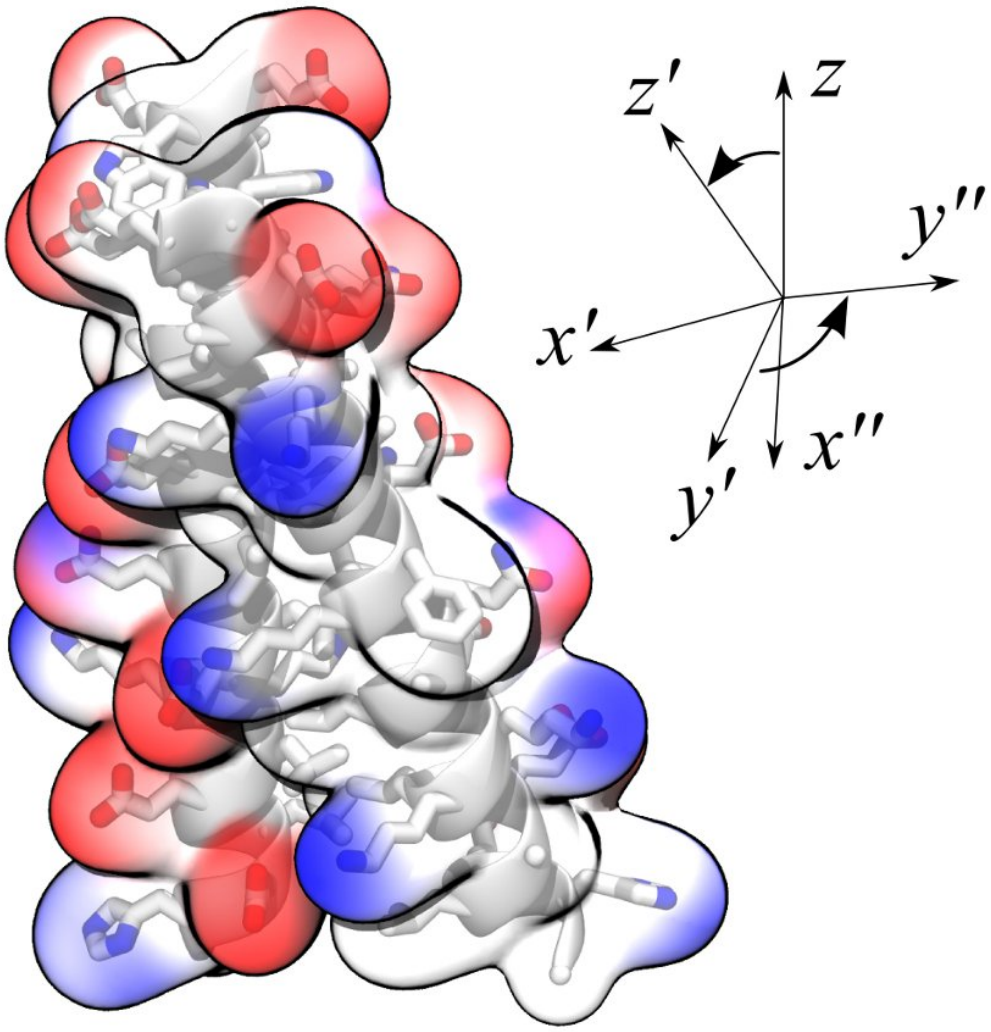
COLLECTIVE VARIABLES MODULE
Reference manual for VMD

Code version: 2020-02-27



Alejandro Bernardin, Haochuan Chen, Jeffrey R. Comer, Giacomo Fiorin, Haohao Fu, Jérôme Hénin, Axel Kohlmeyer, Fabrizio Marinelli, Joshua V. Vermaas, Andrew D. White

Contents

1 Overview

In molecular dynamics simulations, it is often useful to reduce the large number of degrees of freedom of a physical system into few parameters whose statistical distributions can be analyzed individually, or used to define biasing potentials to alter the dynamics of the system in a controlled manner. These have been called 'order parameters', 'collective variables', '(surrogate) reaction coordinates', and many other terms.

