7/install_dir`). This is the location where you will find libraries, executables, include-files, manuals and topologies for gromacs. The `CC,LD,CXX` specify the compiler you intend to use. In this case it will be Intel compilers. If you omit this the default gcc compiler will be used. The flag `--disable-float` specify to compile GROMACS in in double precision which is highly recommended for a correct `PLUMED` intallation . Then the `--program-suffix=_d` specifies that all the executables will have the suffix _d and this is needed so you know that the executables you use are double precision by default. Now it come the patching procedure itself. You first have to specify the `PLUMED` location by setting the environment variable `plumedir`.

```
myuser> export plumedir="my/path/to/plumed"
myuser> chmod +x $plumedir/patches/plumedpatch_gromacs_4.0.4.sh
myuser> $plumedir/patches/plumedpatch_gromacs_4.0.4.sh -patch
```

where `my/path/to/plumed` will look something like `/scratch/tutoXX/1_compilin`
Then have a look to what is happening. If the patching procedure goes fine you should have something like this.

```
* I will try to patch PLUMED version cvs ...
-- Executing pre script
-- Setting up symlinks
-- Setting up recon symlinks
-- Applying patches
patching file ./src/kernel/md.c
patching file ./src/kernel/repl_ex.c
Hunk #4 succeeded at 204 (offset 4 lines).
Hunk #5 succeeded at 220 (offset 4 lines).
Hunk #6 succeeded at 452 (offset 4 lines).
Hunk #7 succeeded at 481 (offset 4 lines).
Hunk #8 succeeded at 511 (offset 4 lines).
Hunk #9 succeeded at 569 (offset 4 lines).
Hunk #10 succeeded at 613 (offset 4 lines).
Hunk #11 succeeded at 682 (offset 4 lines).
patching file ./src/kernel/repl_ex.h
patching file ./src/kernel/Makefile
patching file ./src/kernel/mdrun.c
-- Executing post script
- DONE!
```

If you find some failures at this stage it is likely that the GROMACS download version changed somehow or you choose the wrong patch for this code. Please note that a successful patching does not mean a successful compiling. So please check carefully if you have errors during the compiling after the patching is done.

Now you are ready. Give it a go!

```
myuser> make
myuser> make install
```

Check that you have everything you need:

```
myuser> ls install_dir/bin
```

Now, assuming everything went fine, let us recompile `mdrun` only in double precision, with MPI support. First, let us clean and unpatch:

```
myuser> make clean
myuser> $plumedir/patches/plumedpatch_gromacs_4.0.4.sh -revert
```

Load the parallel compilers:

```
myuser> module load openmpi/1.2.6_intel-10.1.015
```

and now we can reconfigure and repatch:

```
myuser> ./configure LDFLAGS=-L${fftw}/lib
         CPPFLAGS=-I${fftw}/include
         --prefix=$mydir/gromacs-4.0.7/install_dir
         CC=$mycc LD=$myld
         CXX=$mycxx --with-fft=fftw3
         --disable-float --program-suffix=_mpid
         --enable-mpi
myuser> $plumedir/patches/plumedpatch_gromacs_4.0.4.sh -patch
myuser> make mdrun
myuser> make install-mdrun
```

Verify once again that everything went fine in patching and compilation. Now in the `install_dir/bin` directory you should have all the `*_d` executable and `mdrun_mpid` which you just compiled. Now, for having all the executables in your path you have to source GMXRC

```
myuser>source ./install_dir/bin/GMXRC
```

and now `mdrun_d` and `mdrun_mpid` should be in your path. Just chek it by doing

```
myuser>mdrun_d
myuser>mdrun_mpid
```

should work and GROMACS should print out its banner and die (no input file provided).

1.1.2 Compile `PLUMED` with QUANTUM ESPRESSO

Here we show how to compile `PLUMED` with the popular DFT package QUANTUM ESPRESSO (`http://www.quantum-espresso.org`) on the CECAM machine `ottokar`. Actual version of QUANTUM ESPRESSO is 4.2.1 and the plumed patch for version 4.2 works fine for the 4.2.1 version as well. First we load the environments for the Intel compilers and openmpi intel compilers. Now enter the directory `1_compiling/quantumespresso` and do

```
myuser> module purge
myuser> module load intel-cc/10.1.015
myuser> module load intel-fc/10.1.015
myuser> module load intel-mkl/10.0.1.014
myuser> module load openmpi/1.2.6_intel-10.1.015
```

Then we unpack the code and simply configure.

```
myuser> tar -zxvf espresso-4.2.1.tar.gz
myuser> cd espresso-4.2.1
myuser> ./configure
```

This should provide the basic configuration to install the clean QUANTUM ESPRESSO package.

```
myuser> export plumedir="my/path/to/plumed"
myuser> chmod +x ${plumedir}/patches/plumedpatch_qespresso_4.2.0.sh
myuser> ${plumedir}/patches/plumedpatch_qespresso_4.2.0.sh -patch
myuser> make pw
```

As usual, check if the patch produce any error. After few minutes you should have obtained a version of pw (you should have `./bin/pw.x`) that is ready and working.

1.1.3 Compile PLUMED with LAMMPS

Another interesting classical molecular dynamics code is LAMMPS. LAMMPS is a very scalable code intended primarily for solid state and mesoscopic simulations. It includes a variety of force field and functional forms and is fully customizable via an interface that allows you to accept/reject plumed as an addon to the code. You may download it from `http://lammps.sandia.gov` free of charge. Compiling LAMMPS can be non trivial and here we assume to do a very basic installation where the main flags have been setted up for the `ottokar` machine.

First, goto into the directory containing LAMMPS and untar it and load the modules that then you will need for compiling.

```
myuser> cd 1_compiling/lammps
myuser> tar -xvf lammps.tar.gz
myuser> cd lammps-24Sep10
myuser> module purge
myuser> module load intel-cc/10.1.015
myuser> module load intel-fc/10.1.015
myuser> module load fftw/2.1.5_intel-10.1.015
myuser> module load openmpi/1.2.6_intel-10.1.015
```

Now copy some Makefiles that have been specifically hacked to work on the machine you are using

```
myuser> cp  ../Makefile.icc.atc ./lib/atc/Makefile.icc
myuser> cp ../Makefile.openmpi ./src/MAKE/Makefile.openmpi
```

The very first thing one should do is to compile the modules that appears in the lib directory and that one intend to compile (all these Makefiles have been suitably hacked to run on the `ottokar` machine):

```
myuser> cd  ./lib/atc
myuser> make -f Makefile.icc
myuser> cd ../meam
myuser> make -f Makefile.ifort
myuser> cd ../poems
myuser> make -f Makefile.icc
myuser> cd ../reax
myuser> make -f Makefile.ifort
myuser>  cd ../../
```

These modules are additional packages that must compiled as libraries and you might need. We just skip the gpu package as we intend to use the code on a standard CPU based machine. At this stage you are ready for compiling the clean version and you should edit the most appropriate `Makefile.arch` in the `./src/MAKE` directory. In `ottokar` we use the `Makefile.openmpi` configuration file suitably adapted in few things (location of FFTW, location of Intel compilers libraries) that we will provide along with the installation. Overwrite it on `./src/MAKE/Makefile.openmpi` and you are done. Next step is patching the code as usual

```
myuser>  export plumedir="my/path/to/plumed"
```

```
myuser> chmod +x ${plumedir}/patches/plumedpatch_lammps_24-09-2010.s
myuser>  ${plumedir}/patches/plumedpatch_lammps_24-09-2010.sh -patch
```

At this stage we should have everything we need and we should proceed to a standard LAMMPS installation. We first install the modules we need

```
myuser> cd src
myuser> make yes-standard
myuser> make no-gpu
```

and check if `PLUMED` is already turned on (it should be done automatically by the patching procedure).

```
myuser> make  package-status
...
Installed YES: package USER-PLUMED
...
```

Now simply do:

```
myuser> make openmpi
```

and after some ages(!) you should get the executable `lmp_openmpi`.

Chapter 2

Basics: monitoring simulations

In this chapter we will see the basics instruction for creating a `PLUMED` input files and review the output that it produces during the simulation. At the end of this chapter the user should be able to write a simple plumed input file and monitor the CVs of choice by simply consulting the manual for each CV individual syntax. The philosophy behind `PLUMED` requires that one may use the same input with different programs. This is only partly true because different programs have different internal units. For example, in GROMACS, distances are in nm and energies are in kjoule/mol while in NAMD and SANDER the distances are in Å and the energies are in kcal/mol. Keep this in mind because it is crucial to understand which are the internal units of the program you are running before you do a simulation. It may let you save lots of human and computer time. Moreover, in all the CVs, one should specify the atoms involved in it. In different programs the atom indexing may be different. Therefore do not expect that a given torsion that you specify with a number of indexes is transferable from GROMACS to NAMD and viceversa.

Different programs call `PLUMED` in different ways. GROMACS for example calls it at runtime from the command line:

```
mdrun -plumed metadyn
```

for example tells mdrun to enable `PLUMED` and look its input in a file called `metadyn.dat`. The default extension `.dat` is automatically appended to the input file name. The screen output, that in GROMACS is `md.log` contains `PLUMED` screen output. It is quite important that at runtime you check it so to verify that all the option are interpreted correctly from `PLUMED`.

Other programs require a different way to specify that `PLUMED` is enabled. Check the manual for reference and the input files provided in the directory test of the standard `PLUMED` distribution.

Just to mention a few of them, enabling `PLUMED` in NAMD and specification of the input file is done through the following syntax

```
plumed            on
plumedfile     plumed.cfg
```

where here there is no default extension (i.e. the name has to be fully specified).
With SANDER (AMBER MD module), you do similarly, specifying it at runtime:

```
&cntrl
 imin=0, irest=0, ntx=1, ig=71278 ,
 nstlim=1001, dt=0.0002,
 ntc=1, ntf=1,
 ntt=3, gamma_ln=5 ,
 tempi=300.0, temp0=300.0,
 ntpr=200, ntwx=0,
```

```
 ntb=0, igb=0,
 cut=999., plumed=1 , plumedfile='plumed.dat'
/
```

2.1 Syntax for collective variables

A typical input file for PLUMED input is composed by specification of one or more CVs, the printout frequency and a termination line.

Example.
A very simple PLUMED input file.

```
# specify a torsion
phi:  TORSION ATOMS=5,7,9,15
# printout frequency
PRINT ARG=phi STRIDE=10 FILE=COLVAR
# the end of plumed input
ENDPLUMED
```

Comments are denoted with a # and the termination of the input for PLUMED is marked with the keyword ENDMETA. Whatever it follows is ignored by PLUMED. You can introduce blank lines. They are not interpreted by PLUMED.

The line that starts with the keyword PRINT control the frequency for the main PLUMED output file which is called COLVAR. This file contains the data regarding the collective variable positions, the constraint positions, the energy of hills and energy of constraints and other useful informations that will be introduced time by time during the tutorial. The frequency for writing is controlled by W_STRIDE followed by a number that represents the number of steps between one printout and the other. All the informations are appended in the COLVAR file and overwritten if an old COLVAR file already exists. In addition in GROMACS if

an old COLVAR file is detected, this is saved in a file COLVAR.old so to prevent overwriting. As this tutorial is not intended to give a survey on the all CVs available in PLUMED but more on the things you can do with that, please do refer to the manual for specific CV related syntax.

Another useful feature is the use of the groups. It may happen that one want to calculate properties between group of atoms. In this case the keyword LIST can be replaced by some groups denoted by angle bracket <g1>. In this case PLUMED looks for a *group* in the plumed input.

Example.
A very simple PLUMED input file with groups.

```
 # specify a group

g1:  COM ATOMS=15,16,17
# specify a torsion
phi:  TORSION ATOMS=5,7,9,g1
# printout frequency
PRINT ARG=phi STRIDE=10 FILE=COLVAR
# the end of plumed input
ENDPLUMED
```

This can be very useful and the group syntax allows for looping on the atoms as well. Please refer to the manual for this option.

2.2 Monitoring a CV

As first example we perform a simple MD run of alanine dipeptide with GROMOS96 all atom force field. In Fig. 2.1 you can see the molecular structure. Its free energy landscape is conveniently depicted as function of the two dihedral angles Φ and Ψ (also called "Ramachandan plot") and therefore in this exercise we just run simply PLUMED and control the two dihedral angles. First copy the exercise and enter the directory:

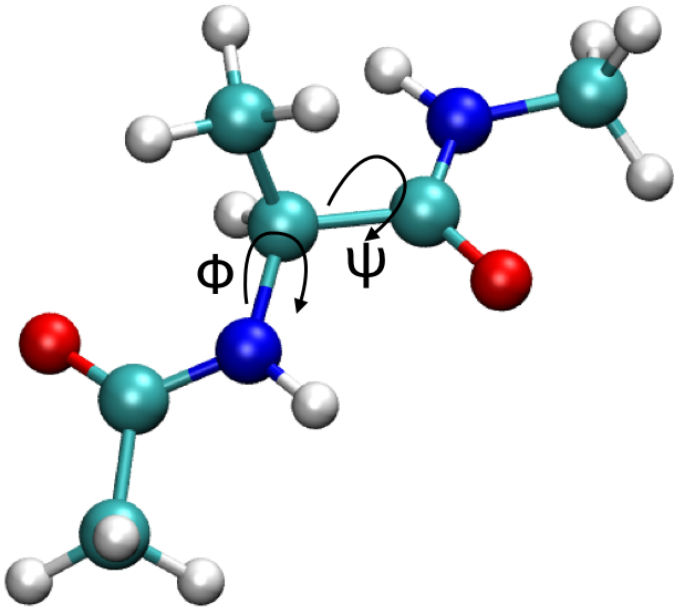


Figure 2.1: A sketch of the molecular structure of alanine dipeptide. The dihedral angles Φ and Ψ are highlited.

```
cp -r /nfs_home/tutoadmin/PLUMED/EXERCISES_FINAL/2_basics .
cd 2_basics/monitoring/
```

In this run we will perform 100 ps of NVT MD and check the evolution of the collective variables with this simple input (open the file `plumed.dat`)

Example.
A very simple PLUMED input file for monitoring the two Φ and Ψ dihedral angles. `# specify phi and psi`

```
phi:  TORSION ATOMS=5,7,9,15
psi:  TORSION ATOMS=7,9,15,17
# printout frequency
PRINT ARG=phi,psi STRIDE=10 FILE=COLVAR
ENDPLUMED
```

Now edit the script `build_and_run.sh`. This is a script that performs all the needed action for doing a run: it sources the correct environments, it creates the binary topology (this step is only required in GROMACS) and performs the calculations. The only thing that you should do is to adapt it to your path, execute it (just by typing `./build_and_run.sh`) and you'll get a bunch of interesting stuff.

First of all you will get a `md.log` file that contains some printout from `PLUMED` so you may check wether the input was correctly read:

```
::::::::::::::::: READING PLUMED INPUT ::::::::::::::::
|-PRINTING ON COLVAR FILE EVERY 100 STEPS
|-INITIAL TIME OFFSET IS 0.000000 TIME UNITS

1-TORSION: (1st SET: 1 ATOMS),
       (2nd SET: 1 ATOMS), (3rd SET: 1 ATOMS) , (4th SET: 1 ATOMS);
|- 1st SET MEMBERS:  5
|- 2nd SET MEMBERS:  7
|- 3rd SET MEMBERS:  9
|- 4th SET MEMBERS:  15


2-TORSION: (1st SET: 1 ATOMS),
       (2nd SET: 1 ATOMS), (3rd SET: 1 ATOMS) , (4th SET: 1 ATOMS);
|- 1st SET MEMBERS:  7
|- 2nd SET MEMBERS:  9
|- 3rd SET MEMBERS:  15
|- 4th SET MEMBERS:  17

|-ANALYSIS: YOU WILL ONLY MONITOR YOUR CVs DYNAMICS
```

This tells you that everything is going fine. The index of atoms are parsed correctly and the printout is correctly understood. Now what you get is a `COLVAR` file that consists in the time evolution of the CVs. Its format looks something like this:

```
#! FIELDS time cv1 cv2 vbias vwall vext
     0.000     -2.655213716      2.760231131      0.000000000      .
     0.020     -2.676506406      2.845943731      0.000000000      .
     0.040     -2.646984155      2.749365464      0.000000000      .
     ...
```

In the first line there is a simple remainder to the elements that you have in each column. Namely time first (in ps by default in GROMACS), then the value of the two CVs followed by the various additional potential energies introduced by plumed. In this case there is no additional potential introduced and therefore all those columns are zeros. Now you can plot the evolution of the CVs with gnuplot by using the command `p "./COLVAR" u 1:2 t "Phi" ,"" u 1:3 t "Psi"` and you'll get something like Fig. 2.2 If you want to understand how they are related on the Ramachandran plot then you might use the command `p "./COLVAR" u 2:3` with gnuplot that results in a plot like that in Fig. 2.3.

2.3 Postprocessing with `driver`

```
#!/bin/bash
source  /usr/local/Modules/3.2.6/init/bash
module load intel-cc/10.1.015
module load intel-fc/10.1.015
```

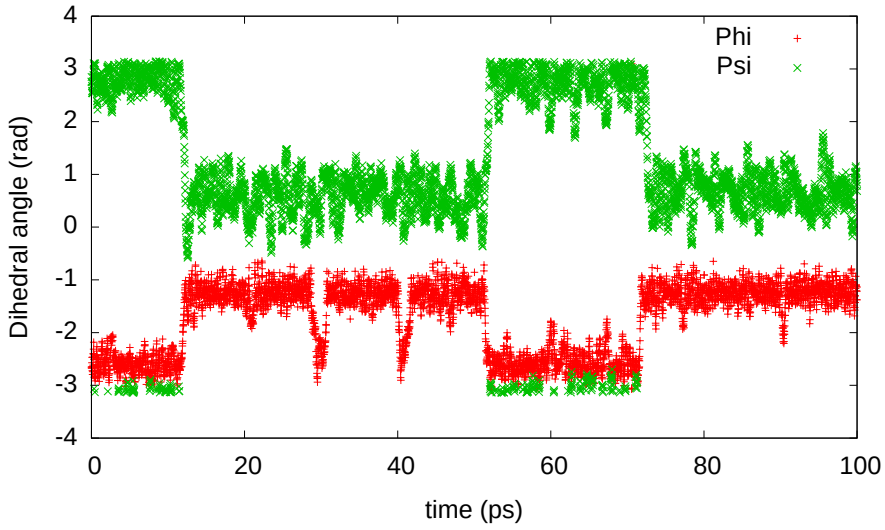


Figure 2.2: The time evolution of the variables Φ and Ψ

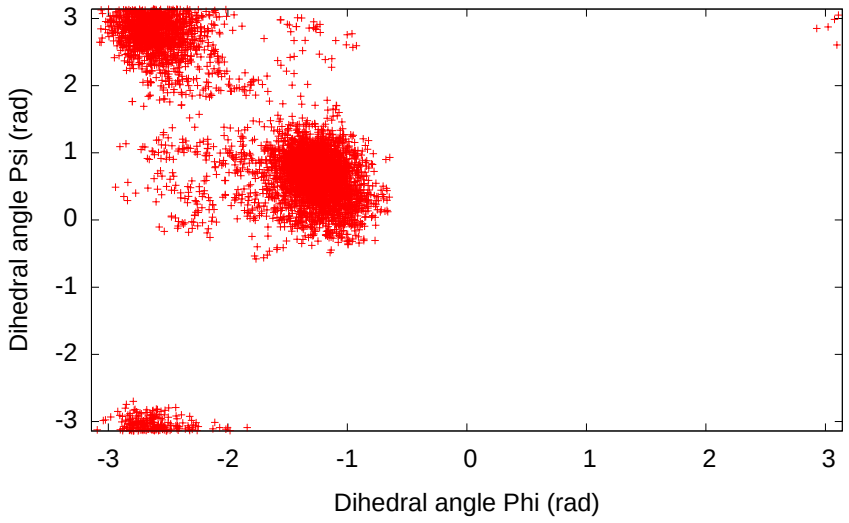


Figure 2.3: The time population of the variables Φ and Ψ during 100 ps run.

```
module load fftw/3.2.2_intel-10.1.015
module load openmpi/1.2.6_intel-10.1.015
source /nfs_home/tutoadmin/PLUMED/EXERCISES
            /installation/gromacs/gromacs-4.0.7/install_dir/bin/GMXR
CATDCD="/nfs_home/tutoadmin/PLUMED/catdcd/LINUXAMD64/bin/catdcd4.0/c
TRJCONV="trjconv_d"
DRIVER="/scratch/tutoxx/1_compiling/plumed/PLUMED-1.2.1/utilities/dr
mv COLVAR COLVAR.bak
#
# generate the pdb
#
echo 0 | $TRJCONV -f 2ala.gro -o 2ala.pdb
#
# generate the dcd
#
$CATDCD  -o traj.dcd -trr traj.trr
#
# run the driver
#
$DRIVER -pdb 2ala.pdb -dcd traj.dcd -ncv 2 -nopbc -plumed plumed2.da
```

"Oh my god! I forgot to include that special CV!". No problem! Max Bonomi wrote the `driver` utility (you may find it in `$plumedir/utilities/driver`) through which you may post-process your run and calculate that special CV you forgot to include runtime. Of course, being limited to the points for which you actually have the trajectory you don't have much statistics unless you collect very often the trajectory points (absolutely not recommended unless you want your supervisor hating you). First compile it. Go into your `PLUMED` directory (should be someting like `/scratch/tutoxx/1_compiling/plumed`).

```
cd PLUMED-1.2.1/
cd utilities/
cd driver/
ln -s ../../common_files/* .
module load intel-cc/10.1.015
module load  intel-fc/10.1.015
make intel
```

Then you should convert the `2ala.gro` and `traj.trr` into a pdb/dcd couple. This can be done with trajconv (from GROMACS) and catdcd (from Klaus Schulten group http://www.ks.uiuc.edu/Developm) At this stage you have what you need (a `*.pdb` and `*.dcd` couple). This is done automatically by sourcing the correct executable from the script `postprocess_with_driver.sh` (have a look into that, if you want to do this exercise automatically you should put the correct path). Let us assume that what you forget to calculate is now the distance for NME3:H (atom 18) - ACE1:O (atom 6) and ALA2:O (atom 16) - ALA2:H (atom 8) . Then a you have to produce a new plumed input (say `plumed2.dat` as in the example)

Example.

```
#
#
# the two distances for hydrogen bonds
#
d1:  DISTANCE ATOMS=18,6
d2:  DISTANCE ATOMS=16,8
# one printout each frame
#
PRINT ARG=d1,d2 STRIDE=1 FILE=COLVAR
ENDPLUMED
```

and by using the following command (assuming that `driver` points to the correct utility)

```
plumed driver --pdb 2ala.pdb --igro traj.gro --plumed plumed2.dat
```

you get a new `COLVAR` file with the data you need and by now you should be able to check the data you are obtaining with gnuplot. All the previos commands are included `postprocess_with_driver.sh`. By simply adapting it to your path and executing it you will have this exercise done.

Chapter 3

Basics: biasing simulations

3.1 Restrained/steered molecular dynamics

Once you define a bunch of collective variables of your interests PLUMED has a number of ways in which you may affect their behavior. The fact that you can actually influence their value depends on the fact that each of the collective variables implemented in PLUMED has analytical derivatives and, by biasing the value of a single CV one turns to affect the time evolution of the system itself. The simplest way in which one might influence a CV is by forcing the system to stay close to a chosen value during the simulation. This is achieved with umbrella potential that plumed provides via the directive UMBRELLA

Very often it may happen that one wants that a given CV just stay within a given range of values. This is achieved in plumed through the directives UWALL and LWALL that act on specific collective variables and limit the exploration within given ranges.

Another useful strategy is the STEERPLAN directive. It allows to perform a series of programmed steered runs and can be helpful in performing adaptive umbrella sampling or multievent reaction within a single simulation.

3.1.1 An umbrella sampling calculation. Alanine dipeptide.

As first example we perform an umbrella sampling calculation of the free energy landscape of alanine dipeptide with GROMOS96 all atom force field already seen in the previous section. In Fig. 2.1 you can see the molecular structure. Its free energy landscape is conveniently depicted as function of the two dihedral angles Φ and Ψ (also called "Ramachandan plot"). An useful approach to depict the free energy landscape of this molecule is based on the so called "Umbrella Sampling" (US) algorithm. Without going much into details the calculation follows like this.

- Put an umbrella potential on a specified value of Φ and Ψ (say Φ_0 and Ψ_0).
- Run a simulation and acquire reasonable statistics of the deviation of Φ and Ψ respect to the position Φ_0 and Ψ_0.
- Change a bit the position of the restrain center Φ_0 and Ψ_0 and start over a new simulation from the previous endpoint.
- Iterate.

By moving the restraint in with a "snake-like" evolution one can cover the whole range of values within the whole domain of interest $-\pi < \Phi < \pi$ and $-\pi < \Psi < \pi$ and, if the "clouds" of probability of restraints are overlapping, one can retrieve the Potential of Mean Force (PMF) by using the Weighted Histogram Analysis Method (WHAM). The program for WHAM is well established and can be downloaded from Alan Grossfield website (http://membrane.urmc.rochester.edu/content/wham).

The input for a single umbrella for `PLUMED` is something like this:

Example.
A typical setup for a single umbrella in the umbrella sampling run

```
#
#
# set up two variables for Phi and Psi dihedral angles
#
phi:  TORSION ATOMS=5,7,9,15
psi:  TORSION ATOMS=7,9,15,17
#
# Impose an umbrella potential on CV 1 and CV 2
# with a spring constant of 500 kjoule/mol
# at fixed points on the Ramachandran plot
#
umb:  RESTRAINT ARG=phi,psi KAPPA=500,500 AT=3.141593,-3.141593
# only some printout for acquiring the statistics
#
PRINT ARG=phi,psi,umb.bias STRIDE=10 FILE=COLVAR
ENDMETA
```

The syntax for the directive `UMBRELLA` is rather trivial. The directive is followed by a keyword `CV` followed by an index that specify the number of the CV on which the umbrella potential has to act. The keyword `KAPPA` determines the hardness of the spring constant and its units are

$$[Energy\ units\ of\ the\ program]/[Units\ of\ the\ CV]^2 \quad (3.1)$$

The additional potential introduced by the `UMBRELLA` takes the form of a simple Hooke's law:

$$U(x) = \frac{\texttt{KAPPA}}{2}(CV(x) - CV_0)^2 \quad (3.2)$$

where CV_0 is the value specified following the `AT` keyword. In output the sum of the additional potential is shown in the `COLVAR` file. For this specific input the `COLVAR` file has in first column the time, in the second the value of CV 1, in the third the value of CV 2, in the fourth the potential from metadynamics (zero if metadynamics is not active), and in the fifth the additional potential introduced by umbrellas and walls. The position of

the restrain is also reported with the keyword `RST` followed by the index of the CV on which the umbrella potential is applied and the position of the constraint.

The procedure explained before is summarized by the script `script.sh` that you find in the exercise folder. After you run that script you should end up with a file called `metadatafile` and a number of files having names like `CV_*`. The file `CV_*` contain the time and the value of the CVs at each time. These are produced by the script itself by simple postprocessing of the `COLVAR` file. The `metadatafile` contains the name of each time evolution `CV_*` followed by the center of the springs and the spring constants. The `wham-2d` program just need the `metadatafile` and `CV_*`. The simple command

```
wham-2d Px=pi -3.14159 3.14159 50 Py=pi -3.14159 3.14159
            50 0.001 300 0 metadatafile free_ene.dat >out_wham
```

should be sufficient in producing the free energy landscape in 2d that is represented in 3.1. In the end you should get something like that in Fig. 3.1. You might obtain such a plot by using this simple gnuplot script:

```
set term pdf noenhanced
set out "rama_gromos96.pdf"
unset key
unset colorbox
set size square
set multiplot
set view map
set xr [-pi:pi]
set yr [-pi:pi]
spl "free_ene.dat" u  1:2:3 w pm3d
```

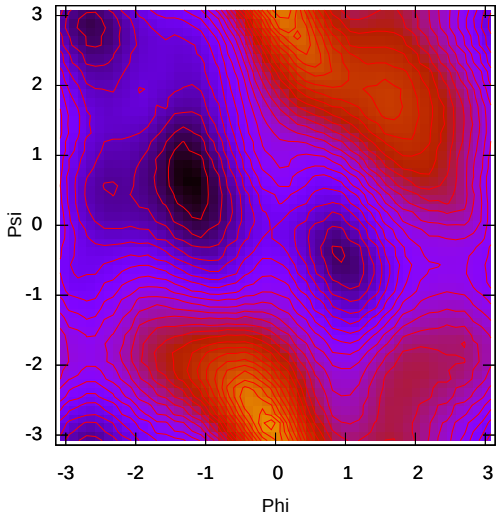


Figure 3.1: The free energy landscape of an alanine dipeptide with the GROMOS96 allatom force field. Isolines are drawn every 1kcal/mol.

```
set xlab "\Phi"
set ylab "\Psi"
unset surf
set contour base
set cntrp lev incr 0,1,40
unset pm3d
unset clabel
spl "free_ene.dat" u 1:2:3 w l
quit
```

3.1.2 A steered molecular dynamics example: targeted MD.

As a second example of the restraint features of PLUMED we discuss here the use of the directive STEER. Very often it is useful to drag the system from an initial configuration to a final one by pulling one or more CVs. Most of time the aim of such simulations is to prepare the system in a particular state or produce nice snapshots for a cool movie. The reason behin the limited usefulness of such simulations is that in most of the cases these are "out-of-equilibrium" simulations and they are not representative of equilibrium ensemble that is generally of interest for simulations. However, Jarzynski [**?**] inequality provides a connection between out-of-equilibrium trajectories and equilibrium free energy differences. This inequality consists in:

$$\Delta F = -\beta^{-1} \ln < \exp(-\beta W) > \tag{3.3}$$

where the average is calculated over the work obtained from a number of out-of.equilibrium trajectories. β is $k_b T$ and ΔF is the free energy difference. The value of W can be obtained via

$$W = \int_0^{t_s} dt \, \frac{\partial H_\lambda(t)}{\partial t} \tag{3.4}$$

where H_λ is a modified hamiltonian which contains an additional term, namely:

$$\begin{aligned} H_\lambda(t) &= H + U_\lambda(t) && (3.5)\\ &= H + \frac{k}{2}(CV(x) - \lambda(t))^2 && (3.6)\\ &= H + \frac{k}{2}(CV(x) - CV_0 - vt)^2. && (3.7) \end{aligned}$$

From which it comes naturally that the derivative is simply:

$$\begin{aligned} \frac{\partial H_\lambda(t)}{\partial t} &= -vk(CV(x) - CV_0 - vt) && (3.8)\\ &= -vk(CV(x) - \lambda(t)) && (3.9) \end{aligned}$$

and therefore the integral W can be obtained simply by quadrature summing up all the deviation respect to the position of the mobile center of the harmonic restraint.

In the limit of lots of pulling therefore meaningful free energy differences can be retrived. Here in this example we do not focus on the Jarzynski limit but on the details on how to steer a molecule fo alanine dipeptide towards one of its minima. We first prepare the system around $\Phi \simeq -2.6, \Psi \simeq 2.8$ (see Fig. 3.1) and we pull the heavy atoms towards the other minimum located around $\Phi \simeq 1, \Psi \simeq -0.5$. In order to do that we won't use the two dihedrals but an input pdb representing the heavy atoms in that situation. This is generally called "Targeted MD". The input looks like:

Example.

```
#
# create a cv that measure MSD from a reference structure
#
rmsd:  RMSD REFERENCE=min.pdb TYPE=OPTIMAL SQUARED
#
# steer this CV to a value of 0.0
#
stmd:  MOVINGRESTRAINT ARG=rmsd STEP0=0 AT0=0.02757 KAPPA0=10000000.0 STEP1=270000
AT1=0.0 KAPPA1=10000000.0
#
# additional variables that may be useful
#
phi:  TORSION ATOMS=5,7,9,15
psi:  TORSION ATOMS=7,9,15,17

PRINT ARG=rmsd,phi,psi,stmd.bias STRIDE=100 FILE=COLVAR
ENDPLUMED
```

Note the syntax of the `TARGETED` that is intended to use a reference in MSD space (min.pdb in this case). The syntax of `STEER` is also specified. It requires a keyword `CV` in order to specify the index of the variable on which the steering has to be carried out. In this system there are three variables (TORSION are also CVs) but only the first CV (TARGETED) has to be steered to a given value. The target value is specified by the keyword `TO` followed by the value that has to be reached. Then the keyword `KAPPA` comes and specify the hardness of the spring constant (in this case the units are (kjoule/mol)/nm^4 as the units of the `TARGETED` variable are nm^2. This is true only for GROMACS. Other programs adopt Å^2). One can follow the evolution through gnuplot by visualizing the instantaneous value of the CV and the position of the moving constraint by the simple command `p "./COLVAR" u 1:2 t "CV","" u 1:10 t "Moving constraint"` and obtain a plot like the one reported in Fig. 3.2. Now we can also give a try in calculating the work W. Run the awk script with the command:

```
./integrate.awk time=1 cv=2 constr=10 kappa=10000000.0
```

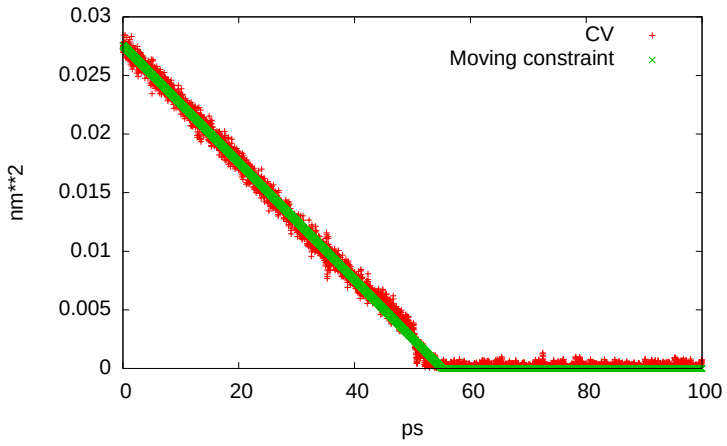


Figure 3.2: Following a steered MD run during time

`COLVAR >work.dat`
and plot work.dat with gnuplot and the command `p "./work.dat" u 2:($3/4.186) w lp` so you will obtain a plot of the work (see Fig. 3.3).

3.1.3 A programmed steered MD with steerplan.

Another feature that is implemented in `PLUMED` is a "tunable" moving restraint called `STEERPLAN`. The basic idea behind is the fact that many times what one wants to study is a multistep process. For example consider an enzymatic reaction which is made of several proton transfers and nucleophilic attack. It can be rather usefil whenever one needs to create a guess path for path collective variables calculations. Instead of creating multiple steered MD run, each of those has a single cv which is steered at a time, you can put all the variables in a single `PLUMED` input and then plan the steering so that you first steer one CV then you switch this potential off and switch another

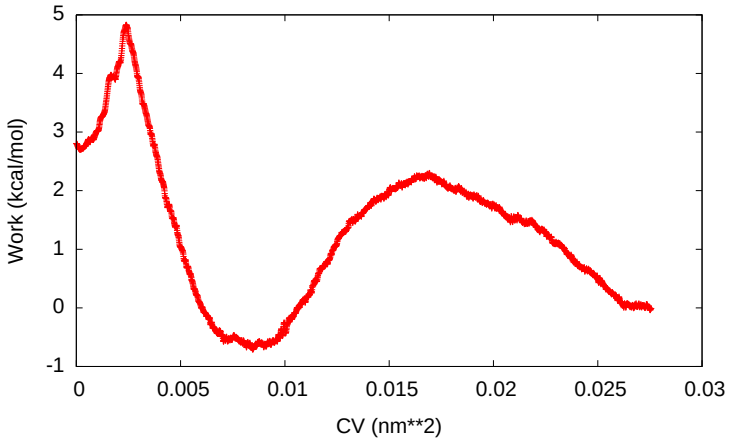


Figure 3.3: The work calculated during a steered MD run.

potential that steers a different CV, all within a single run. In this example we will see the usual alanine dipeptide and we will reach the minimum at $\Phi \simeq 1, \Psi \simeq -0.5$ through a two-step process. First we will pull Ψ to -0.5 and keep Φ fixed at -2.6, then we will pull Φ to 1.0 and then release all the constraint. The input is simple

Example.

```
#
# additional variables that may be useful
#
phi:  TORSION ATOMS=5,7,9,15
psi:  TORSION ATOMS=7,9,15,17
#
# steerplan
#
MOVINGRESTRAINT ...
ARG=phi,psi
STEP0=0 KAPPA0=100,100 AT0=-2.6,2.8
STEP1=150000 KAPPA1=100,100 AT1=-2.6,-0.5
STEP2=300000 KAPPA2=100,100 AT2=1.0,-0.5
STEP3=350000 KAPPA3=0,0 AT3=1.0,-0.5
LABEL=stpl
...  MOVINGRESTRAINT

PRINT ARG=phi,psi,stpl.bias STRIDE=100 FILE=COLVAR
ENDPLUMED
```

which means: at time 0.0 put an harmonic constraint on the CV 1 whose hardness is 100 kjoule/mol centered on the value -2.6. On the CV 2 do the same but centered on the value 2.8. At later time (30 ps) the constraint on CV 1 is the same while on CV 2 its center should be now at 0.5. Thus the program is keeping a fixed harmonic constraint on the first variable while moving the second up to reaching -0.5. From 30 ps to 60 ps is the CV 1 that should move from -2.6 to 1.0. Finally, from 60 to 70 ps the springs are switched off (the spring constant are put to 0.0) and the system is now free to thermalize in the new minimum. Consider that `STEERPLAN` allows the use of wildcards `*` on the position of the constraint whose meaning is "start with an umbrella centered where the system is" (please see the manual for the full documentation). `COLVAR` file reports in the final columns the steerplan action in progress denoted by `STP` which is followed by something like :

```
CV  1 X    -2.600000 K   100.000000 T 1
```

that tells on which CV the steerplan is active, where is the center of the potential, the spring constant and the type of the potential used (there exist three different types `CENTRAL` whose code is 1, `POSITIVE` whose code is 2, `NEGATIVE` whose code is 3. See manual for further details). After you run the example it is straightforward to understand what happened by plotting the `COLVAR` file with gnuplot by using the following command: `p "./COLVAR" u 2:3 t "CVs","" u 13:21 t "designed path"` (see Fig. 3.4) .

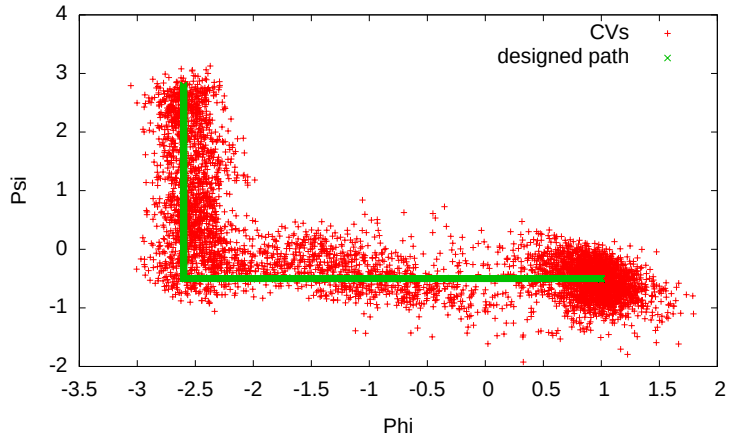


Figure 3.4: Evolution of the CV when steerplan is used.