Here we use primarily the term 'collective variable', often shortened to *colvar*, to indicate any differentiable function of atomic Cartesian coordinates, x_i, with i between 1 and N, the total number of atoms:

$$\xi(t) = \xi(X(t)) = \xi\left(x_i(t), x_j(t), x_k(t), \ldots\right) , \quad 1 \leq i, j, k \ldots \leq N \tag{1}$$

This manual documents the collective variables module (**Colvars**), a software that provides an implementation for the functions $\xi(X)$ with a focus on flexibility, robustness and high performance. The module is designed to perform multiple tasks concurrently during or after a simulation, the most common of which are:

- apply restraints or biasing potentials to multiple variables, tailored on the system by choosing from a wide set of basis functions, without limitations on their number or on the number of atoms involved;
- calculate potentials of mean force (PMFs) along any set of variables, using different enhanced sampling methods, such as Adaptive Biasing Force (ABF), metadynamics, steered MD and umbrella sampling; variants of these methods that make use of an ensemble of replicas are supported as well;
- calculate statistical properties of the variables, such as running averages and standard deviations, correlation functions of pairs of variables, and multidimensional histograms: this can be done either at run-time without the need to save very large trajectory files, or after a simulation has been completed using VMD and the `cv` command.

Note: although restraints and PMF algorithms are primarily used during simulations, they are also available in VMD to test a new input for a simulation, or to evaluate the relative free energy of a new structure based on data from a previous calculation. *Options that only have an effect during a simulation are also included for compatibility purposes.*

Detailed explanations of the design of the Colvars module are provided in reference [1]. Please cite this reference whenever publishing work that makes use of this module.

Using the Colvars Module in VMD Within VMD, the Colvars Module can be accessed in two ways:

- Using the Colvars dashboard, an intuitive, but partial interface to the Colvars module, to easily define and analyze **collective variables, but not biases** (section 3).
- Using the full-featured Tcl scripting interface as documented in section 4.2; see in particular the example in section 4.2.4.

2 Writing a Colvars configuration: a crash course

The Colvars configuration is a plain text file or string that defines collective variables, biases, and general parameters of the Colvars module. It is passed to the module using back-end-specific commands documented in section 4. Writing the configuration fora collective variable in VMD is made much easier using the dashboard and its configuration editor (section 3). However, note that the dashboard does not handle biases: if necessary, they should be managed separately using the scripting interface.

Now let us look at a complete, non-trivial configuration. Suppose that we want to run a steered MD experiment where a small molecule is pulled away from a protein binding site. In Colvars terms, this is done by applying a moving restraint to the distance between the two objects. The configuration will contain two blocks, one defining the distance variable (see section 5 and 5.2.1), and the other the moving harmonic restraint (7.5). Note that in VMD, no biasing forces are applied, but biases may be useful in the context of an analysis script, e.g. to collect histograms or to compute bias energies.

```
colvar {
  name dist
  distance {
    group1 { atomNumbersRange 42-55 }
    group2 {
      psfSegID PR
      atomNameResidueRange CA 15-30
    }
  }
}

harmonic {
  colvars dist
  forceConstant 20.0
  centers 4.0         # initial distance
  targetCenters 15.0  # final distance
  targetNumSteps 500000
}
```

Reading this input in plain English: the variable here named *dist* consists in a distance function between the centers of two groups: the ligand (atoms 42 to 55) and the α-carbon atoms of residues 15 to 30 in the protein (segment name PR). To the "*dist*" variable, we apply a `harmonic` potential of force constant 20 kcal/mol/Å^2, initially centered around a value of 4 Å, which will increase to 15 Å over 500,000 simulation steps.

The atom selection keywords are detailed in section 6.

3 The Colvars dashboard

The Colvars dashboard is a graphical interface for interactive visualization and refinement of collective variables aided by molecular structures and trajectories. It is accessible in VMD's Main Menu under "Extensions/Analysis/Colvars Dashboard". Throughout the interface, keyboard shortcuts for common operations are indicated in square brackets.