3.1.4 Soft walls

PLUMED implements soft and hard walls. Here the soft walls are explained. Very often during a metadynamics simulation it might happen that one wants to avoid to sample some regions because they are not chemically relevant for the pricess under study. In this case one can adopt the so-called "walls". In PLUMED there exist two kind of walls: LWALL (lower wall) and UWALL (upper wall). They act on two different directions. LWALL prevents the system to go lower than a specified value while UWALL prevents the system to explore regions whose CV is higher than a specified value. This is achieved by using (by default) a fourth order harmonic function. Some parameters can also be tuned. The walls take the form for values larger or smaller than LIMIT in case of UWALL and LWALL respectively:

$$V_{wall}(s) = \texttt{KAPPA}\left(\frac{s - \texttt{LIMIT} + \texttt{OFF}}{\texttt{EPS}}\right)^{\texttt{EXP}}, \tag{3.10}$$

where KAPPA is an energy constant in internal unit of the code, EPS a rescaling factor and EXP the exponent determining the power law. By default: EXP = 4, EPS = 1.0, OFF = 0 and are optional arguments In the exercise we run alanine dipeptide in a region close to the saddle point. In order to do that we confine with UWALL and LWALL the variable Φ. The PLUMED input looks something like this

Example.

```
#
# Phi and psi angles
#
phi:  TORSION ATOMS=5,7,9,15
psi:  TORSION ATOMS=7,9,15,17
#
# LOWER_WALL: a lower wall
# UPPER_UWALL : an upper wall
#
uwl:  UPPER_WALLS ARG=phi AT=0.1 KAPPA=100.0
lwl:  LOWER_WALLS ARG=psi AT=-0.1 KAPPA=100.0
PRINT ARG=phi,psi,uwl.bias,lwl.bias STRIDE=100 FILE=COLVAR
ENDPLUMED
```

The syntax for LWALL and UWALL closely resembles the ones from the UMBRELLA. You can at this point easily figure out its meaning. LIMIT needs to be followed by one number that specifies the bundary. KAPPA is the spring constant. Have a look to the CVs during the run by using the command in gnuplot: `p "./COLVAR" u 2:3`. You can notice that the variable very often passes the imposed boudaries. This is not surprising because the spring potential simply discourage to sample those regions but is does not a-priori forbids their exploration. By tuning KAPPA one can limit the exploration to the imposed region. Try harder KAPPAs. What does it happen?

3.2 Committment analysis

At this stage it might be interesting to figure out you may use `PLUMED` and GROMACS together in a single bash script so to perform useful tasks. One of this task is the calculation of the so-called "committor" function. A central issue of the rare events research is how to judge wether a given transition state (i.e. a saddle point) in the free energy has the expected dynamical meaning. Without dwelling into details (see the good literature for that [**?**, **?**, **?**, **?**, **?**, **?**]) the committor probability is calculated as the probability of falling into product state before falling into the reactant state when starting from a given point in the phase space (that can be a given region in the collective variable). Therefore, we added this section here as in the previous exercise we already collected a bunch of point along the transition state. This is evident if you plot the Ramachandran angles in the usual way. You might find that part of the points fall at $\Phi < 0$ and part fall at $\Phi > 0$ that suggest that, across the point $\Phi \simeq 0$ you have a transition state of some kind. Further proof of it can be obtained from Fig. 3.1. In order to do this we first collect some points along the TS with the run having walls (see Sect. 3.1.4) because previously no frames were collected in the trajectory (you can tune this in GROMACS input with the flag `nstxout`). Then we have to follow this workflow:

- Convert `traj.trr` into a dcd (using `catdcd`).
- Convert `2ala_ts.gro` into a pdb (using `trjconv`).
- Postprocess the `.pdb` and `.dcd` with the `driver` so to retrieve the exact position of Φ per each frame
- Postprocess the `COLVAR` file produced by the `driver` through

`awk` and create a series of files containing the indexes of the frames along the trajectory for each slice along the collective variable.

- For each frame contained in each slice, run a MD with the `PLUMED` file containing the `STOPWHEN` keyword so to kill the run as soon as it reaches one boundary. In this case we consider to kill the run when $\Phi < -0.8$ or $\Phi > 0.8$ and we increase a counter only if the reactant is reached.

- At the end of running all the frames contained in the slice, then the ratio between the frames that reached the reactant and the total number of frames give the resulting value of the committor.

Have a look to the script `committor.sh`: it contains all the steps to perform this simple committor analysis. If you launch it with `./committor.sh >outcomm` at the end you should obtain a plot like the one in Fig. 3.5.

3.3 Metadynamics

Metadynamics adds an external potential to the simulation. This bias potential acts on a restricted number of degrees of freedom of the system $\boldsymbol{S}(\boldsymbol{R}) = (S_1(\boldsymbol{R}), ..., S_d(\boldsymbol{R}))$ often referred to as collective variables or CVs. The metadynamics potential $V(\boldsymbol{S}, t)$ varies with time t and is constructed as a sum of Gaussian functions, or hills, deposited during the simulation:

$$V(\boldsymbol{S}, t) = \int_0^t dt' \omega \exp\left(- \sum_{i=1}^{d} \frac{(S_i(\boldsymbol{R}) - S_i(\boldsymbol{R}(t'))^2}{2\sigma_i^2} \right), \quad (3.11)$$

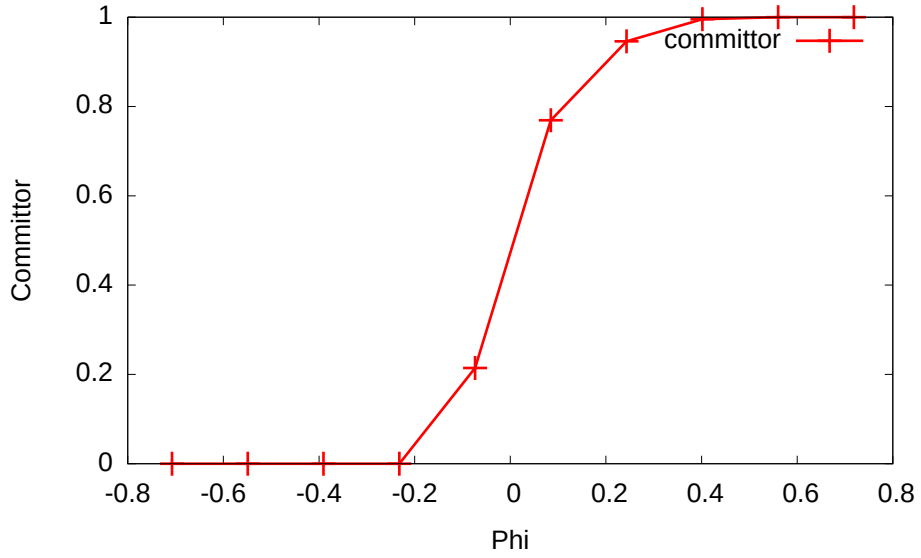


Figure 3.5: The committor as calculated in the exercise.

where σ_i is the Gaussian width corresponding to the i-th CV and ω the rate at which the bias grows. In the practice, Gaussians of height equal to W are deposited every τ MD steps, so that $\omega = W/\tau$.

In order to perform a metadynamics simulation we have to:

- Choose wisely the set of CVs to address the problem. This is a long story. To cut it short, CVs a) should clearly distinguish between the initial state, the final state and the intermediates, b) should describe all the slow events that are relevant to the process of interest, c) their number should not be too large, otherwise it will take a very long time to fill the free energy surface.

- Choose the Gaussian height W and the deposition stride τ. These two variables determine the rate of energy added to your simulation. If this is too large, the free energy surface

will be explored at a fast pace, but the reconstructed profile will be affected by large errors. If the rate is small, the reconstruction will be accurate, but it will take a longer time. The error on the reconstructed FES depends on the ratio W/τ, not on the two parameters alone [**?**].

- Choose the Gaussian width σ_i. This parameter determines the resolution of the reconstructed FES. The sum of Gaussians reproduces efficiently (*i.e.* in a finite simulation time) features of the FES on a scale larger than σ_i. A practical rule is to choose the width as a fraction (half or one third) of the CV fluctuations in an unbiased simulation. This is not a golden rule, since the value of the fluctuations is not universal but usually depends on the position in the CV space.

To activate a metadynamics calculation in `PLUMED` you have to use the directive `HILLS`. The deposition stride τ is specified in unit of time steps by the keyword `W_STRIDE`, the height W by `HEIGHT` in internal units of energy of the MD code used. The Gaussian width σ_i must be specified on the line of each CVs with the keyword `SIGMA`.

A typical `PLUMED` input file for a metadynamics calculation looks as follows.

Example.
```
phi:  TORSION ATOMS=5,7,9,15
METAD ARG=phi PACE=1000 HEIGHT=0.4 SIGMA=0.35
ENDPLUMED
```

Beside the usual `COLVAR` file, when you run a metadynamics calculation you get an additional file called `HILLS` which contains

a list of the Gaussians deposited during the simulation. In the example above, this file would look like:

```
1.000     -2.617548716     0.350000000     0.400000000   0.00
2.000     -2.718742869     0.350000000     0.400000000   0.00
3.000     -2.662313657     0.350000000     0.400000000   0.00
4.000     -2.378730722     0.350000000     0.400000000   0.00
5.000     -2.120031391     0.350000000     0.400000000   0.00
```

where:

- the first column contains the time t (in internal unit of the MD code) at which the Gaussian was deposited;
- the following d columns contain the centroid of the Gaussian, $S_i(\boldsymbol{R}(t))$, one for each CV i;
- the following d columns contain the Gaussian sigma σ_i, one for each CV i;
- the last but one column contains the value of W;
- the last column is meaningful only in well-tempered metadynamics simulations (see below).

This file will be used to calculate the estimate of the free-energy at the end of our metadynamics calculation.

Beside the metadynamics CVs, we can add other variables that we want to monitor during the simulation. A typical `PLUMED` input file looks as follows.

Example.
Here we want to run a metadynamics calculation using a dihedral as CV. During the run we want to monitor the evolution of another dihedral angle.

```
phi:  TORSION ATOMS=5,7,9,15
psi:  TORSION ATOMS=7,9,15,17
mtd:  METAD ARG=phi,psi PACE=1000 HEIGHT=0.4 SIGMA=0.35,0.35
PRINT ARG=phi,psi,mtd.bias STRIDE=500 FILE=COLVAR
ENDPLUMED
```

The evolution of the additional variable can be monitored by looking at the `COLVAR` file.

```
#! FIELDS time cv1 cv2 vbias
     0.000     -2.655726464      2.760989206      0.000000000
     0.500     -2.492360677      1.370313209      0.000000000
     1.000     -2.617548716     -0.807834974      0.400000000
     1.500     -2.163294675      0.453630761      0.172299337
     2.000     -2.718742869     -0.557066598      0.783627027
     2.500     -2.743156519      0.246435084      0.774082974
     3.000     -2.662313657      0.024026518      1.191579452
```

3.4 Restarting metadynamics

In order to restart a metadynamics run, the flag `RESTART` must be added on the line of the directive `HILLS`. This allows a metadynamics simulation to be restarted after an interruption or after a run has finished. The `HILLS` files will be read at the beginning of the simulation and the bias potential applied to the dynamics. Note that the presence of the `RESTART` flag only affects the metadynamics part of the simulation, and thus the usual procedure for restarting a MD run must be followed. This depends on the particular MD engine used and can be found in the relative documentation.

Example.
The following is an example of input file for restarting a metadynamics simulation.

```
RESTART
phi:  TORSION ATOMS=5,7,9,15
METAD ARG=phi PACE=1000 HEIGHT=0.4 SIGMA=0.35
ENDPLUMED
```

3.5 Free-energy reconstruction

In the long-time limit, the bias potential of metadynamics converges to the free-energy changed in sign [**?**]. At any time during the simulation we can sum the Gaussians deposited so far and obtain the current estimate of the FES using the utility `sum_hills`. This code is very flexible and can be executed with several options (see the manual). The most commonly used are:

Example.

```
plumed sum_hills --hills HILLS --outfile fes.dat --idw cv1 cv2 --kt 0.6 --bin 100 100
100

 --hills     input file with the list of Gaussians
 --outfile   output file with the FES
 --idw       ID of the variables for FES in output
 --bin       grid mesh dimension
 --kt        kT in the energy units
 --stride    how often the FES is written
```

The file in output `fes.dat` contains the estimate of the free-energy calculated on a regular grid whose dimension is specified by `--bin`. These parameters should be chosen with care. To calculate accurately the potential in a given point of the CV space, a practical rule is to choose the bin size to be half the Gaussian sigma.

The `sum_hills` code should be used to monitor the convergence of a metadynamics simulation. This can be easily achieved

by calculating the estimate of the FES at regular interval in time using the `--stride` option and then evaluating the free-energy difference among relevant regions (minima) of the FES as a function of time. Below we report an example of bash script that can be used for this purpose.

Example.
Example of script to evaluate the convergence of a metadynamics run. Here we performed a metadynamics calculation using 2 CVs. We have defined two regions in the projection of the FES onto the first variable. The F region in which CV1 is greater than 2, and the U region in which CV1 is lower than 2. The free-energy difference between F and U is calculated as a function of time and saved in the file `DeltaF`.

```
#!/bin/bash
plumed sum_hills --stride 100 --idw cv1 --bin 100 100 --kt 0.6 --hills HILLS
for file in fes.dat.?  fes.dat.??  fes.dat.???
do
if [ -f $file ]; then
F=‘awk ’BEGIN{tot=0}{if(NF==2 && $1>2.0)tot=tot+exp(-$2/0.6)}END{print -0.6*log(tot)}’
$file‘
U=‘awk ’BEGIN{tot=0}{if(NF==2 && $1<=2.0)tot=tot+exp(-$2/0.6)}END{print -0.6*log(tot)}’
$file‘
delta=‘echo "$F - $U" | bc -l‘
echo $delta >> DeltaF
fi
```

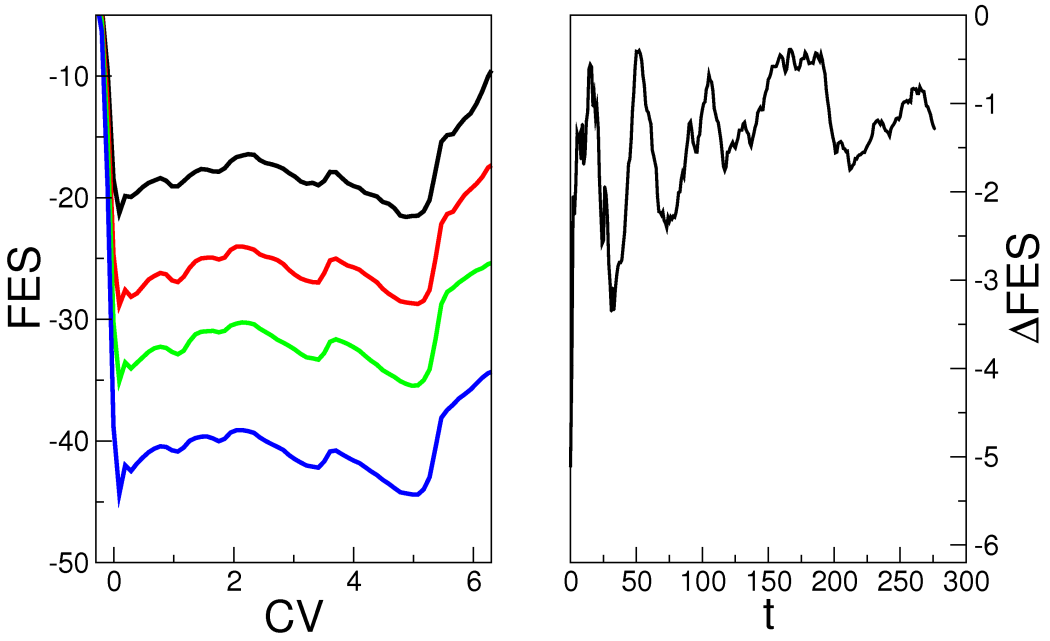


Figure 3.6: Checking the convergence of a metadynamics run. Left panel. Free-energy estimate at different time steps. Right panel. Free-energy difference between state F and U as a function of time.

3.6 Well-tempered metadynamics

In well-tempered (WT) metadynamics, the Gaussian height W is automatically rescaled during the simulations following:

$$W = W_0 \, e^{-\frac{V(\boldsymbol{S},t)}{k_B \Delta T}}, \tag{3.12}$$

where W_0 is the initial Gaussian height and ΔT a parameter with the dimension of a temperature. The use of Eq. 3.12 guarantees that the bias potential converges in a single simulation and does not oscillate around the FES value, causing the problem of overfilling:

$$V(\boldsymbol{S}, t \to \infty) = -\frac{\Delta T}{T + \Delta T} F(\boldsymbol{S}) + C, \tag{3.13}$$

where T is the temperature of the system and C a costant.

The quantity $T + \Delta T$ is often referred to as the (fictitous) CV temperature, while the ratio $(T + \Delta T)/T$ as bias factor. To perform a WT metadynamics simulation with `PLUMED` you have to specify in the directive of `METAD` the parameters described above using the keyword `BIASFACTOR`. In addition, the temperature of the system must be specified explicitly with `TEMP`.

Here are some practical rules to choose wisely the parameters in WT metadynamics simulations:

- The bias factor (or equivalently the CV temperature) regulates how fast the amount of bias potential added decreases with simulation time and eventually controls the extent of exploration. The choice of these parameters depends on the typical free-energy barriers involved in the process under study. In biomolecular simulation, a bias factor of 10-15 which corresponds to barriers of the order of 6-9 kcal/mol

at 300K is usually appropriate. Note that this parameter can be changed on-the-fly as needed.

- The optimal choice of the initial Gaussian height W_0 is less crucial and at the same time less trivial. It is irrilevant in the long time regime and affects only the transient part of the simulation. A short initial filling period can be desirable if the transverse degrees of freedom relax quickly, otherwise a moderate initial energy rate is a better choice.

Example.
The following is an example of input file for a WT metadynamics simulation at 300 (internal units of temperature) with a bias factor equal to 15 and an initial Gaussian height of 0.4 (internal units of energy).

```
phi:  TORSION ATOMS=5,7,9,15
METAD ARG=phi PACE=1000 HEIGHT=0.4 SIGMA=0.35 TEMP=300 BIASFACTOR=15
ENDPLUMED
```

In WT metadynamics, the Gaussians height as written in the `HILLS` file is multiplied by the factor $(T + \Delta T)/\Delta T$. This guarantees that when you sum the Gaussians (by means for example of the `sum_hills` code) you get directly the FES. The last column of the `HILLS` file contains the value of the bias factor used in the WT metadynamics simulation. For the example above, this file would look like:

```
1.000     -2.617548716     0.350000000     0.428571429   15.0
2.000     -2.718742869     0.350000000     0.423889087   15.0
3.000     -2.662247736     0.350000000     0.419017741   15.0
4.000     -2.380845469     0.350000000     0.418270876   15.0
5.000     -2.119639700     0.350000000     0.420668977   15.0
```

Chapter 4

Parallel machines

Parallel machines can be exploited at two different levels:

- Parallel molecular dynamics - i.e. splitting particles on the processors so as to run your simulation faster.
- Parallel sampling algorithms - i.e. running multiple independent or dependent simulations at the same time.

The first type of parallelism will give a more or less linear scaling (e.g. using two processors the simulation will be twice as fast), depending on the size of the system (better scaling for larger systems) and the efficiency of the MD code. The second type of parallelism uses intrinsically different algorithms, discussed below, and the choice of the number of processors is more subtle. The two levels can be combined. For instance, if you have a 128-processor machine, you can perform 16 simulations with 8 processors each.

4.1 Exploiting MD parallelization

Most of the MD codes compatible with PLUMED can be run with some form of parallelism, such as particle decomposition or

domain decomposition. Often this is done simply by choosing the proper number of processors with the command mpirun. With GROMACS:

Example.
```
mpirun -np 8 mdrun -plumed plumed.dat
```

Notice that perfect scaling (i.e. speed proportional to number of processes) is difficult to achieve. Moreoever, PLUMED is going to slow done parallel simulations in some cases. In particular, the computation of the collective variables is done serially and could become the bottleneck. As an example, try to run beta-hairpin in water using an increasing number of processors and measure the bottleneck arising from plumed.

Example.
```
time mpirun -np 1 mdrun
time mpirun -np 2 mdrun
time mpirun -np 4 mdrun
time mpirun -np 1 mdrun -plumed plumed.dat
time mpirun -np 2 mdrun -plumed plumed.dat
time mpirun -np 4 mdrun -plumed plumed.dat
```

Try to do it with a light collective variable:

Example.
```
g1: COM ATOMS=1-500
g2: COM ATOMS=501-1000
d1: DISTANCE ATOMS=g1,g2
PRINT ARG=d1 STRIDE=1 FILE=COLVAR

ENDPLUMED
```

and with a heavy variable:

Example.

```
g1: GROUP ATOMS=1-500
g2: GROUP ATOMS=501-1000
coord: COORDINATION GROUPA=g1 GROUPB=g2 NN=6 MM=12 R_0=0.75 D_0=3.

PRINT ARG=coord STRIDE=1 FILE=COLVAR

ENDMETA
```

4.2 Multiple-walkers metadynamics

When performing metadynamics, one needs to "fill" wells in the free-energy landscape so as to force the system to leave them. The time required for filling depends on the shape of the wells and on the number of CVs used, and can be reduced optimizing the hills width and height. However, increasing too much the width leads to a loss of resolution, and increasing too much the height leads to larger fluctuations in the free-energy estimate and potentially to systematic errors. If a parallel machine is available, the filling speed can be increased by performing more simulations at the same time, with the multiple-walkers algorithm[**?**].

In the multiple-walkers algorithm, several metadynamics simulations are performed at the same time using the same CVs (and, usually, the same hills parameters). Each replica is adding his own hills to a different HILLS file, but at the same time it is also feeling the force due to the hills added by other walkers. It has been shown[**?**] that in this manner the statistical error in the free energy estimation is expected to be independent on the number of walkers, whereas the filling speed grows proportionally to the number of walkers, provided they sample the space independently from each other. The (minimal) communication between replicas is just performed through the filesystem, and

there is no need to synchronize the simulations. They can even be run on machines of different type, provided there is a common file system.

As a first test, run the `ala3` system with the following `plumed.dat`:

Example.

```
phi: TORSION ATOMS=11,13,15,21
psi: TORSION ATOMS=21,23,25,31

mtd: METAD ARG=phi,psi SIGMA=0.35,0.35 HEIGHT=0.25 PACE=100 TEMP=300 BIASFACTOR=7

PRINT ARG=phi,psi,mtd.bias STRIDE=100 FILE=COLVAR

ENDPLUMED
```

Then estimate the free energy landscape.

To use multiple walkers, you should set up an independent directory and use an independent `plumed.dat` file for each walker. Here is a template `plumed.dat` which can be used with the `ala3` system.

Example.

```
phi: TORSION ATOMS=11,13,15,21
psi: TORSION ATOMS=21,23,25,31

METAD ... ARG=phi,psi HEIGHT=0.25 PACE=100 SIGMA=0.35,0.35
TEMP=300 BIASFACTOR=7
WALKERS_N=10 WALKERS_ID=@id@ WALKERS_DIR=../ WALKERS_RSTRIDE=1000 LABEL=mtd ...
METAD
PRINT ARG=phi,psi,mtd.bias STRIDE=100 FILE=COLVAR

ENDPLUMED
```

The @id@ string is then substituted with the proper walker number in each of the `plumed.dat` files:

Example.

```
for((i=0;i<4;i++)) ; do
mkdir $i
cp topol.tpr $i/
sed "s/@id@/$i/" plumed.dat > $i/plumed.dat
done
```

Each walker shoud receive a unique identifier (`ID` keyword) and will write the hills in a file `HILLS.X` where `X` is the walker number. The file will be placed in the upper directory (`WALKERS_DIR` keyword). Each replica should also be instructed about the maximum number of allowed walkers, so as to know which HILLS files need to be searched for (`WALKERS_N` keyword). This is just an upper estimate of the number of actual replicas, which can be even decided a posteriori. Just avoid to put "WALKERS_N=1000000", otherwise the system will spend a lot of time in trying to open not-existent files. The `WALKERS_RSTRIDE` keyword allows to set the number of steps between subsequent trial to load new hills added by the other walkers. This number should be as small as possible provided that there is no impact on the performance due to input/output overhead. A thousand step should be a reasonable value.

To run the simulation, just type:

Example.

```
for(i=0;i<4;i++)
do
cd $i
mdrun -plumed plumed.dat &
cd ../
done
```

You can then follow the trajectories in the `run0/COLVAR`, `run1/COLVAR`, `run2/COLVAR` and `run3/COLVAR` files. In this example all the walkers are starting from the same configuration.

Thus, at the beginning, they will not be independent from each other, and this could lead to an initially large error in the free energy estimate. A possible manner to decrease this initial error is to wait some time between starting the walkers (a few thousand steps), and use different seeds to initialize the velocities on each walker.

Finally, combine the hills in a single file with:

Example.
```
cat HILLS.0 HILLS.1 HILLS.2 HILLS.3 | sort -n > HILLS
```

and plot the free energy landscape.

4.3 Parallel-tempering metadynamics

This algorithm is only implemented for GROMACS.

One of the problems of metadynamics is related to the difficulty in choosing the collective variables. As an example, let us consider again the `ala3` case, but instead of using the two ϕ angles as CVs we use the end-to-end distance of the oligo-peptide. This distance is related to the backbone dihedrals, but is not a good CV, as it can be seen running metadynamics on it with the following input

Example.
```
d1: DISTANCE ATOMS=11,31
# we just monitor the two dihedral angles
phi: TORSION ATOMS=11,13,15,21
psi: TORSION ATOMS=21,23,25,31

mtd: METAD ARG=d1 HEIGHT=0.1 PACE=100 SIGMA=0.1 TEMP=300 BIASFACTOR=10

PRINT ARG=d1,phi,psi,mtd.bias STRIDE=100 FILE=COLVAR

ENDPLUMED
```

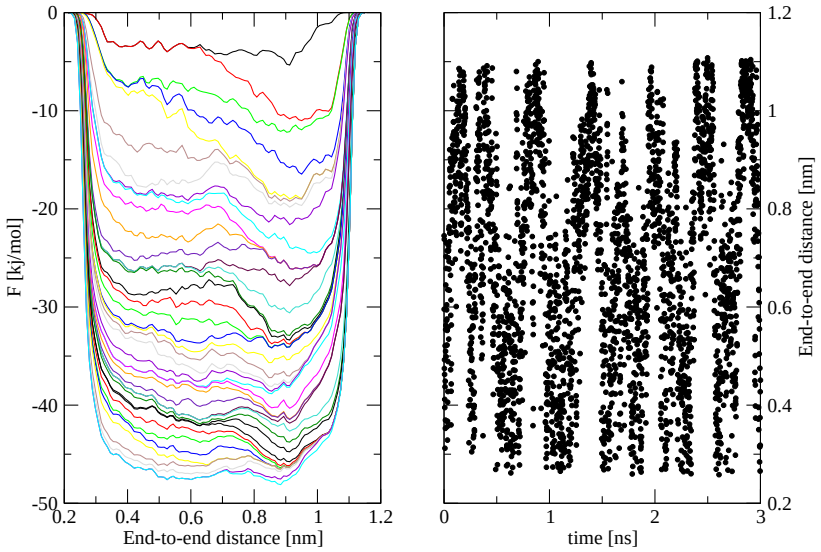


Figure 4.1: (left) Free energy profiles of ala3 as a function of the end to end distance, plotted with a stride of 1000 hills (i.e. 20 ps), obtained with serial metadynamics (right) Time series of the end-to-end distance. From both plots, one can clearly see that there is a strong histeresis, which is a signature of a poorly chosen collective variable.

4.3.1 Parallel tempering

A possible manner to accelerate sampling which is totally independent from metadynamics is parallel tempering. In parallel tempering N independent replicas of the system are run at different temperatures. From time to time an exchange of coordinates is tried in a Monte Carlo fashion, with acceptance (for replicas i and j)

$$P = \min\left(1, \exp^{\Delta\beta\Delta U}\right) \tag{4.1}$$

where $\Delta U = U_i - U_j$ and $\Delta\beta = 1/(k_B T_i) - 1/(k_B T_j)$. This allows each trajectory to walk up and down in temperature space. When evolved at high temperature, the system is able to cross barriers. When annealed down to room temperature, the correct canonical distribution is recovered again.

To run a parallel tempering simulation with GROMACS just use the following command:

Example.
```
mpirun -np 16 mdrun -multi 8 -replex 100
```

Here you will have 8 replicas running on 16 processors (i.e. 2 processors per replica), performing a trial exchange every 100 MD steps. Each output file will be suffixed with the replica number. 8 input files will be necessary for gromacs, one per replica. These input files can be generated using the following script

Example.
```
REPLICAS="300 380 460 540 620 700"

for rep in $REPLICAS
do
sed "s/@temp@/$rep/" grompp.mdp > grompp$i.mdp
grompp -f grompp$i.mdp -o topol$i.tpr -c conf$i.gro
done
```

Notice that here the file `grompp.mdp` is used as a template to build the real input files `grompp0.mdp`, `grompp1.mdp`, ... Thus, instead of the room temperature, ones should set the temperature to `@temp@`:

Example.
; extract from grompp.mdp
...
ref_t=@temp@
gen_temp=@temp@
...

Again (similarly to the multiple-walkers case), the replicas are starting from the same configuration and are correlated. To accelerate the decorrelation, it might be better to use different starting velocities on the different replicas.

Temperatures should be tuned so as have an uniform acceptance ratio across the replicas. An simple tool to generate them for atomistic simulations in explicit water is available here `http://folding.bmc.uu.se/remd/`.

4.3.2 The best from both worlds

Since parallel-tempering is accelerating all the degrees of freedom, it can be optimally combined with metadynamics, where only a few selected CVs need to be chosen. When parallel-tempering is combined with metadynamics[**?**], several metadynamics simulations are performed in parallel at different temperatures. Each simulation is thus building its own bias, which tends to compensate for its own free-energy surface. When exchanges are tried, the acceptance needs to take into account the fact that the bias potentials are different, so that the acceptance is now:

$$P = \min\left(1, \exp^{\Delta\beta\Delta U} \exp^{\beta_1 V_{b,1}(s_1)+\beta_2 V_{b,2}(s_2)-\beta_1 V_{b,1}(s_2)-\beta_2 V_{b,2}(s_1)}\right) \quad (4.2)$$

where $V_{b,i}(s_j)$ is the bias in replica i calculated at the coordinates of replica j.

To run parallel-tempering-metadynamics you don't need to add anything to the `plumed.dat` input file and run GROMACS as follows

Example.
mpirun -np 16 mdrun -multi 8 -replex 100 -plumed plumed.dat

8 hills file and 8 colvar files will be created, named `COLVARX` and `HILLSX`, with `X` the replica index. All the replicas will run in the same directory and acccess to the same plumed.dat file. Notice that hills will be rescaled proportionally to system temperature, thus adding larger hills to high temperature replicas. The free energy profile can be reconstructed for each replica using `plumed sum_hills` in the usual manner. Notice that with this approach one can reconstruct at the same time the free energy surface at different temperatures.

4.4 Bias-exchange metadynamics

This algorithm is only implemented for GROMACS.

In all variants of metadynamics the free-energy landscape of the system is reconstructed by gradually filling the local minima with gaussian hills. The dimensionality of the landscape is equal to the number of CVs which are biased, and typically a number of CVs smaller than three is employed. The reason for this is that qualitatively, if the CVs are not correlated among them, the simulation time required to fill the free-energy landscape grows exponentially with the number of CVs. This limitation can be severe when studying complex transformations or reactions in which more than say three relevant CVs can be identified.

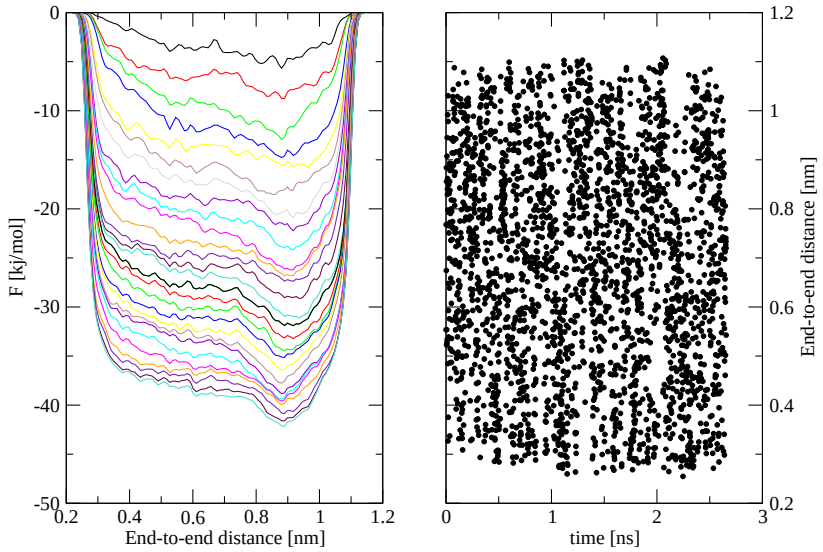


Figure 4.2: (left) Free energy profiles of ala3 as a function of the end to end distance, plotted with a stride of 1000 hills (i.e. 20 ps), obtained with parallel-tempering metdynamics. (right) Time series of the end-to-end distance. Histeresis is strongly decreased with respect to serial metadynamics.

A possible technique to overcome this limitation is parallel-tempering metadynamics, in the last Section. A different solution is performing a bias-exchange simulation [**?**, **?**]: in this approach a relatively large number N of CVs (say 10) is chosen to describe the possible transformations of the system (e.g., to study the conformations of a peptide one may consider all the dihedral angles between amino acids). Then, N metadynamics simulations (replicas) are run on the same system at the same temperature, biasing a different CV in each replica. Normally, in these conditions, each bias profile would converge very slowly

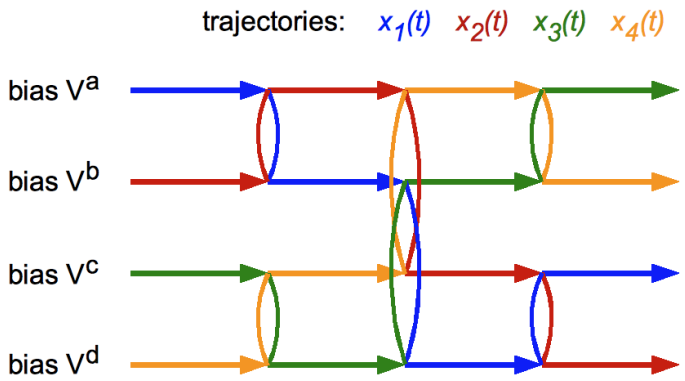


Figure 4.3: Schematic representation of a bias-exchange simulation employing four replicas and four bias potentials.

to the equilibrium free-energy, due to hysteresis. Instead, in the bias-exchange approach every fixed number of steps (say 10,000) an exchange is attempted between a randomly selected pair of replicas a and b. The probability to accept the exchange is given by a Metropolis rule:

$$\min\left(1, \exp\left[\beta(V_G^a(x^a,t) + V_G^b(x^b,t) - V_G^a(x^b,t) - V_G^b(x^a,t))\right]\right) \quad (4.3)$$

where x^a and x^b are the coordinates of replicas a and b and $V_G^{a(b)}(x,t)$ is the metadynamics potential acting on the replica $a(b)$. Each trajectory evolves through the high dimensional free energy landscape in the space of the CVs sequentially biased by different metadynamics potentials acting on one CV at each time (see scheme in Fig. 4.3). The results of the simulation are N one-dimensional projections of the free energy, whose convergence is monitored as usual: if the chosen CVs describe all the slow degrees of freedom, after the filling time each V_G grows evenly. In the following example, a bias-exchange simulation is performed on a Ala-Ala-Ala peptide (zwitterionic form, in vacuum with $\epsilon = 80$, force field amber03), using the four backbone dihedral

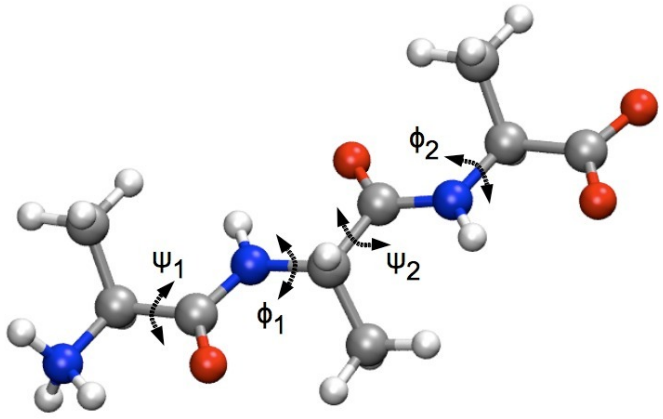


Figure 4.4: Ala_3 peptide and dihedral angles used as CVs for bias-exchange metadynamics.

angles as CVs (see Fig. 4.4): $\psi_1 = (N^1\text{-}C^1_\alpha\text{-}C^1\text{-}N^2)$, $\phi_1 = (C^1\text{-}N^2\text{-}C^2_\alpha\text{-}C^2)$, $\psi_2 = (N^2\text{-}C^2_\alpha\text{-}C^2\text{-}N^3)$, $\phi_2 = (C^2\text{-}N^3\text{-}C^3_\alpha\text{-}C^3)$. Four replicas of the system are employed, each one biased on a different CV, thus four similar Plumed input files are prepared, or even better a single common input file included in four different plumed files as follows:

Example.
common input file plumed-common.dat for bias-exchange on Ala_3

```
RANDOM_EXCHANGES
# psi1
psi1:  TORSION ATOMS=1,5,11,13
# phi1
phi1:  TORSION ATOMS=11,13,15,21
# psi2
psi2:  TORSION ATOMS=13,15,21,23
# phi2
phi2:  TORSION ATOMS=21,23,25,31

FLUSH STRIDE=500
```

Example.
first input file plumed.dat.0 for bias-exchange on Ala_3

```
# 1 kJ/mol (gromacs units) every 4 ps (timestep = 1 fs)
INCLUDE FILE=plumed-common.dat
mtd:  METAD ARG=psi1 HEIGHT=1.0 PACE=4000 SIGMA=0.314
PRINT ARG=psi1,phi1,psi2,phi2,mtd.bias STRIDE=500 FILE=COLVAR
ENDPLUMED
```

Example.
second input file plumed.dat.1 for bias-exchange on Ala_3

```
INCLUDE FILE=plumed-common.dat
mtd:  METAD ARG=phi1 HEIGHT=1.0 PACE=4000 SIGMA=0.314
PRINT ARG=psi1,phi1,psi2,phi2,mtd.bias STRIDE=500 FILE=COLVAR
ENDPLUMED
# 1 kJ/mol (gromacs units) every 4 ps (timestep = 1 fs)
```

Example.
third input file plumed.dat.2 for bias-exchange on Ala_3

```
# 1 kJ/mol (gromacs units) every 4 ps (timestep = 1 fs)
INCLUDE FILE=plumed-common.dat
mtd:  METAD ARG=psi2 HEIGHT=1.0 PACE=4000 SIGMA=0.314
PRINT ARG=psi1,phi1,psi2,phi2,mtd.bias STRIDE=500 FILE=COLVAR
ENDPLUMED
```

Example.
fourth input file plumed.dat.3 for bias-exchange on Ala_3

```
# 1 kJ/mol (gromacs units) every 4 ps (timestep = 1 fs)
INCLUDE FILE=plumed-common.dat
mtd:  METAD ARG=phi2 HEIGHT=1.0 PACE=4000 SIGMA=0.314
PRINT ARG=psi1,phi1,psi2,phi2,mtd.bias STRIDE=500 FILE=COLVAR
ENDPLUMED
```

Note that Plumed automatically enforces the periodicity of the CV `TORSION` between (-π,π) (e.g., when the system is close to π it feels also the hills which were put close to -π). The four replicas start from the same Gromacs topology file (even if it is not necessary), replicated four times (`topol0.tpr, topol1.tpr, topol2.tpr, topol3.tpr`). Finally, Gromacs is launched as a parallel run on 4 cores, with one replica per core, with the command

Example.
```
mpirun -np 4 mdrun -plumed plumed -multi 4 -replex 5000
```

where `-replex 5000` indicates that every 5000 molecular-dynamics steps all replicas are randomly paired (e.g. 0-2 and 1-3) and exchanges are attempted between each pair (as printed in the Gromacs `*.log` files). The frequency by which exchanges are attempted is a parameter of the simulation which should be optimized for the specific problem at hand, but different benchmarks show that the convergence of the simulation is robust with respect to it [**?**].