3.1 A mini-tutorial

Here are the steps for a quick first tour of the Dashboard:

1. load an MD trajectory into VMD;
2. open the Dashboard;
3. click "New" to create a new collective variable;
4. in the Editor window, click "Apply" to accept the dein the Dashboard window, fault template;
5. in the Dashboard window, click "Show atoms" to display the two atom groups involved in this distance coordinate;
6. click "Timeline plot";
7. click anywhere in the timeline plot to navigate in the trajectory.

Now, clicking "Edit" in the Dashboard window, you can modify the collective variable to reflect interesting geometric properties of the system. The power of the collective variables approach lies in the variety of geometric functions ("components") and their combinations. The editor window provides a number of helpers to make it easy and quick to define the most relevant variables. See section 3.4 for details.

3.2 The Dashboard window

The Dashboard window displays a table listing currently defined variables, and their values for the current frame indicated at the bottom of the window. By default the frame is updated to track VMD's currently displayed frame, but that can be changed by toggling the "Track frame" checkbox, e.g. to animate the trajectory without recomputing expensive variables. Vector-values variables can be expanded to list their scalar elements. This is necessary when individual scalar quantities have to be selected for plotting. Other operations act on variables as a whole and ignore specific selected scalar elements.

Buttons above the table allow for general operations on the state of the Colvars Module. Buttons below the table offer operations on selected variables.

If several molecules are loaded, the dashboard only interacts with the molecule labeled "top" (T in VMD's main window). If the top molecule is changed, the Colvars Module needs to be reset using the Reset button. This will remove all current definitions, so make sure to save the variables to a file beforehand.

If variables are modified, added or deleted interactively or by an external script, hit "Refresh" or press F5 to update the displayed variables and values. Starting the dashboard also enables trajectory animation using the left/right arrow keys within VMD's graphical window. Atomic coordinates can be modified using

VMD's "Mouse/Move" functions, and the Colvars Module can then be updated by pressing F5 directly from the graphical window.

A dropdown list allows for changing the current unit system if no variables are defined. If some variables are already defined, it is recommended to edit the configuration for all of them at once (eg. pressing Ctrl-a, then Ctrl-e), checking that all quantities are expressed in the desired set of units, and adding the `units` keyword to the general parameters, outside of `colvar {}` blocks (4.1).

Another dropdown lets the user change which VMD molecule is associated with the Colvars module. Internally, this requires recording the configuration of currently defined colvars, deleting the current instance of the Colvars module, creating a new one linked to the target molecule, and applying the saved configuration. Beware of incompatible colvar definitions, such as atom groups listing atom IDs that exist in one molecule, but not the other. Auto-updating selections (see below) can be used to adapt the colvar definitions to a different system using VMD selection texts.

3.3 Loading / Saving configuration files

This saves the configuration of all defined collective variables to a file. Neither biases, nor general parameters of the Colvars Module are saved: editing them is beyond the scope of the dashboard. We recommend keeping them in separate configuration files, and reading them separately in biased MD simulations.

3.4 The configuration editor

The configuration editor can be started with the "Edit" or "New" buttons. Using the "Edit" button, the configuration of selected variables is loaded, and those variables will be replaced when applying the new configuration.

The editor window offers links to online documentation, as well as helpers to write correct configuration files.

As a first step, the most useful helper is the collection of template files. Some parameters that must be supplied are indicated by the symbol `@`. Colvar templates can be inserted at the beginning of the configuration, whereas "component" templates define basis functions that belong inside a colvar block. Templates are indented using 4 spaces per level to indicate their position in the nested structure of the configuration: general options, colvars and biases at level 0, bias and colvar parameters like components at level 1, component parameters such as atom groups at level 2, and atom group parameters at level 3.

The next helper buttons allow importing atom selections from VMD, either typing a VMD atom selection text, by copying the selection of an existing graphical representation, or by inserting the list of atoms currently labeled in VMD using the "Pick atom" feature. Atom selections should be inserted within an atom group block, within a component block (such as `distance`). By default atom selections are preceded with a comment line marking them as *auto-updating*. This instructs the Dashboard to update the list of atoms whenever the configuration is applied, that is, when it is edited, when the file is loaded by the Dashboard, or when changing the VMD molecule linked to Colvars. This is useful when working across systems with different atom numberings, but topologies that make the relevant atom groups identifiable using VMD selection texts. Special fields (O, B, and user) may be used, as well as atom positions (e.g. `z > 0`). If the selection text is modified manually, the atom list will be updated when applying the new configuration. This auto-updating behavior can be disabled by removing the special comment line or altering the keywords "auto-updating selection".

Note that atom lists are **not auto-updated**:

1. when changing frame within the same molecule (animating a trajectory);

2. when the configuration is read by the Colvars module outside of VMD (eg. within an MD engine).

The "Insert labeled..." button combined with the selection box allows for inserting components matching VMD's geometry measurements: Bonds (distances), angles, and dihedrals. Hidden labels are not used for inserting components.

3.5 Plotting and visualizing collective variables

Timeline plots show the selected variables as a function of time. A vertical bar indicates the current frame, which can be changed either using VMD's trajectory animation controls, or directly in the plot window by clicking the mouse inside the graph, or using the keyboard left/right arrows. Shift+arrow skips frames for faster animation, and Ctrl+arrow skips more frames. The up/down arrows operate a zoom/unzoom along the time axis. Visible data can be fitted vertically using the h key. All data can be fitted horizontally using the h key.

Pairwise scatterplots are useful to identify correlation between variables. To create a pairwise plot, select exactly two scalar variables (or scalar components of vector variables), and click "Pairwise plot". Frames are represented by circles, and lines connect consecutive frames. The blue dot tracks the current frame. Arrow keys animate the trajectory as in the timeline plot. Clicking a circle jumps to the corresponding frame.

"Show atoms" creates representations of the atoms involved in the definition of the selected colvars. Each atom group is shown in a different color. "Show gradient" is available for scalar variables only. It creates a graphical representation of the atomic gradients of the selected variables, visualizing how the value of the collective variable would vary in response to a change in atomic coordinates. Vectors representing the gradient are rescaled as indicated by the radio buttons *Set max. vector norm* and *Set scaling factor*. *Set max. vector norm* rescales gradients so that the largest vector component of each colvar's gradient is represented by an arrow of the specified length, in Å. *Set scaling factor* rescales gradients by the specified factor, divided by the colvar's `width` parameter (1 by default, see 5.16). This use of `width` makes it easier to compare the gradients of collective variables that are not commensurate. The scaling factor has the unit $\text{Å} * L/(cv/width)$, where L is the current length unit, and cv represents the natural unit of the collective variable. By default $width$ is unity, but $(cv/width)$ may be seen as dimensionless if $width$ is expressed in cv units.

4 Enabling and controlling the Colvars module in VMD

Here, we document the syntax of the commands and parameters used to set up and use the Colvars module in VMD. One of these parameters is the configuration file or the configuration text for the module itself, whose syntax is described in 4.3 and in the following sections.

4.1 Units in the Colvars module

The "internal units" of the Colvars module are the units in which values are expected to be in the configuration file, and in which collective variable values, energies, etc. are expressed in the output and colvars trajectory files. Generally **the Colvars module uses internally the same units as its back-end MD engine, with the exception of VMD**, where different unit sets are supported to allow for easy setup, visualization and analysis of Colvars simulations performed with any simulation engine.

Note that **angles** are expressed in degrees, and derived quantites such as force constants are based on degrees as well. Atomic coordinates read from **XYZ files** (and PDB files where applicable) are expected to be expressed in Ångström, no matter what unit system is in use by the back-end or the Colvars Module.