4.4.1 Convergence of the simulation

The convergence of the one-dimensional free-energy profiles (parallel growth) can be monitored by plotting the sum of the hills at diferent times with the program `sum_hills.x`:

Example.
generate FES profiles from HILLS.0 every 100 hills (important: -pi 1 declares a (-π,π) periodic range):
```
plumed sum_hills --hills HILLS.0 --stride 100
```

plot the FES profiles fes.dat.1, fes.dat.2, etc.:
```
xmgrace fes.dat*
```

The parallel growth of the bias profiles at different times for each CV is an indication that the simulation is converged. However the best final estimate of the free energy profile is obtained by averaging the instantaneous bias profiles after the filling time:

$$F(s) \approx -\frac{1}{t - t_F} \int_{t_F}^{t} dt \; V_G(s, t) \tag{4.4}$$

as discussed in Ref. [**?**, **?**]. The resulting estimate of the FES profile can be finally compared to the files `EQUIL_FESx` which contain the "exact" FES computed from a very long (3 μs) equilibrium MD simulation without any bias (Fig. 4.5). In the latter case the free energy along each dihedral angle is simply obtained as $F(s) = -k_B T \log(p(s))$, with s a dihedral angle. As one can notice, the agreement is excellent (at a fraction of the computational cost!), and the bias-exchange profile has a smaller uncertainty on the barriers as there the equilibrium simulation has a very poor sampling.

An important feature of the bias-exchange technique is that, provided good CVs are employed, the computational cost scales approximately linearly with the number of CVs, even if the volume of the CV-space grows exponentially: this can be easily verified increasing the length of the peptide to Ala_4, Ala_5, etc. and using the 6, 8, etc. backbone dihedrals as CVs. In this case, using a proportional number of CPUs the simulation time per replica remains similar. However, as for all enhanced sampling techniques, very complex systems with a large number of metastable states (e.g., folding a protein of say 60 amino acids) represent a major challenge.

The output files of the bias-exchange simulation can be also used to reconstruct the fully four-dimensional free-energy landscape employing the weighted-histogram technique, as explained

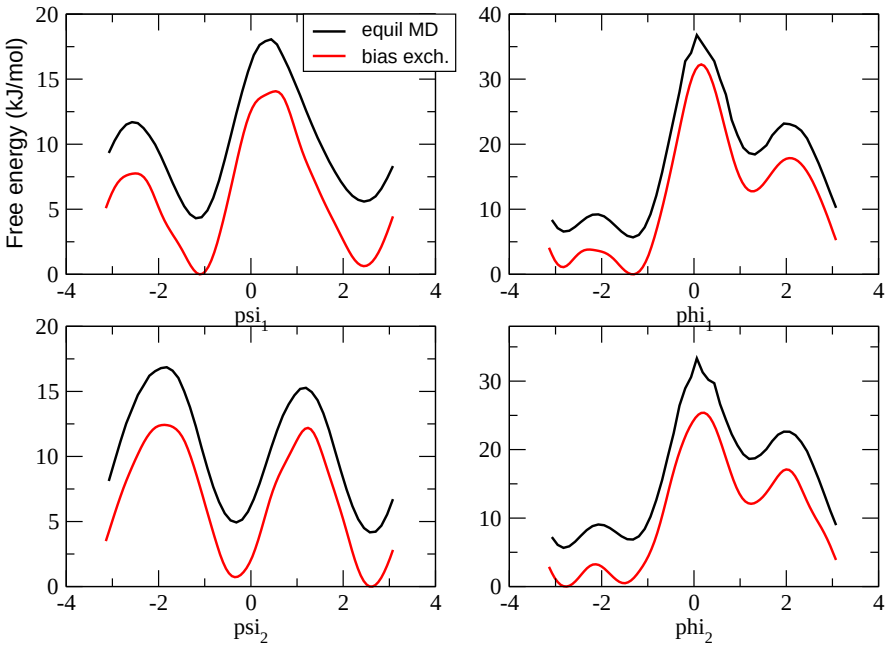


Figure 4.5: Comparison of Ala_3 free-energy profiles obtained from bias-exchange (4 replicas running for 4 ns) and from equilibrium MD (3 μs).

in Section 5.4.2.

Chapter 5

Advanced techniques

5.1 Path based collective variables

Path collective variables (PathCV) are very useful whenever one wants to find an optimal free energy channel connecting two specific regions in the phase space and calculate the associated free energy profile. Typical examples where the use of pathCV is beneficial are docking/undocking of a ligand from a target, complex chemical reactions and conformational changes in biomolecules. Whether or not "two specific regions" can be defined for an initial state and a final state depends on the nature of the system under study and the choice of the subspace in which the path is defined (coarse graining). For example one could think that the folding of a peptide is not a suitable case for pathCVs as the ' 'folded" state is easily defined, but the "unfolded" generally includes several different states. However, by suitably "remapping" the space in which the path is defined also the unfolded state may be univocally defined. A good choice for folding could be the use of the number of native hydrogen bonds. In this case the folded state would be the sate with the maximum number of native-like hydrogen bonds, while

the unfolded state would have no hydrogen bonds. The folding paths in this space will be all the energetically favorable changes of hydrogen bond patterns connecting the unfolded state to the unfolded state.[**?**] The level of coarsening in defining the path (i.e. the definition of the subspace in which the path is defined) has to be decided by the user according to its chemical intuition and it is not always easy.

A given point in the Cartesian space corresponds to a vector $\mathbf{\Theta}(x)$ in the coarse grained space (adopting the notation of Maragliano/Vanden-Eijnden). In particular the components of this vector are defined with $\theta_m(x)$, namely:

$$\mathbf{\Theta}(x) = (\theta_1(x), \theta_2(x), \ldots, \theta_N(x)) \tag{5.1}$$

where N is the number of coarse grained variables needed to simulate a given process. For example, in a chemical reaction it might turn to be useful to describe the reaction in terms of some h-bond distances and specific angles. Each of these is one $\theta_m(x)$. In `PLUMED` the choice is (so far) limited to three possibilities: `RMSD`, `DRMSD` and `CMAP`. In RMSD the representation of the configuration is done in terms of explicit Cartesian coordinates of the subset of atoms involved in the CV. In the `DRMSD` each configuration is represented in terms of a set of specific distances. In the `CMAP` representation each configuration is represented in terms of a set of contacts (whose value depends on specific distances through a tunable switching function).

Once the subspace in which the path is defined has been chosen, a reference path in this subspace must be defined. In `PLUMED` the reference path is defined in terms of a number M of points (called "frames") along this parametric functions, say:

$$\mathbf{\Theta}_{path}(i) = (f_1(i), f_2(i), \ldots, f_N(i)). \tag{5.2}$$

The number of frames defining the path depend on the length scale of the process. An important consideration is that the frames must be as equally spaced (in the current metric) as possible. PLUMED can run with paths whose frames are unequally spaced, but this can result is a poor reconstruction of the free energy profile.

Once a reference path is defined (see also below) two collective variables can be calculated in PLUMED. The first returns the projection of the current phase-space vector on the reference path. This tells us if we are closer to the reactant state, or to the product state or if we are somewhere in between. This CV is defined as it follows:

$$S(x, \{\mathbf{\Theta}_{path}(m)\}_{m=1,M}) = \frac{\sum_i i \ \exp -(\lambda|\mathbf{\Theta}_{path}(i) - \mathbf{\Theta}(x)|)}{\sum_i \ \exp -(\lambda|\mathbf{\Theta}_{path}(i) - \mathbf{\Theta}(x)|)} \quad (5.3)$$

that is a function which returns a fractional corresponding to the closest point along the path. If the closest point is 2 then its exponential will be larger respect to the other ones and therefore the CV will have a value close to 2. However, the progress along this CV alone can be misleading. If our system is very far from the pre-defined path or if it takes a tangential path, we might still see that S smoothly varies from 1 to M while in reality it is following a different path. For this reason is very useful to measure a distance from the reference path, defined as it follows:

$$Z(x, \{\mathbf{\Theta}_{path}(m)\}_{m=1,M}) = -\frac{1}{\lambda} \ln \sum_i \ \exp -(\lambda|\mathbf{\Theta}_{path}(i) - \mathbf{\Theta}(x)|) \quad (5.4)$$

which is the distance along the path. Consider once again the case that one exponential is larger than the other then it is easy to retrieve from the formula the distance of the closest frame.

Now we can discuss how the parameter λ is chosen and how a good set of frames $\{\mathbf{\Theta}_{path}(m)\}_{m=1,M}$ defining the reference path is found. Our rule of thumb for λ is to use the following formula:

$$\lambda = 2.3/ < |\mathbf{\Theta}_{path}(i) - \mathbf{\Theta}_{path}(i+1)| > . \tag{5.5}$$

The reason for that is simple. The S variable should change smoothly from a frame to another. To obtain this effect the exponentials centered on each of the M frames must overlap. That is, even if you are precisely on one frame, the tails of the neighbor exponential must not vanish. If any two frames are distant Δ in our subspace, we want to tune lambda so that the exponential value of the neighbor is around 0.1.

$$0.1 = \exp(-\lambda\Delta) \tag{5.6}$$

which, by inversion, gives exactly the value reported above. If you are not able to find a set of frames that have a uniform spacing then you must calculate the value of `LAMBDA` according the largest inter-nodal spacing. This might limit the resolution the reconstructed free-energy profile.

Another important consideration is that the frames should reproduce a parametric curve in a reliable way. That is, the closest neighbors to point i in the sequence (which are $i-1$ and $i+1$) must be the closest also in the chosen space.

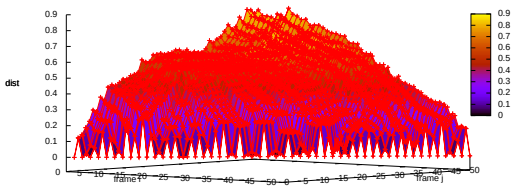
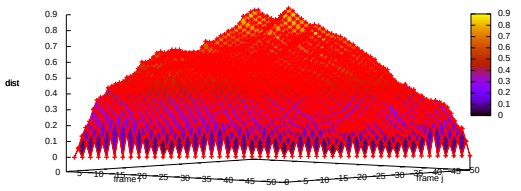
$$|\mathbf{\Theta}_{path}(i)-\mathbf{\Theta}_{path}(i+1)| = |\mathbf{\Theta}_{path}(i)-\mathbf{\Theta}_{path}(i-1)| < |\mathbf{\Theta}_{path}(i)-\mathbf{\Theta}_{path}(j)| \tag{5.7}$$

with $j < i-1$ or $j > i+1$ and

$$|\mathbf{\Theta}_{path}(i-1)-\mathbf{\Theta}_{path}(i+1)| \simeq 2|\mathbf{\Theta}_{path}(i)-\mathbf{\Theta}_{path}(i-1)|. \tag{5.8}$$

This last condition can be difficult to fulfill if we are in a non-euclidean space. Sometimes it helps to add more points along the path.

To judge how good is the parameterization of the path we can visualize the all-against-all frame distances in the chosen subspace.

Badly parameterized path	Well parameterized path
	

In Fig. 5.1 it is evident that the distance of each frame with its neighbors is very bad (the diagonal elements are irregular and their distance changes a lot along the path) while in 5.1 the diagonal elements are much better spaced and uniform along the path. The way to produce a well parametrized path may vary a lot. The simplest choice is to just take a set of equally spaced frames along a straight line connecting the initial and final state. This choice, albeit it is simple, has some drawbacks as we will see in the first exercise. A more systematics choice is to select the frames from a trajectory produced with `PLUMED` using a steered MD or targeted MD simulations. This will ensure that the frames that you use correspond to accessible states in the phase space. Out of this set you can choose the most equidistant points along the path via a Monte Carlo procedure. This is an effective but ad-hoc solution. We recommend you to find your own effective protocol that will be be system dependent.

A different possibility is to use the finite temperature string method procedure described in Ref. [**?**]. It produces equidistant frames. But it has the drawback of not taking into account

the curvature of the space and moreover it may produce fictitious interpolated points that may not correspond to physical situation.

Another important point is the choice of the "metrics". This means the way in which we calculate the distances $|\mathbf{\Theta}_{path}(i) - \mathbf{\Theta}(x)|$. The adopted metrics in `PLUMED` depends on the chosen representation. For example, in the `RMSD` representation the distance between the running simulation frame and one reference frame is calculated as a sum of the distances squared after optimal alignment of the two structures through Kearsley[**?**] alignment method. In case of `CMAP` it is performed by summing up the difference in contacts between the reference and the running frame, squared and similarly in `DRMSD` the sum of the differences (squared) between two different set of distances (running frame vs one of the references)are calculated.

`PLUMED` does not yet implement a general scheme for mixing variables because one one side these could be rather tricky due to the inherent difficulty of managing different CVs within one single CV but on the other side you might think that this can offer a safe approach to the users (and tested). If you are interested in all the tricks behind a general variable which is function of other variables please refer to [**?**].

Now, let's come to the more practical part of this section.

5.1.1 Metadynamics on a sub-optimal path

The first exercise that we propose is to try is a metadynamics run with pathCV on a path that does not correspond to a "low-free energy channel". Out of time consideration, we will make use of alanine dipeptide for this example.

Its free energy landscape is conveniently represented by two

angles that are called Ramachandran angles and are generally useful because specific secondary structures of protein present specific combinations of these two dihedrals. The plot of these two dihedral is called "Ramachandran plot". A free energy for this small peptide as function of these two dihedrals resembles the one showed in 5.1. It is visible a pathway that leads to a minima which is located at $\Phi \simeq -2.6, \Psi \simeq 2.8$ which is called C_{ax} and another one located at $\Phi \simeq 1, \Psi \simeq -0.5$ which is connected to another minimum called C_{eq}.

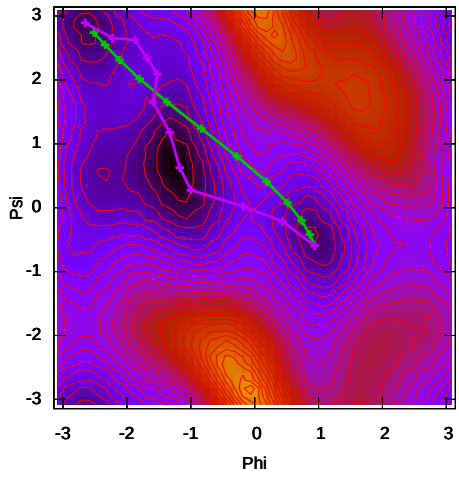


Figure 5.1: The free energy landscape of an alanine dipeptide with the GROMOS96 allatom force field. Iso-lines are drawn every 1kcal/mol. Two paths are traced. A "bad" path is in green and is obtained by straight interpolation. In pink an optimized path is shown.

Let's assume that those two minima are known. In this case one could naively imagine that the typical pathway would be something that leads from C_{ax} to C_{eq} via a linear path. This

means that we can take a linear interpolation of the Cartesian coordinates that go from the reactant (C_{ax}) to the product state (C_{eq}) and simply assume this as a reactive pathway (see green pathway in Fig. 5.1). Of course we can immediately imagine that, although the two extrema are stable and low energy, some intermediate states are rather unhappy to lie there and we might expect to find an energetic funnel which is not running all the way close to the initial guess. In PLUMED the input for path collective variables is as follows:

Example.
The following is an example of input file for a path collective variable run.

```
PRINT W_STRIDE 100
#
# a rather rude metadynamics.  do not take this as production conditions
#
HILLS W_STRIDE 200.0 HEIGHT 0.4
#
# the two path variables #
S_PATH TYPE RMSD FRAMESET frame_ NFRAMES 12 LAMBDA 10300 SIGMA 0.3
Z_PATH TYPE RMSD FRAMESET frame_ NFRAMES 12 LAMBDA 10300 SIGMA 0.002
#
# additional ramachandran variables might be useful to understand what is happening
#
TORSION LIST 5 7 9 15
TORSION LIST 7 9 15 17
ENDMETA
```

The syntax for path collective variables consists of a initial directives S_PATH or Z_PATH.

S_PATH is the CV that gives the progress along the path while Z_PATH gives the distance from the path itself. Then a keyword is required that is the TYPE followed by the type. In this example is RMSD and this determines the format for the input that one has to provide. In this case it is a simple pdb format. In this particular type it is very important that the index in the second column of the pdb correspond to the absolute indexing

within the program. For example, in NAMD the indexing must correspond to that of the same atoms in the formatted .coor. In SANDER I dump a pdb from the restart or coordinate file with ptraj and use that indexing. Similarly in GROMACS I use trajconv and dump a pdb and use that indexing. Generally it works. Labels and residue name and number can be different from the original one because `PLUMED` is not aware of those information to double check (in many programs there are not easy to retrieve). A last note for the `RMSD` keyword. The last column of the pdb (generally beta and occupancy) are here used to specify the weight to use in the alignment and the displacement. For example, consider a case of protein-ligand docking: in this case you want to measure the progress of the detachment of the ligand from the target by using the protein as reference system. In this case the atoms of the protein have to be included in the beta column with a value of 1.00 while the atoms of the ligand should be marked with a 0.00. The fact that the only thing that one wants to measure are the atoms of the ligand are denoted by reporting a 1.00 for the atoms of the ligand in the occupancy column while putting the atoms of the protein used for of the alignment to 0.00. It is also allowed to use the same atom for alignment and the measure. The `LAMBDA` determines the fudge factor for the summation of the exponential for the formulas of the path. See equation 5.6 and note that in GROMACS the units are nm^2 (quite common pitfall!!!). This example is performed with GROMACS. Note that in case of `RMSD` it is much better you use the double precision version of the code. Therefore it consists in the following steps.

- Modify the md.mdp (which is the GROMACS input parameter file) according to your needs. Here is the point where

you should provide the number of steps. Note that in this case we use fully flexible hydrogens and no constraints on them. This is an accepted standard for alanine dipeptide.

- Generate the `topol.tpr` via `grompp` GROMACS preprocessing program. This file is the only one needed to run a plain run with `mdrun`. In case of a run with `PLUMED` you should specify that you use `PLUMED` at runtime and use the input reported above. For example it is likely that by running `mdrun -plumed metadyn` (where`metadyn.dat` is the metadynamics input file reported above) will work if your `mdrun` version is compiled with `PLUMED` .

The script file is therefore something like this

Example.

```
grompp -f md.mdp -c 2ala.gro -p gromacs.top
mdrun -plumed metadyn
```

I find personally very helpful to have a look to the `COLVAR` file and `HILLS` file while are being produced. Just by asking yourself if what you see is what you expect you can double check if your input file is correct. What you might see from this experiment is that `COLVAR` (column 2:3, plot with gnuplot and a command like `plot "COLVAR" u 2:3 w p`) you have a plot like the one reported below in Fig. 5.2.

Another additional test you could try to do yourself is to try to reduce the `LAMBDA` of a couple of orders of magnitude and try to increase it. What do you observe? (Large `LAMBDA` "atomize" regions together and hills width become strongly anisotropic. Small value lump all the space as it was one. Is this expected? Give a look to the equations of path CVs to make up your mind about that).