To avoid errors due to reading configuration files written in a different unit system, it can be specified within the input:

- `units` ⟨Unit system to be used⟩
  Context: global
  Acceptable values: string
  Description: A string defining the units to be used internally by Colvars. Allowed values are: *real* (Å, kcal/mol), *gromacs* (nm, kJ/mol), *metal* (Å, eV), and *electron* (Bohr, Hartree). In VMD, the default system of units for Colvars is VMD's native units: *real* (Å, kcal/mol). However, the `units` keyword will switch to a different unit system than the current one if no colvars were defined before reading the current configuration. If colvars are already defined, `units` will refuse changing the unit system to avoid making the definition of those variables erroneous in the new system of units. If needed this precaution can be overridden using the `cv units` scripting command (8.1).

4.2 Using the cv command to control the Colvars module

At any moment after the first initialization of the Colvars module, several options can be read or modified by the Tcl command `cv`, with the following syntax:
`cv <subcommand> [args ...]`
The `cv` command is used by the Dashboard graphical interface,(3), but can be also used in scripts or interactively from the command-line terminal (for example, in remote terminal sessions) or in the Tk Console. The most frequent uses of the `cv` command are discussed here. For a complete list of all sub-commands of `cv`, see section 8.

4.2.1 Setting up the Colvars module

The first step to using Colvars in VMD is choosing which "molecule" (i.e. which system): because VMD can handle multiple "molecules", the Colvars module needs to remain attached to a specific VMD molecule. For example:
`cv molid top`
will attach the Colvars module onto the molecule currently holding the "top" status (alternatively, you can refer to a molecule by its numeric ID in lieu of `top`). All following invocations of the `cv` command will continue operating on the same molecule, regardless of whether other molecules are loaded, or which one has the "top" status. The `cv molid` command without argument will return the molid currently associated with Colvars.

To define collective variables and biases, configuration can be loaded using either:
`cv configfile colvars-file.in`
to load configuration from a file, or:
`cv config "keyword { ... }"`
to load configuration as a string argument.

The latter version is particularly useful to dynamically define the Colvars configuration.

4.2.2 Using the Colvars version in scripts

The vast majority of the syntax in Colvars is backward-compatible, adding keywords when new features are introduced. However, when using multiple versions simultaneously it may be useful to test within the script whether the version is recent enough to support the desired feature. `cv version` can be used to get the Colvars version number for this use:

```
if { [cv version] >= "2020-02-25" } {
  cv config "(define a recent feature)"
}
```

4.2.3 Loading and saving the Colvars state and other information

After a configuration is fully defined, `cv load` may be used to load a state file from a previous simulation that contains e.g. data from history-dependent biases), to either continue that simulation or analyze its results:
`cv load <oldjob>.colvars.state`
or more simply using the prefix of the state file itself:
`cv load <oldjob>`

`cv save`, analogous to `cv load`, saves all restart information to a state file. This is normally not required during a simulation if `colvarsRestartFrequency` is defined (either directly or indirectly by the VMD restart frequency), but it is necessary in post-processing e.g. with VMD. Because not only a state file (used to continue simulations) but also other data files (used to analyze the trajectory) are written, it is generally clearer to use `cv save` with a prefix rather than a file name:
`cv save <job>`

See 8.1 for a complete list of scripting commands used to manage the Colvars module.

4.2.4 Analyzing a trajectory in VMD

One of the typical uses of Colvars in VMD is computing the values of one or more variables along an existing trajectory. A complete example input for this use case is shown here.

```
# Activate the module on the current VMD molecule
cv molid top
# Load a Colvars config file
cv configfile test.in
set out [open "test.colvars.traj" "w"]
# Write the labels to the file
puts -nonewline ${out} [cv printframelabels]
for { set fr 0 } { ${fr} < [molinfo top get numframes] } { incr fr } {
  # Point Colvars to this trajectory frame
  cv frame ${fr}
  # Recompute variables and biases (required in VMD)
  cv update
  # Print variables and biases to the file
  puts -nonewline ${out} [cv printframe]
}
close ${out}
```

4.2.5 Managing collective variables

After one or more collective variables are defined, they can be accessed via `cv colvar [args ...]`. For example, to recompute the collective variable `xi` the following command can be used:
`cv colvar xi update`
This ordinarily not needed during a simulation run, where all variables are recomputed at every step (along with biasing forces acting on them). However, when analyzing an existing trajectory a call to `update` is generally required.