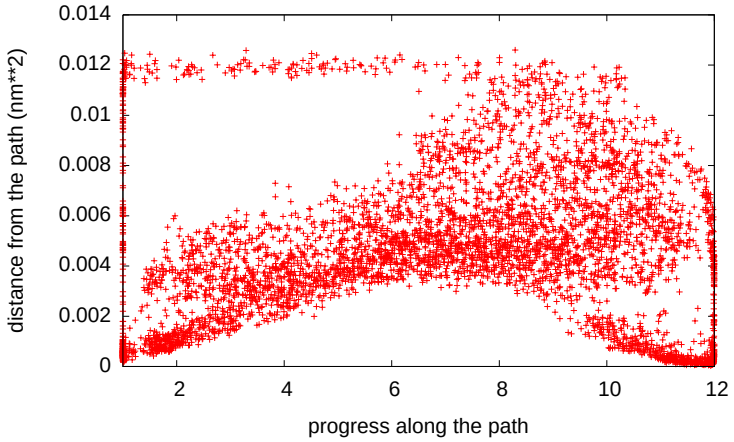


Figure 5.2: Progress and distance from the path in case of wrong path. The system does not follow the path closely but most of time is spent far away from the guess path.

An important observation, the most important of all, is that for most of the progress along the path your system does not follow the path closely (Ideally the system follows the path when most of time stays at distance of the path$\simeq$ 0). This is crucial because it underlines that the guess pathway does not reproduce the reactive event as imagined. Fine, but you see the transition. So why bother? The fact is that states (and transition states) at large distance from the path include a number of structures that may be quite diverse. Therefore the transition state you might get is not composed of structures that have 0.5 probability of falling into the reactant before falling into the products (which is a "coarse" but correct definition of dynamical transition state). The result is that the free energy can be lower than expected from the experiment (because you lump in the TS a set of states that may be pre and post TS which, by definition, have lower energy). For these reason, whenever your system does not like

to walk right on your path, please be suspicious and consider to adopt an improved version of the path (more about how to obtain it below).

5.1.2 Metadynamics on a correct path

What happens when the path is almost good? In this example we do the same exercise as done previously (same input files. Just minor modifications) but on an improved pathway (the pink one in Fig. 5.1) . In case you change the pathway there are different options. One of that is just substitute the old reference files for the guess path (in the previous example their name was `frame_1.pdb`, `frame_2.pdb`, ..., `frame_12.pdb`) and remember to change the `LAMBDA`. This is mandatory because if the path changes but the number of guess point changes it is likely that the internodal distance change as well and the factor `LAMBDA` should be recalculated with the equation 5.6. In this case `LAMBDA` is 3800 that is a signal that now the pathway is longer. By repeating the same metadynamics calculation as before then you find a different landscape that is much better in that the path is strongly followed by the system during the metadynamics as an equivalent plot of the `COLVAR` file points out. As you might expect now, the plot you obtain is pretty similar to the one you obtained before. It is just a "deformation" of it. This is rather important. Path collective variables may create "ad-hoc" foliation of the space in an adaptive way and allow you to have many different representation, depending on your ability of retriving a good guess pathway.

Now that the system follows more closely the guess path you may hope that, by performing a committor analysis on the TS this gives a value of 0.5 and you can really trust your transitions

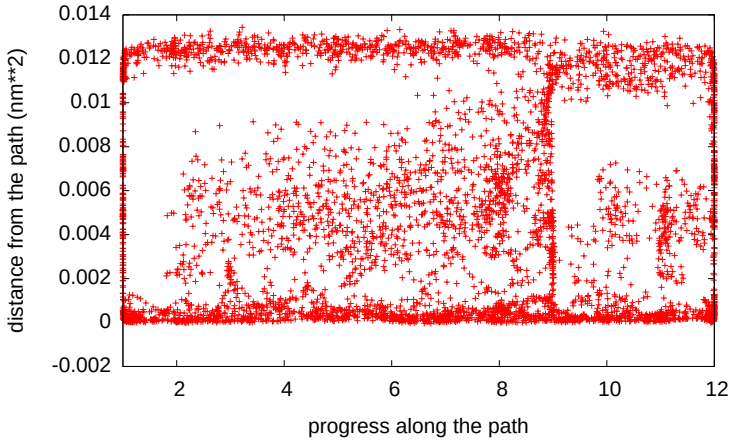


Figure 5.3: Progress and distance from the path in case of a correct path. The system does now follow closely the path.

state. Try to figure out, from the Fig. 5.1, why you see all those pathways and how you can "map" them onto ramachandran plot so to double check if what you see is just an effect of the projection of the states on the PCVs or those pathways you see have an equivalent on the Ramachandran plot. How is it possible that a periodic representation on the Ramachandran plot is now "unfolded" in a not periodic representation?

Finally, the cherry on the cake. Do a free energy with the usual command. `sum_hills -ndim 2 -ndw 1 2 -ngrid 100 100 -file HILLS` Before doing that, if you want to obtain meaningful results, you would be better to increase deposition time and simulation time as well. Three nanoseconds should be sufficient. The other way is to limit the path on the distance along the path and then limit the exploration only to the optimized region. This is reasonable as, for the reasons explained before, the regions at large distance are not well resolved. In this case we can still limit ourselves to just 100ps. A possible

input can be something like this:

Example.
The following is an example of input file for a free energy calculation along the path in a limited region of the path itself.

```
PRINT W_STRIDE 100
#
# a rather rude metadynamics.  do not take this as production conditions
#
HILLS W_STRIDE 400.0 HEIGHT 0.4
#
# the two path variables:  metadynamics only on S_PATH, the other is just for
monitoring
#
S_PATH TYPE RMSD FRAMESET frame_ NFRAMES 12 LAMBDA 10300 SIGMA 0.25
Z_PATH TYPE RMSD FRAMESET frame_ NFRAMES 12 LAMBDA 10300
#
# A wall over Z can be useful to limit the orthogonal exploration and speed up
convergence
#
U_WALL CV 2 LIMIT 0.0 KAPPA 100.0
#
# additional ramachandran variables might be useful to understand what is happening
#
TORSION LIST 5 7 9 15
TORSION LIST 7 9 15 17
ENDMETA
```

At the end you can integrate simply with the usual command `sum_hills -ndim 1 -ndw 1 -file HILLS` you can retrieve the free energy file `fes.dat` that you can easily plot with gnuplot and the command `p "./fes.dat" u 1:($2/4.186) w lp`. Note that the factor 4.186 is the kJoule to kcal conversion factor as the GROMACS internal energy units (and therefore hills height) are in KJoule/mol. What can you see? Can you compare with the free energy along the pink path in Fig. 5.1? Are you getting something you are expecting?

Now a final important remark . A very important point in understanding how to choose path collective variables is the choice of the metrics. In `PLUMED` you have the choice between `RMSD`, `DRMSD` and `CMAP`. However one can be tempted in sim-

ply producing an artificial path via `TARGETED` and then using the frames as an input for pdbs for the type `RMSD`. This is not always the best choice (very often it is not at all). Consider carefully your problem. If you expect large entropic basins probably something like `CMAP` is good for you as the degeneracy does not increase so dramatically as function of the distance from the path as for `RMSD`. The `PLUMED` developers team is working on a flexible scheme for introducing a flexible scheme for adopting a complete flexible metrics. A good reference could be [**?**] where we had to use two different metrics depending on the transition that we wanted to do.

5.1.3 Path optimization

An important question is how to obtain and improve the reference path. There are a number of ways to perform this task and a detailed treatment goes far beyond the scope of this tutorial. Anyway it is quite important to provide pointers to some relevant literature and briefly discuss the main issues that one might encounter. Historically path optimization was first performed in small molecules in electronic structure calculations.

In this regime, if the number of atoms is small (around 10-20 atoms), then it is possible to do a guided search on the potential energy surface (PES) and find the saddle point. This is in generally not completely automatic and requires some knowledge on the system to define some "meaningful" set of additional coordinates (called "redundant coordinates") that would increase the possibility of finding a saddle point on the PES. The saddle point is defined as the point in the configurational space that has zero force and whose Hessian matrix (the matrix of the second derivatives of the energy respect to the positions) has positive

eigenvalues but one. That specific negative eigenvalues is associated to an eigenvector that, when followed in two opposite directions, joins the reactant and the product. This is a standard strategy that is based on saddle point search and intrinsic reaction coordinate following which is implemented in many codes, among which the popular Gaussian code. However this approach has some severe limitations. First, when scaling up the number of atoms (with a solvent layer), this procedure becomes inefficient as many minima may interfere and the saddle point search may be stuck on non-useful unreactive saddle points. Moreover in this regime the free energy is hardly obtained with the rigid rotor approximation and many non-harmonic effect start to appear. similarly, adding another redundant coordinate might be of limited help because as the number of atoms increase, cooperatively starts to appear. These issues have been addressed with the "Nudged Elastic Band" (NEB) [**?**, **?**, **?**] approach where many replica of the same system, each one in a chain of guess states that go from the reactant to the product, are evolved together and each replica is chained to the next replica by a spring which keep each replica equidistant from the other (the so called "re-parametrization force").

It works well at T=0 K for chemical reaction. Here the trick is that in this way the path is "forced" to make the reaction and therefore one can find the closest local minima to the input guess path thus avoiding the situation in which the saddle point is found but it does not connect reactant and products.
What about T>0 K? In the recent years many techniques were introduced were the main task is, instead minimizing the path on the PES, one wants to minimize the path using the FES. The best known is the so-called "Finite temperature string method"

[**?**, **?**, **?**] . This technique can be easily implemented with `PLUMED` as it follows: create a number of replicas of the system along the path and put some umbrella on `S_PATH` on the nodal points (say 1.0, 2.0, 3.0, etc. . .) and a `U_WALL` on

`Z_PATH` so that each point does not escape dramatically. Then you might collect configurations and average them so to extract a new path. Re-parametrization requires some care and an ad-hoc code should be written for reparametrizing each metrics (Cartesian coordinates require optimal aliened in a correct way while angles require that correct periodicity is applied). Please refer to [**?**]. Another technique I have been applying is to use the NEB chain of springs [**?**, **?**, **?**] to produce an additional reparametrizing force and then evolve in a steepest descent fashion. For this approach please refer to [**?**].

Another possible approach is to performe a slow steered MD in slow regime with a limit on `Z_PATH` and then resorting the new frames from the trajectory. However the path found can be suboptimal if you pull too fast.

5.2 Variables for secondary structure in proteins

Many fundamental protein processes like folding, unfolding or misfolding to a pathological form involve transformations of the secondary structure. If we define the ϕ and ψ backbone dihedral angles as formed by atoms (C^1-N^2-C^2_α-C^2) and (N^1-C^1_α-C^1-N^2) respectively, then alpha secondary structure is localized in a region around (-60°,-45°) in the (ϕ,ψ) Ramachandran plot, whereas beta is in a region around (-135°,+135°) (see Fig. 5.4-a). Both alpha helix and beta sheet are characterized by hydrogen bonds

between the backbone C=O and N-H groups of different amino acid residues. In helices the hydrogen-bonded residue pairs are of the type $(i, i+n)$, $(i+1, i+n+1)$, $(i+2, i+n+2)$, etc., where $n = 3$, 4, or 5. Beta sheets are characterized by similar pairs as alpha but with larger n (parallel beta) or by pairs $(i, i+n)$, $(i-1, i+n+1)$, $(i-2, i+n+2)$, etc., with $n > 4$ (antiparallel beta). Further, the side chains of neighboring amino acids have a typical stacking: in the alpha helix they point outward, in the beta sheet they form pairs which are alternatively above or below the sheet.

In Plumed there are several CVs which can induce changes of secondary structure based on the previous structural features: the H-bonds (CVs `COORD`, `HBONDS`), the typical dihedrals (CVs `TORSION`, `DIHCOR`, `ALPHABETA`), or the detailed chain conformation (CVs `ALPHARMSD`, `ANTI/ PARABETARMSD`). Here we will illustrate only the CVs `ALPHABETA` and `xRMSD`. As protein model we consider a poly-valine peptide Val_{20} in vacuum (with ACE/NME capping groups, amber03 force field) as it allows to obtain realistic secondary structure: valine has a volume which is the average of all amino acids.

5.2.1 `ALPHABETA`

This CV measures the (approximate) number of backbone dihedral angles which are similar to a given target angle [**?**, **?**, **?**]. Each dihedral is specified by a set of four atom indexes followed by the target angle. Here we consider the ψ dihedrals (which distinguish better than ϕ between alpha and beta) defined by atoms (N^1-C^1_α-C^1-N^2), and a reference value of -45° = −0.7854 rad which corresponds to alpha helix. The CV value ranges between 19 (the number of ψ dihedrals in this example) when all

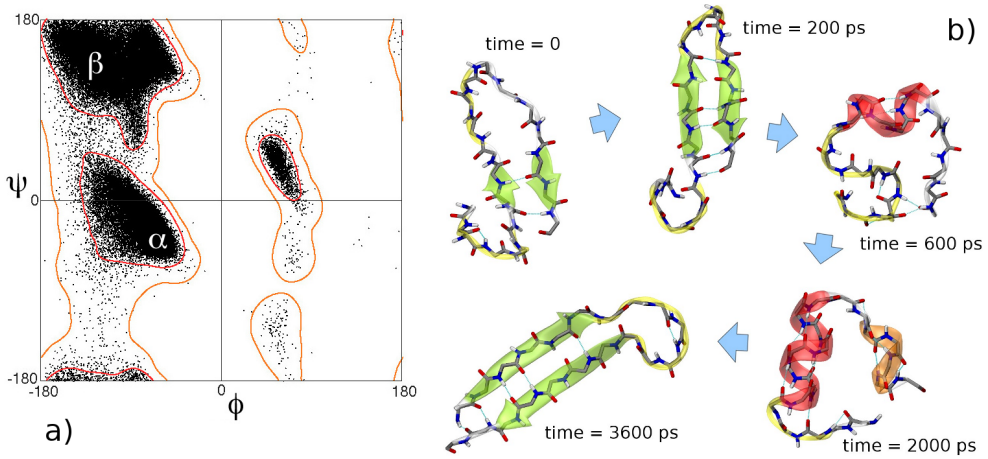


Figure 5.4: a) Ramachandran plot of backbone dihedral angles. b) structures explored during a simulation on the Val_{20} peptide using the CVs `ALPHARMSD` and `ANTIBETARMSD`.

$\psi = -45°$, down to 0 when all $\psi = 135°$. Therefore the CV is able to switch between all-alpha and all-beta structures: in this example approximately 2 ns are needed for a complete loop, but estimating the FES would require extended simulations due to the complexity of the conformational space of a 20-amino acid peptide (i.e. the number of possible structures is very large).

Example.
Input for secondary structure exploration in a Val_{20} peptide

```
# 5 kJ/mol (gromacs units) every 5 ps (timestep = 2 fs)
HILLS HEIGHT 5.0 W_STRIDE 2500
PRINT W_STRIDE 500

# psi dihedrals:  reference = -45 deg = -0.78540 rad
ALPHABETA NDIH 19 SIGMA 1.0
7 9 21 23 -0.78540
23 25 37 39 -0.78540
39 41 53 55 -0.78540
55 57 69 71 -0.78540
71 73 85 87 -0.78540
87 89 101 103 -0.78540
103 105 117 119 -0.78540
119 121 133 135 -0.78540
135 137 149 151 -0.78540
151 153 165 167 -0.78540
167 169 181 183 -0.78540
183 185 197 199 -0.78540
199 201 213 215 -0.78540
215 217 229 231 -0.78540
231 233 245 247 -0.78540
247 249 261 263 -0.78540
263 265 277 279 -0.78540
279 281 293 295 -0.78540
295 297 309 311 -0.78540

ENDMETA
```

5.2.2 `ALPHARMSD` and `ANTIBETARMSD`

This class of CVs counts the (approximate) number of 3+3-residue blocks which are similar to the ideal alpha helix or beta sheet (i.e., to the average experimental structures) [**?**]. The similarity with respect to the ideal secondary structure is estimated by the root mean square deviation between the distance matrices among backbone N, C_α, C, O and C_β atoms (this is equivalent to the root mean square cartesian distance in ref [**?**]). These CVs are more efficient than the dihedral-based `ALPHABETA` to observe the formation of beta sheets in explicit solvent simulations of larger proteins.

Example.
Input for secondary structure exploration in a Val_{20} peptide

```
# 5 kJ/mol (gromacs units) every 5 ps (timestep = 2 fs)
HILLS HEIGHT 5.0 W_STRIDE 2500
PRINT W_STRIDE 500

# with gromacs4+domain decomp.  it's important to use this:
ALIGN_ATOMS LIST <secstr>
secstr->
# N CA C O CB
7 9 21 22 11
23 25 37 38 27
39 41 53 54 43
55 57 69 70 59
71 73 85 86 75
87 89 101 102 91
103 105 117 118 107
119 121 133 134 123
135 137 149 150 139
151 153 165 166 155
167 169 181 182 171
183 185 197 198 187
199 201 213 214 203
215 217 229 230 219
231 233 245 246 235
247 249 261 262 251
263 265 277 278 267
279 281 293 294 283
295 297 309 310 299
311 313 325 326 315
secstr<-

# 0.1 = conversion from nm (gromacs) to angstrom (internal reference structures)
ALPHARMSD LIST <secstr> SIGMA 0.3 R_0 0.08 NN 8 MM 12 ANGSTROM_SCALE 0.1 STRANDS_CUTOFF
1.  NOPBC

ANTIBETARMSD LIST <secstr> SIGMA 0.3 R_0 0.08 NN 8 MM 12 ANGSTROM_SCALE 0.1
STRANDS_CUTOFF 1.  NOPBC

ENDMETA
```

In the latter input file the `R_0`, `NN`, `MM` parameters are optimized to count ~ 1 only if the structure is similar to the ideal one, while `STRANDS_CUTOFF 1` limits the computation of the CVs to protein segments which are not farther than 1 nm from each other, for efficiency, and finally `NOPBC` prevents the pairing of beta strands between periodic replicas of the protein (even if in the present example the system is isolated, without pbc).

As a list of 20 residues is given in input, the CV measures the total content of alpha or antiparallel beta secondary structure in the protein, without telling the specific position in the chain. Therefore e.g. a value of `ALPHARMSD` equal to 4 may correspond to a short helix at the beginning, at the center, or at the end of the protein. By using several times the CVs for different chain segments it is possible to localize the secondary structure in specific regions. In a few ns different secondary structure elements can be observed (Fig. 5.4-b). Estimating the FES, as explained above for `ALPHABETA`, would however require extended simulations due to the complexity of the conformational space.

5.3 Potentials on a grid

As the simulation goes on, the computational time spent in the evaluation of the metadynamics contribution to the forces becomes larger and larger and eventually comparable with the time needed to calculate the main forces in the MD code. This effect is particularly visible when the system simulated is small or when using a simplified coarse-grained potential.

A possible solution to this problem is storing an array containing the current value of the bias potential (and of the derivatives with respect to the CVs) on a grid. In this way the computational cost of metadynamics becomes constant during the simulation. This corresponds to the cost of evaluating a single Gaussian function on the whole grid with a frequency given by the stride between subsequent hills.

In order to activate the grid in `PLUMED`, the directive `GRID` must be specified for every collective variable `CV`. The keyword `MIN` and `MAX` are used to fix the CV interval, `NBIN` the number

of bins and the flag PBC if the CV is periodic.

In working with grids, we use the following conventions:

- the actual number of bins created is NBIN + 1. For example, if MIN is 0, MAX is 5 and NBIN is 5, the grid is made by 6 bins: [0,1), [1,2), [2,3), [3,4), [4,5), [5,6).
- a point is aligned to the left. 0.999 belongs to [0,1) and 1.001 to [1,2).

Special labels can be used in the definition of the interval with MIN and MAX, such as -pi, +pi, +2pi, -2pi, pi, 2pi. These labels may be particularly useful with the CVs ANGLE or TORSION.

Example.
In this example we run metadynamics using a dihedral angle as CV. The bias potential is put on a grid of 100 bins defined between $-\pi$ and $+\pi$. The grid is periodic.

```
HILLS W_STRIDE 1000 HEIGHT 0.4
WELLTEMPERED SIMTEMP 300 BIASFACTOR 8
TORSION LIST 5 7 9 15 SIGMA 0.35
GRID CV 1 MIN -pi MAX +pi NBIN 100 PBC
ENDMETA
```

As in standard metadynamics, a HILLS file containing the list of Gaussians deposited is produced. This file is needed to restart a metadynamics simulation also when GRID is used. Alternatively, one can dump the grid of the potential on a file using the keyword WRITE_GRID and restart the simulation with READ_GRID (see the manual).

Some rules to keep in mind:

- GRID must be activated (or switched off) on ALL the CVs;
- GRID can be used together with multiple walkers metadynamics, bias-exchange and parallel tempering metadynamics;

- For an accurate calculation of the potential and forces, the bin size must be smaller than half the Gaussian sigma. If a larger size is used, the code will stop.
- If the simulation goes out of the grid, the code will stop. Please increase `MIN` or `MAX` and restart metadynamics.

5.4 Reweighting techniques

From a converged metadynamics run we can calculate directly the canonical probability distribution of the collective variables at a given temperature. On the contrary, the statistics of other degrees of freedom is somehow distorted by the application, during the simulation, of a time-dependent external potential on the CVs. Different possible techniques have been proposed to reconstruct the probability distribution of variables other than the CVs. Here we describe two of them.

5.4.1 Well-tempered metadynamics calculations

In WT metadynamics, the reconstruction of the distribution of variables different from the CVs is particularly simple since for long times the amount of bias added decreases to zero and the system becomes closer and closer to equilibrium.

The algorithm described in Ref. [**?**] consists of three different steps:

1. Accumulate the histogram of the CVs plus the other variables of interest between two updates of the bias potential;
2. When a new Gaussian is added, evolve the histogram fol-

lowing:

$$P(\boldsymbol{R}, t+\Delta t) = e^{-\beta(\dot{V}(\boldsymbol{S}(\boldsymbol{R}),t)-\langle\dot{V}(\boldsymbol{S},t)\rangle)\Delta t}\, P(\boldsymbol{R}, t), \quad (5.9)$$

where $P(\boldsymbol{R}, t)$ is the biased probability distribution, $\dot{V}(\boldsymbol{S}(\boldsymbol{R}), t)$ the time derivative of the bias potential and the average in the exponent is calculated in the biased ensemble;

3. At the end of the simulation, the unbiased distribution $P_B(\boldsymbol{R})$ can be recovered from the histogram collected by using a standard umbrella sampling reweighting:

$$P_B(\boldsymbol{R}) \propto e^{\beta V(\boldsymbol{S}(\boldsymbol{R}),t)} \cdot P(\boldsymbol{R}, t). \quad (5.10)$$

Starting from `PLUMED` version 1.3, we will provide a code to perform this (and other) kind of reweighting procedure. In this tutorial we are presenting a beta version of `reweight`. Here is an example of typical usage.

Example.
We performed a metadynamics run with 2 CVs and we are interested in reconstructing the distribution of a third variable.

```
reweight -colvar COLVAR -hills HILLS -ncv 2 -nvar 3 -stride 1 -fes 3 -temp 300
-welltemp

 -hills      HILLS filename
 -colvar     COLVAR filename
 -out        FES filename
 -ncv        number of variables in HILLS
 -nvar       number of variables in COLVAR
 -stride     ratio between COLVAR and HILLS stride
 -fes        ID of the variables for FES in output
 -temp       temperature in Kelvin
 -ngrid      histogram grid dimension
 -nreject    discard initial steps
 -timeout    stride for FES printout
 -pi         ID of the variables with [−π;π] periodicity
 -welltemp   control for well-tempered metadynamics
 -kjoule     energy in kjoule/mol
```

The code needs two files with the same format of the `PLUMED HILLS` and `COLVAR` files. In the latter, the metadynamics CVs should appear in the first d column followed by the variables whose distribution one wants to reweight. The ratio between the stride in `COLVAR` and `HILLS` must be constant and greater than 1. The more data you have for the histogram, the better.

Some important things to keep in mind:

- For the choice of the bin size, please follow the suggestions described in section 3.5;
- Eq. 5.9 is exact. However, at the beginning of the simulation the average of $\dot{V}(\boldsymbol{S}(\boldsymbol{R}), t)$ can be calculated only approximately. Luckily, a possible initial error is recovered for long times. Alternatively, one could discard the first part of the trajectory using the `-nreject` option. Please always check that your results are robust to a discard of initial parts of the trajectory;
- As for the calculation of the FES with `sum_hills`, remember to control the convergence by plotting the reconstructed distribution at different times by using `-timeout`;

5.4.2 Weighted-histogram analysis of bias-exchange simulations

As explained in Section 4.4, a bias-exchange metadynamics simulation consists in a number N of replicas of the system, each one reconstructing a one-dimensional free-energy profile along a different CV. Instead of having only N one-dimensional projections, it is much more insightful to know the full N-dimensional free-energy landscape of the system, which may resolve all the

relevant minima and transition states. This can be achieved by combining the data from all the bias-exchange replicas into a suitable weighted-histogram technique [**?**].

The basic idea is the following: imagine to divide the N-dimensional CV-space into a grid of small N-dimensional bins (Fig. 5.5). The probability of the states (= bins) visited along the trajectory is not given by the Boltzmann equilibrium distribution because the simulation is affected by the bias potential. Similarly to umbrella sampling, after filling time the equilibrium probability of state α can be estimated as:

$$p_\alpha^i \approx \sum_{t \in \Omega_\alpha^i} e^{\beta(V^i(s_t^i) - f^i)} \tag{5.11}$$

where i is the replica index, Ω_α^i is the set of configurations of replica i belonging to state α, s_t^i is the trajectory in CV-space, V^i is the bias potential (time-averaged after filling time), and f^i is a shift constant. Each replica which visited state α gives an estimate p_α^i, and one can make a weighted average of them to obtain the best overall probability p_α:

$$p_\alpha = C \sum_i \pi_\alpha^i p_\alpha^i \tag{5.12}$$

where C is a normalization constant and the weights π_α^i are computed by minimizing the statistical error:

$$\sigma^2(p_\alpha^i) = g \sum_{t \in \Omega_\alpha^i} e^{2\beta(V^i(s_t^i) - f^i)} \quad , \qquad \sigma^2(p_\alpha) = C^2 \sum_i (\pi_\alpha^i)^2 \sigma^2(p_\alpha^i) \tag{5.13}$$

$$\pi_\alpha^i = \frac{e^{\beta(f^i - \bar{V}_\alpha^i)}}{\sum_j e^{\beta(f^j - \bar{V}_\alpha^j)}} \quad , \qquad e^{\beta \bar{V}_\alpha^i} = \frac{\sum_{t \in \Omega_\alpha^i} e^{2\beta V^i(s_t^i)}}{\sum_{t' \in \Omega_\alpha^i} e^{\beta V^i(s_{t'}^i)}} \tag{5.14}$$

where g is the number of trajectory frames which are time-correlated and the shift constants f^i are determined self-consistently

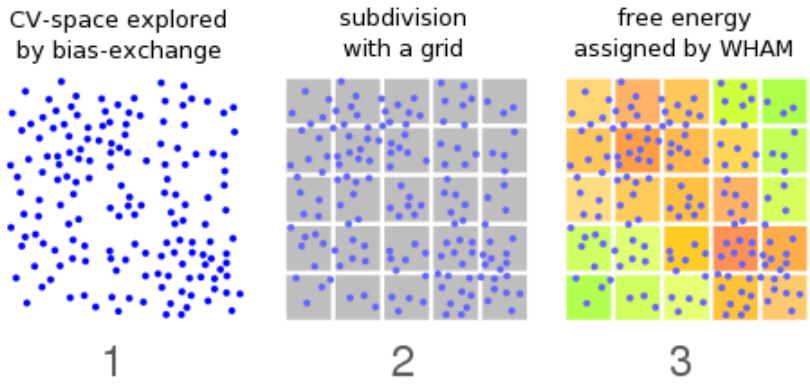


Figure 5.5: The three steps of the bias-exchange + weighted histogram analysis method.

(see Ref. [**?**] for details). Finally, the free energy is given by the usual formula $F_\alpha = -k_B T \log p_\alpha$. In this way, if the grid of bins is fine-grained, the (discretized) N-dimensional free-energy landscape is reconstructed. A scheme of the whole procedure is reported in Fig. 5.5.

As an example we consider the trajectories and Plumed output files generated from the bias-exchange simulation on Ala_3 in Section 4.4. The simulation consisted of four replicas, therefore four atomic trajectories, and each one reconstructed a one-dimensional free-energy profile along a different backbone dihedral ψ_1, ϕ_1, ψ_2, or ϕ_2. To reconstruct the fully-detailed four-dimensional free-energy landscape we employ the VMD plugin `bemeta_analyzer.tcl`, developed by Xevi Biarnes and Alessandro Laio. The plugin reads the files `traj0-3.xtc`, `COLVAR0-3`, and `HILLS0-3`. There must be a one-to-one correspondence between each trajectory file and CVs file, e.g., each frame in `traj0.xtc` must correspond to a line in `COLVAR0` and viceversa. The plugin needs also an input file `cluster.in`:

Example.
file cluster.in for bemeta_analyzer.tcl:

```
KT 2.4943
HILLS_FILE hills0
HILLS_FILE hills1
HILLS_FILE hills2
HILLS_FILE hills3
GRO_FILE start.gro
COLVAR_FILE colvar0 traj0.xtc
COLVAR_FILE colvar1 traj1.xtc
COLVAR_FILE colvar2 traj2.xtc
COLVAR_FILE colvar3 traj3.xtc
TRAJ_SKIP 1
NCV 4
CVGRID 1 -3.14159 3.14159 10 PERIODIC
CVGRID 2 -3.14159 3.14159 10 PERIODIC
CVGRID 3 -3.14159 3.14159 10 PERIODIC
CVGRID 4 -3.14159 3.14159 10 PERIODIC
ACTIVE 4 1 2 3 4
T_CLUSTER 500.
T_FILL 500.
N_MIN 1
DELTA 4
G_CORR 1
REPR "all" "Licorice" "Name" "Opaque"
```

In this example $k_BT = 2.5$ kJ/mol, an initial structure is given (`start.gro`), no frames are skipped (`TRAJ_SKIP 1`), the total number of CVs is 4, for each CV a grid is defined by the minimum, maximum, and number of subdivisions (periodicity, due to the nature of the dihedral CVs, is asked for), the indexes of the active (biased) CVs in the `COLVAR` files is given, the clustering and filling time are 500 ps and a preferred VMD representation is selected. Further parameters are `N_MIN` (minimum number of visits of each bin to be considered reliable), `DELTA` (maximum allowed deviation between estimates of the free-energy profiles in the first and second half of the trajectory), and `G_CORR` (number of time-correlated frames in the trajectory).

Finally, the plugin can be run from the TK console of VMD (open it from the "VMD Main" window, clicking on Extensions, Tk console):

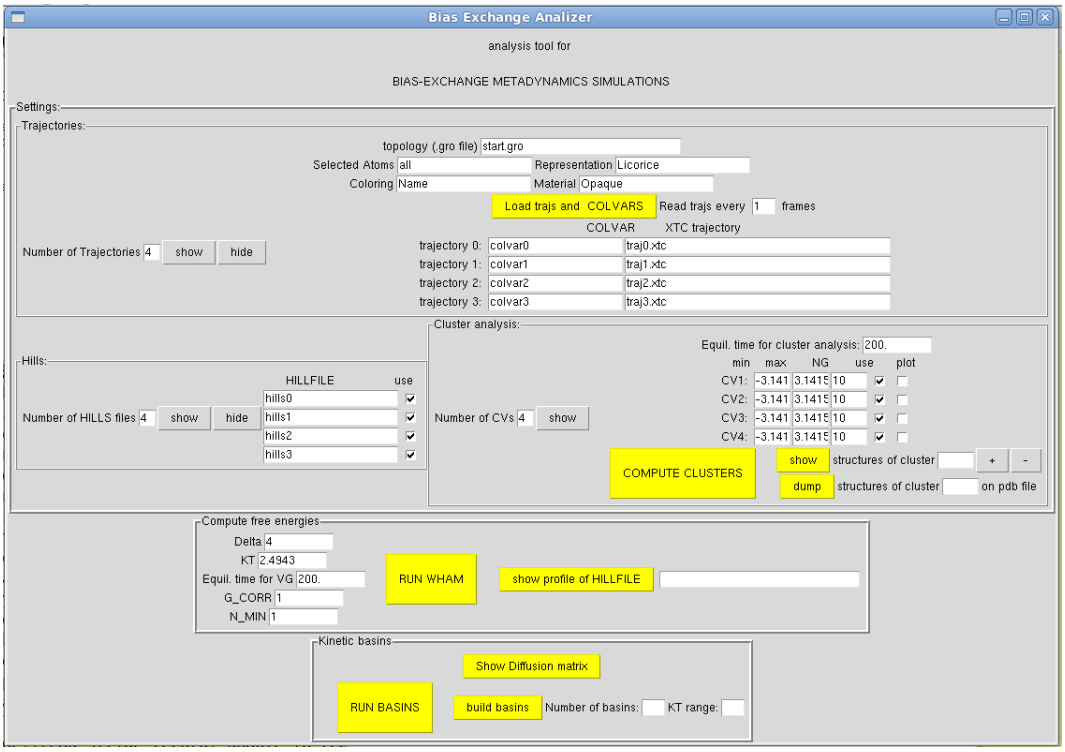


Figure 5.6: Graphical interface of the VMD plugin `bemeta_analyzer.tcl`.

Example.

```
source "bemeta_analyzer.tcl"
::bemeta_gui
```

The input file `cluster.in` is automatically read, and a graphical interface opens (see Fig. 5.6).

The following two commands load the data into VMD (trajectories and CVs). The fourth command subdivides the CV-space into a grid of bins and assigns each frame of the trajectory to a bin (or cluster). The last command calls the program `wham_on clusters_newalign.x` which must be located in the same directory as all the input files. This program estimates the free-

energy of each bin (or cluster) as explained above, writing it in the file CLUSTERS.FES:

Example.
the file CLUSTERS.FES: the columns are 1) the cluster index, 2) the population, 3,4,5,6) the 4 CVs defining the centre, and 7) the free energy

```
1    10    -2.61799 -2.61799 0.52359 -2.61799      10.173
2    42    1.57079 -1.5707 -0.52359 -1.57079      4.305
3    35    -0.52359 -2.61799 -0.52359 -1.57079      5.224
4    23    -0.52359 -2.61799 0.52359 -2.61799      7.183
.....
```

(note that some states may have a free energy of 1000: this means that due to poor statistics the free energy could not be assigned, and probably it is very high). This file contains therefore a discrete representation of the four-dimensional free-energy landscape on a $10 \times 10 \times 10 \times 10$ grid. One can partially visualize the landscape, limited e.g. to the two ψ dihedrals, with gnuplot:

Example.

```
set zrange [0:50]
set xlabel "psi1"
set ylabel "psi2"
set zlabel "F (kJ/mol)"
splot "CLUSTERS.FES" u 3:5:7
```

or with other combinations (4:5:7, 5:6:7, 4:6:7, etc.). The free-energy landscape can be compared with the one obtained from a long (3 μs) equilibrium MD simulation (see Section 4.4 and Fig. 4.5), in file `EQUIL_CLUSTERS.FES`. The script `compare_clusters.sh` performs the comparison. Note that the equilibrium MD has a poor sampling of the barrier regions, while metadynamics has a good sampling, therefeore the agreement between `EQUIL_CLUSTERS.FES` and `CLUSTERS.FES` deteriorates at high free energy.

The plugin `bemeta_analyzer.tcl` allows further to perform a kinetic clustering of the hundreds of states in file `CLUSTERS.FES`, by constructing an approximate kinetic transition matrix and analyzing its spectrum. This procedure locates the free-energy basins (i.e. significant local minima, each one including many states). The detailed explanation of this procedure is beyond the scope of this tutorial, and the reader is referred to specific documentation. Only a rapid introduction is given here. In short, the program `kinetic_basins.x` must be in the same directory, and the commands for `bemeta_analyzer.tcl` are the following:

Example.

```
run_basins
build_basins
show_basins
```

At this point VMD visualizes the clusterized structure of the CV-space, with basins as sets of small spheres of the same color, and basin attractors (the centers of local free-energy minima) as bigger spheres. By clicking on the key "1" (picking mode) inside the VMD display, it becomes possible to select a state from the grid by left-clicking with the mouse. The corresponding atomic configurations of Ala_3 then appear. To return to the CV-space representation it is sufficient to click on "D" in the VMD Main window for the label "CLUSTERS.FES.BASINS".

Chapter 6

Inside PLUMED

6.1 How to plug PLUMED in your MD code

In this part of the tutorial we will learn how to plug `PLUMED` in your MD code.

The usual manner to patch `PLUMED` into a supported MD code is to use the tools in the `patches/` directory of `PLUMED`. These scripts are slightly modifying the host MD code by:

- changing the original Makefile so as to compile also plumed source
- linking the plumed source code in the proper directory
- adding proper calls to `PLUMED` routines

The first two issues are very code dependent, and should be done in an ad-hoc manner. We will focus here on the third issue. To provide a working example, we will patch `PLUMED` on a simple Lennard-Jones code written in basic `FORTRAN`. So, as a first step get the `simplemd.tgz` code. It contains a `src` directory with source code and a `xyz` directory with input files. Compile it and run it as follows

Example.
cd src
./compile
cd ../xyz
../src/simplmd.x < in

The MD code needs to call `PLUMED` routine for initialization (once at the beginning of the simulation) and, at each step, to calculate the forces coming from the bias. The two routines are `mtd_data_init` and `meta_force_calculation`. Have a look to the `common_files/metadyn.c` file. You will find several versions of these routines, depending on which host code is used. Instead of adding new routines, we will just adapt the one used for AMBER. The reason is that AMBER is also written in `FORTRAN`, and its interface is the simplest one.

We copy the plumed source in the `src` directory

Example.
cp PLUMED-ROOT/common_files/*.c `src/` cp PLUMED-ROOT/common_files/metadyn.h `src/`

We change the compile script in such a way to compile also plumed source. Notice that we have to tell to plumed the name of the host MD code. For this example, since we are using the AMBER interface, we add a `-DAMBER` compilation flag.

Example.
Compilation script:
gfortran -fdefault-real-8 -O2 -c *.f90
gcc -DPLUMED_AMBER -O2 -c *.c
gfortran *.o -o simplemd.x -lm

At this point the code should compile without errors. Now we have to set the proper calls to `mtd_data_init` and `meta_force_calculation`.

The first one should be called just after initialization. A good point for this code is after velocity randomization

Example.
```
! velocities are randomized according to temperature
call randomize_velocities(natoms,temperature,masses,velocities,idum)

CALL init_metadyn(natoms,tstep,masses,masses,1,1.0D0,"plumed.dat"//char(0))
```

Notice that the interface is a bit cluttered because it is designed so as to be used from many MD codes (QuantumEspresso, AMBER and DLPOLY). The first argument is the number of atoms, followed by the time-step, the array with the masses. Then we should have the charges (we just set them equal to the masses since there are no charges in LJ codes). The next argument indicates the type of pbc, followed by energy units, and by the name of the plumed input (we leave it hardcoded to plumed.dat).

The call to plumed forces should be put just after the call to LJ forces:

Example.
```
call compute_forces(natoms,listsize,positions,cell,forcecutoff,point,list,forces,engconf)
call meta_force_calculation(cell9,istep,positions,0,0,forces,0,0,engconf)
```

Notice here the two zeros following positions and forces. The reason is that the interface is designed so as to be used also with DLPOLY, where three different arrays are passed (x, y and z components).

6.2 How to add a new CV

In this part of the tutorial we will learn how to add a new CV to `PLUMED`. Inside the code, every CV is identified by a unique name such as `DISTANCE`, `ANGLE` or `TORSION` and by a numeric ID (see the manual for a list). In the following we will call the new CV using the keyword `NEWCV` and use the ID `0` when needed.

Adding a new CV to PLUMED consists of three main steps:

1. create a new file called restraint_newCV.c. This file will contain all the routines needed to parse the PLUMED input file and to calculate the value of NEWCV and its derivatives with respect to the coordinates of the system;
2. modify the PLUMED source code in a few places;
3. reapply the patch and recompile your MD code.