While in all typical cases all configuration of the variables is done with `cv config` or `cv configfile`, a limited set of changes can be enacted at runtime using `cv colvar <name> modifycvcs [args ...]`. Each argument is a string passed to the function or functions (called colvar components, or CVCs 5.1) used to compute the variable. For example, the a variable `DeltaZ` made of a single `distanceZ` CVC can be made periodic with a period equal to the unit cell dimension along the Z-axis:
`cv colvar DeltaZ modifycvcs "period $Lz"`
where `$Lz` may be obtained for example as:
`set Lz [molinfo top get c]`.
This option is currently limited to changing the values of `componentCoeff` and `componentExp` (e.g. to update the polynomial superposition parameters on the fly), of `period` and `wrapAround`, and of the `forceNoPBC` option for all components that support it.

If the variable is computed using more than one CVC, it is possible to selectively turn some of them on or off:
`cv colvar xi cvcflags <flags>`
where `<flags>` is a list of 0/1 values, one per component. This is useful for example when Tcl script-based path collective variables in Cartesian coordinates (5.10.3) are used, to minimize computational cost by disabling the computation of terms that are very close to zero.

Important: None of the changes enacted by `modifycvcs` or `cvcflags` will be saved to state files, and will be lost when restarting a simulation, deleting the corresponding collective variable, or resetting the module with `cv delete` or `cv reset`.

4.2.6 Applying and analyzing forces on collective variables

As soon as a collective variable is up to date (during a MD run or after `update` has been called), forces can be applied to it as part e.g. of a custom restraint implemented by `scriptedColvarForces`:
`cv colvar xi addforce $force`
where `$force` is a scalar or a vector (depending on the type of variable `xi`) defined by a user function. The force will be physically applied to the corresponding atoms during the simulation after Colvars communicate all forces to the rest of VMD. (In VMD, these forces will never have an effect.) Until then, the applied force from all biases can be retrieved by:
`cv colvar xi getappliedforce`
(see also the use of the `outputAppliedForce` option).

To obtain the total force projected on the variable `xi`:
`cv colvar xi gettotalforce`
Note that not all types of variable support this option, and the value of the total force may not be available immediately: see `outputTotalForce` for more details.

See 8.2 for a complete list of scripting commands used to manage collective variables.

4.2.7 Managing collective variable biases

Because biases depend only upon data internal to the Colvars module (i.e. they do not need atomic coordinates from VMD), it is generally easy to creat them or update their configuration at any time. For example, given the most current value of the variable `xi`, the restraint named `harmonic_xi` can be updated as:
`cv bias harmonic_xi update`
Again, this is not generally needed during a running simulation, when an automatic update of each bias is already carried out. Instead, calling `update` is most useful for just-defined biases or when changing their configuration. When `update` is called e.g. as part of the function invoked by `scriptedColvarForces`, it is executed before any biasing forces are applied to the variables, thus allowing to modify them.

A common use for `update` is part of the definition of bias-exchange algorithms as part of the VMD script. Because a bias is a relatively light-weight objects, the easiest way to change the configuration of an existing bias is deleting it and re-creating it:

```
# Delete the restraint "harmonic_xi"
cv bias harmonic_xi delete
# Re-define it, but using an updated restraint center
cv config "harmonic {
  name harmonic_xi
  centers ${new_center}]
  ...
}"
# Now update it (based on the current value of "xi")
cv bias harmonic_xi update
```

It is also possible to make the change subject to a condition on the energy of the new bias:

```
...
cv bias harmonic_xi update
if { [cv bias harmonic_xi energy] < ${E_accept} } {
    ...
}
```

4.2.8 Loading and saving the state of individual biases

Some types of bias are history-dependent, and the magnitude of their forces depends not only on the values of their corresponding variables, but also on previous simulation history. It is thus useful to load information from a state file that contains information specifically for one bias only, for example:
`cv bias metadynamics1 load old.colvars.state`
or alternatively, using the prefix of the file instead of its full name:
`cv bias metadynamics1 load old`
A corresponding `save` function is also available:
`cv bias metadynamics1 save new`
This pair of functions is also used internally by Colvars to implement e.g. multiple-walker metadynamics (7.4.7), but they can be called from a scripted function to implement alternative coupling schemes.