6.2.1 Creating restraint_newCV.c

Let's start by including the file that contains all the definition of routines and variables that are probably needed, namely metadyn.h.

The parser

The first important routine that must be defined in restraint_newCV.c is the parser, usually named read_newCV. This routine will be called by PLUMED when the main parser finds the keyword NEWCV in the input file.

As far as parsing is concerned, there are two main types of CVs. The majority of them needs to read only one line from the input file. An example is ANGLE:

```
ANGLE LIST 11 13 17 SIGMA 0.35
```

It should be noted that this CV may be defined also in terms of group of atoms, whose definition is given in another place of the PLUMED input file:

```
ANGLE LIST 11 13 <CA> SIGMA 0.35
```

```
CA-> 3 7 10 CA<-
```

The parser for this kind of CVs is declared as:

```
int read_angle (char **word,int count,t_plumed_input *input,
FILE *fplog);
```

where:

- `**word` contains the entire line of the input file that starts with `ANGLE`;
- `count` is a CV counter;
- `*input` is a complex structure which contains all the `PLUMED` input file;
- `*fplog` points to the `PLUMED` log file.

A few CVs need to read from the input file data with a syntax different from groups of atoms. To do so, we need to pass an additional information to the parser. An example is `RMSDTOR`. This CV needs to scan the input file for a list of atoms that defines a dihedral angle and to read a reference value:

```
RMSDTOR NDIH 2 SIGMA 0.35
13 15 17 1 0.5
15 17 1 3 2.0
```

The parser for this kind of CVs is declared as:

```
int read_rmsdtor (char **word,int count,t_plumed_input
*input,int *iline,FILE *fplog);
```

where `*iline` indicates the line of the input file that is currently parsed. This variable is modified when scanning the rest of the input file to get the additional data required by the CV.

Here we will focus on the first type of CVs. In the example box below we show the essential lines of the `ANGLE` parsing routine.

Example.
Structure of a typical parsing routine.

```
int PREFIX read_angle(char **word, int count, t_plumed_input *input, FILE *fplog){

int j, iw;
double delta = 0.0;


iw = seek_word(word,"LIST");
if(iw>=0) {
j=plumed_get_group(word[iw+1],&colvar.cvatoms[count],colvar.natoms[count],input,fplog);
colvar.natoms[count]+=j;
colvar.list[count][0]=j;
j=plumed_get_group(word[iw+2],&colvar.cvatoms[count],colvar.natoms[count],input,fplog);
colvar.natoms[count]+=j;
colvar.list[count][1]=j;
j=plumed_get_group(word[iw+3],&colvar.cvatoms[count],colvar.natoms[count],input,fplog);
colvar.natoms[count]+=j;
colvar.list[count][2]=j;
} else { fprintf(fplog,"|- NEEDED LIST KEYWORD FOR ANGLE");}


iw=seek_word(word,"SIGMA");
if(iw>=0){ sscanf(word[iw+1],"%lf", &delta);
colvar.delta_r[count] = (real) delta; }


colvar.type_s[count] = 4;


snew(colvar.myder[count], colvar.natoms[count]);


return colvar.natoms[count];
}
```

The example above shows some of the fundamental variables and routines that are used in the parser:

- `seek_word` looks for specific words in the parsed line;

- `plumed_get_group` is used to read a group of atoms;
- `colvar.natoms` is the total number of atoms used to define this CV;
- `colvar.cvatoms` is the list of atoms used to define this CV;
- `colvar.delta_r` is the Gaussian sigma;
- `colvar.type_s` is the unique ID of the CV;
- `colvar.myder` will contain the derivatives of this CV with respect to the coordinates of the atoms and must be initialized in the parser routine.

CV definition and derivatives

The second important routine that must be included in `restraint_newCV.c` contains the mathematical definition of the CV and its derivatives and it is usually called `newCV_restraint`. The declaration of this routine is common among all the CVs. Below is the example of the `ANGLE` restraint.

Example.
```
void angle_restraint (int i_c, struct mtd_data_s *mtd_data);
```

The variables that appear in the declaration of `newCV_restraint` are:

- `i_c`, a CV counter;
- `*mtd_data`, the fundamental structure of `PLUMED` that contains data passed from the main MD code. These include positions, masses and charges of all the atoms, information about the time steps, temperature, unit of measure, periodic boundary conditions and others.

The routine `newCV_restraint` uses the information in `mtd_data` to calculate at every MD step the value of `NEWCV` and to store it in the `colvar.ss0[i_c]` variable. The derivatives with respect to all the atoms involved must be also calculated and stored in the `colvar.myder` array.

6.2.2 Modifying the `PLUMED` source code

Once the routines for parsing the new CV and calculating its value and derivatives have been implemented, we need to modify the source code to instruct `PLUMED` to call them at the right moment. To do so, we have to act in a few selected spots:

1. we have to add the declaration of the new routines `read_newCV` and `newCV_restraint` in the `metadyn.h` file;

2. in the `read_restraint.c` file, we have to tell `PLUMED` to call `read_newCV` when the keyword `NEWCV` is found in the input file. Here are the relevant lines for the `ANGLE` CV:

   ```
   } else if(!strcmp(word[0],"ANGLE")){
   read_angle(word, count, &input, mtd_data->fplog);
   count++;
   }
   ```

3. in the `restraint.c` file, we have to tell `PLUMED` to call `newCV_restraint` when appropriate. In the usual example of `ANGLE` CV, whose ID is 4:

   ```
   switch(colvar.type_s[i_c]){
   ...
   case 4:  angle_restraint(i_c, mtd_data); break;
   ...
   }
   ```

6.2.3 Final steps

Once the modifications to the `PLUMED` source code are done, we need to reapply the patch and recompile the MD host code. After that, we are ready to use our new CV.

Before starting a real calculation, it is better to check that we have done everything correctly. The most frequent source of error is the implementation of the analytical derivatives of the CV with respect to the coordinates of the system. To detect possible errors, we can use the keyword `DEBUG`. Before doing so, we have to manually add `NEWCV` to the list of variables for which debugging is active. This list is defined in the file `testderivatives.c` together with the routines for comparing the analytical derivatives of a CV to the value calculated with finite differences. These routines call `newCV_restraint` several times to calculate the CV value for different atoms positions. Have a look at `testderivatives.c` and add all the calls to `newCV_restraint` that are needed, following the usual example of `angle_restraint`!

Chapter 7

Real life applications

7.1 The Stone-Wales transformation in a carbon nanotube

As an example of an activated process in a condensed matter system we consider the Stone-Wales transformation in a carbon nanotube. This transformation can be seen as the rotation of a C-C dimer by 90°, which leads to the conversion of four hexagons into two pentagons and two heptagons. During the process, which has been observed e.g. in nanotubes subject to strain, overall two covalent bonds are broken and two are formed, resulting in a large energy barrier of several eV. Here we consider a 480-atoms (10,0) carbon nanotube periodically repeated, and we adopt the AIREBO interatomic potential as implemented in lammps. One can imagine different suitable reaction coordinates for the Stone-Wales transformation. Here we use the number of covalent three-atoms bridges among different groups of atoms, as detailed in Fig. 7.1. Rotation of the C-C dimer leads to breaking of two "vertical" bridges and to forming two "horizontal" bridges. A suitable Plumed CV is `WATERBRIDGE` (the name water has only historical sense since the CV was origi-

nally implemented to count the H-bonded bridges formed by water molecules between two protein surfaces).

Example.
input file for the Stone-Wales transformation in a carbon nanotube with lammps

```
# 0.4 eV (lammps "metal" units) every 50 fs (timestep = 1 fs)
HILLS HEIGHT 0.4 W_STRIDE 50
PRINT W_STRIDE 10

WATERBRIDGE LIST <hor1> <hor2> <dimer> R_0 1.9 NN 8 MM 20 SIGMA 0.2
hor1->
229 259
hor1<-
hor2->
226 256
hor2<-
dimer->
238 248
dimer<-

WATERBRIDGE LIST <ver1> <ver2> <dimer> R_0 1.9 NN 8 MM 20 SIGMA 0.2
ver1->
256 259
ver1<-
ver2->
226 229
ver2<-

ENDMETA
```

Within 10 ps of simulation, the Stone-Wales transformation occurs reversibly multiple times, and a FES can be reconstructed with `sum_hills`:

Example.
generate FES file fes.dat:

```
sum_hills.x -file HILLS -out fes.dat -ndim 2 -ndw 1 2
```

plot FES with gnuplot:

```
set pm3d
set contour base
splot 'fes.dat' with pm3d
```

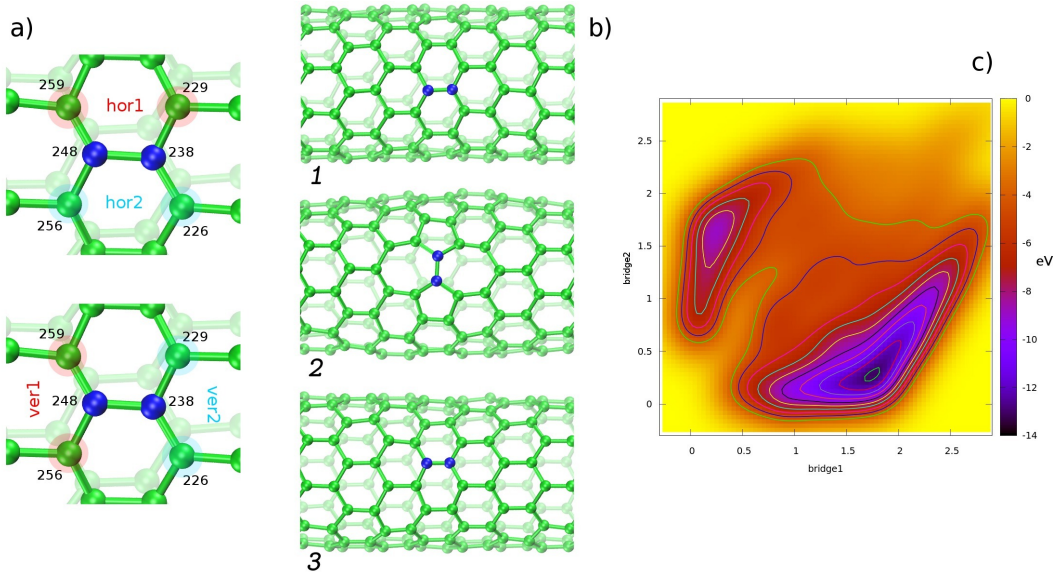


Figure 7.1: a) Definition of the groups of atoms in the input file. b) Reversible Stone-Wales transformation during the simulation with the CV `WATERBRIDGE`. c) reconstructed FES after deposition of 400 hills.

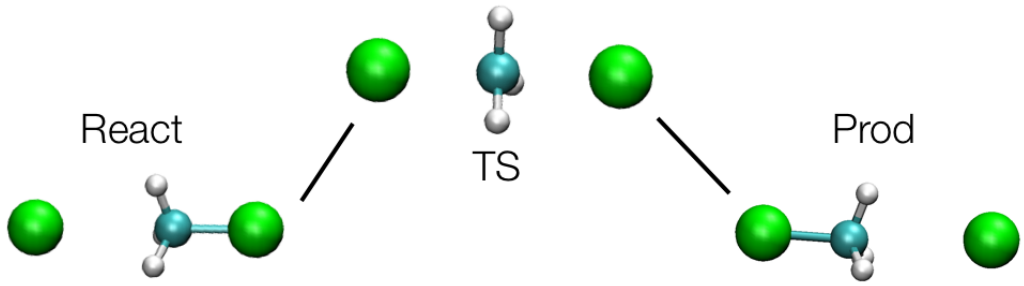


Figure 7.2: A sketch of SN2 reaction

7.2 A SN2 reaction in vacuum with quantum espresso

This section will show a simple chemical reaction example done with QUANTUM ESPRESSO code. This code is meant here to do Born-Oppenheimer dynamics but `PLUMED` is also implemented to work in the Car-Parrinello module of the suite. The goal of the exercise is to calculate the free energy for the reaction depicted in 7.2. This reaction has been studied before [**?**] and here we will try to sketch out the barriers but the aim is to take a short amount of time. In the PW code input (file `bo.in`) `PLUMED` is activated with the flag `use_plumed=.true.` in the control section then the code expects a `plumed.dat` file.

```
&control
    title = 'ch3cl',
    calculation='md'
    restart_mode='from_scratch',
    pseudo_dir = '.',
    outdir='.',
    dt=20,
    nstep=2000,
```

```
    disk_io='low',
    prefix = 'md',
    use_plumed = .true.,
/
```

Pay attention: in this code distances are in Bohr (1 Bohr = 0.529177249 Å) and the energies in Rydberg (1 Ry= 313.755 kcal/mol).

Now you have to choose what to do at this stage. You have first a bunch of techniques available.

- Metadynamics? Can be an option but you have a fixed small amount of time available here. And you don't know how much it will take to make a single event but you also might have multiple recrossing.
- Thermodynamic integration (In the steered-md flavour). Efficient and guaranteed to make the reaction but no average on multiple recrossing events. And what about the dissipative work? Here you don't have many orthogonal degrees of freedom. Probably could work.
- Umbrella Sampling. Similar problems to TI, with the exception that the work are calculated at equilibrium. No problem with dissipative work.

And the second thing that you should decide is the CV to be used.

- Distances?
- Path ? (probably is overshooting?
- Does the angle matters?

- Coordination number?
- Anything else?

For Metadynamics a possible input can be

Example.
```
# switching on metadynamics and Gaussian parameters
HILLS HEIGHT 0.001 W_STRIDE 2
# instruction for CVs printout
PRINT W_STRIDE 1
# the distance between C-Cl' and C-Cl
DISTANCE LIST 1 3 SIGMA 0.3
DISTANCE LIST 2 3 SIGMA 0.3
#WALLS: prevent to depart the two mols
UWALL CV 1 LIMIT 7.0 KAPPA 100.0
LWALL CV 1 LIMIT 2.5 KAPPA 100.0
UWALL CV 2 LIMIT 7.0 KAPPA 100.0
LWALL CV 2 LIMIT 2.5 KAPPA 100.0
# end of the input
ENDMETA
```

Here two independent variable on a range are used. A possible other input (case of steered md):

Example.
```
# instruction for CVs printout
PRINT W_STRIDE 1
# the distance between C-Cl' and C-Cl
DISTANCE LIST 1 3 DIFFDIST 2 3
STEER CV 1 TO 2.35 VEL 2.50 KAPPA 0.5
# end of the input
ENDMETA
```

Example.
```
# instruction for CVs printout
PRINT W_STRIDE 1
# the distance between C-Cl' and C-Cl
DISTANCE LIST 1 3 DIFFDIST 2 3
DISTANCE LIST 1 2
STEER CV 1 TO 2.35 VEL 2.50 KAPPA 0.5
UWALL CV 2 LIMIT 8.7 KAPPA 0.5
# end of the input
ENDMETA
```

In this other option you use one single cv and you keep a wall on the other. Which are the advantages/disadvantages? One variable is faster but you don't know where and how much the wall is affecting your simulation. Is there any way to calculate and address this issue? You can run the job like this

```
source  /usr/local/Modules/3.2.6/init/bash
module load intel-cc/10.1.015
module load intel-fc/10.1.015
module load intel-mkl/10.0.1.014
module load openmpi/1.2.6_intel-10.1.015
PATH_TO_ESPRESSO=/my/path/to/espresso
mpirun -np 4 $PATH_TO_ESPRESSO/espresso-4.2.1/PW/pw.x < bo.in > bo.o
```

Choose one of those two technique and run the calculation. Use `sum_hills` or the awk script to calculate the work (in the steering example) to calculate the energy profile.

7.3 Folding of the GB1 C-terminal β–hairpin

In this part of the tutorial, we will use `PLUMED` in combination with GROMACS to study the folding process of a small peptide, the C-terminal domain of protein GB1 [**?**, **?**]. This 16-residue peptide is a prototypical example of β–hairpin structure (see Fig. 7.3) which has attracted the attention of both the experimental and theoretical community in the recent years.

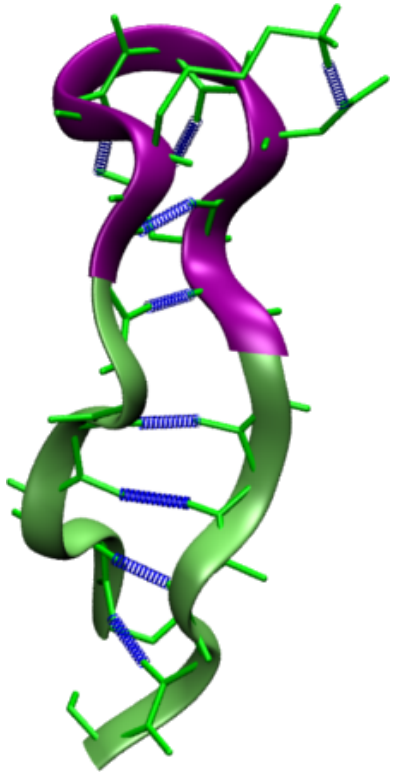


Figure 7.3: Structure of the C-terminal domain of protein GB1. In violet, the residues that define the turn region.

A variety of computational methods and models for the protein and the water force fields, together with a number of dif-

ferent descriptors have been used in those works aimed at characterizing the folding landscape of this peptide [**?**, **?**, **?**, **?**, **?**, **?**, **?**, **?**]. In particular, among the CVs used are:

- Number of hydrogen-bonds;
- Backbone radius of gyration;
- Hydrophobic core radius of gyration;
- RMSD;
- ASA;
- Principal components;
- Path Collective Variables;
- Contact maps.

In this exercise, we will experiment with different collective variables and free-energy methods. We will simulate the hairpin in vacuum using the OPLS-AA [**?**] force field. More precisely, in this exercise you should combine many of the techniques learnt in this tutorial and:

1. Choose a limited number of CVs;
2. Choose a free-energy method;
3. Restrict the turn region to simplify the problem;
4. Monitor the RMSD of the protein from the native state to see if the peptide is correctly refolding;
5. Monitor the convergence of your calculation;

6. Calculate and analize the FES.

Here is an example of `PLUMED` input file with some of the CVs that can be used to address this problem.

Example.

```
PRINT W_STRIDE 500
# choose your free-energy method
HILLS W_STRIDE 500 HEIGHT 0.2

# if you choose metadynamics, try to use GRID
# GRID CV 1 MIN 0 MAX 10 NBIN 100

# CHOOSE YOUR CVs or create your own
# number of native CA contacts
CMAP INDEX CMAP_native.CA

# total number of CA contacts
CMAP INDEX CMAP_all.CA

# RMSD**2 from native structure calculated on CA
TARGETED TYPE RMSD FRAMESET hpin_native_CA.pdb

# RMSD**2 from native structure calculated on backbone
TARGETED TYPE RMSD FRAMESET hpin_native_BACKBONE.pdb

# number of hbonds
# TYPE means
# 0 ALL
# 1 BETA ODD
# 2 ALPHA
# 3 BETA EVEN
# 4 NATIVE (like PAIR coordination)
# 5 BETA ALL
HBONDS LIST <H> <O> TYPE 1
H->
15 30 54 68 89 101 113 123 137 159 173 193 207 223
H<-
O->
28 52 66 87 99 111 121 135 157 171 191 205 221 235
O<-

# radius of gyration of the hydrophobic core
RGYR LIST <HC>
HC->
29 31 33 36 37 39 40 42 43 45 47 49 51 52
67 69 71 74 75 77 79 81 83 84 86 87 172 174 176 179 180
182 184 186 188 190 191 206 208 210 212 216 220 221
HC<-

# CA radius of gyration
RGYR LIST <CA>
CA->
9 16 31 55 69 90 102 114
124 138 160 174 194 208 224 238
CA<-

# you may want to keep the turn formed
TARGETED TYPE RMSD FRAMESET hpin_native_TURN.pdb
UWALL CV 2 LIMIT 0.01 KAPPA 5000000.0 EXP 2.0

ENDMETA
```