See 8.3 for a complete list of scripting commands used to manage biases.

4.3 Configuration syntax used by the Colvars module

The Colvars configuration is usually read using the commands `cv configfile` (with a filename as argument) or `cv config` (with the configuration as a string argument). Each configuration line follows the format "`keyword value`", where the keyword and its value are separated by any white space. The following rules apply:

- keywords are case-insensitive (`upperBoundary` is the same as `upperboundary` and `UPPERBOUNDARY`): their string values are however case-sensitive (e.g. file names);
- a long value, or a list of multiple values, can be distributed across multiple lines by using curly braces, "{" and "}": the opening brace "{" must occur on the same line as the keyword, following a space character or other white space; the closing brace "}" can be at any position after that; any keywords following the closing brace on the same line are not valid (they should appear instead on a different line);
- many keywords are nested, and are only meaningful within a specific context: for every keyword documented in the following, the "parent" keyword that defines such context is also indicated;
- Tcl syntax is generally not available, but it is possible to use Tcl variables or bracket expansion of commands within a configuration string, when this is passed via the command `cv config ...`: for example, it is possible to convert the atom selection $*sel* into an atom group (see 6.1) using `cv config "atomNumbers { [$sel get serial] }"`;
- if a keyword requiring a boolean value (`yes|on|true` or `no|off|false`) is provided without an explicit value, it defaults to '`yes|on|true`'; for example, '`outputAppliedForce`' may be used as shorthand for '`outputAppliedForce on`';

- the hash character # indicates a comment: all text in the same line following this character will be ignored.

4.4 Global keywords

The following keywords are available in the global context of the Colvars configuration, i.e. they are not nested inside other keywords:

- `colvarsTrajFrequency` ⟨Colvar value trajectory frequency⟩
  Context: global
  Acceptable values: positive integer
  Default value: `100`
  Description: The values of each colvar (and of other related quantities, if requested) are written to the file *outputName*`.colvars.traj` every these many steps throughout the simulation. If the value is `0`, such trajectory file is not written. For optimization the output is buffered, and synchronized with the disk only when the restart file is being written.

- `colvarsRestartFrequency` ⟨Colvar module restart frequency⟩
  Context: global
  Acceptable values: positive integer
  Default value: `restartFreq`
  Description: Allows to choose a different restart frequency for the Colvars module. Redefining it may be useful to trace the time evolution of those few properties which are not written to the trajectory file for reasons of disk space.

- `indexFile` ⟨Index file for atom selection (GROMACS "ndx" format)⟩
  Context: global
  Acceptable values: UNIX filename
  Description: This option reads an index file (usually with a `.ndx` extension) as produced by the `make_ndx` tool of GROMACS. This keyword may be repeated to load multiple index files. A group with the same name may appear multiple times, as long as it contains the same indices in identical order each time: an error is raised otherwise. The names of index groups contained in this file can then be used to define atom groups with the `indexGroup` keyword. Other supported methods to select atoms are described in 6.

- `smp` ⟨Whether SMP parallelism should be used⟩
  Context: global
  Acceptable values: boolean
  Default value: `on`
  Description: If this flag is enabled (default), SMP parallelism over threads will be used to compute variables and biases, provided that this is supported by the VMD build in use.

To illustrate the flexibility of the Colvars module, a non-trivial setup is represented in Figure 1. The corresponding configuration is given below. The options within the `colvar` blocks are described in 5, those within the `harmonic` and `histogram` blocks in 7. **Note:** *except* `colvar`*, none of the keywords shown is mandatory*.

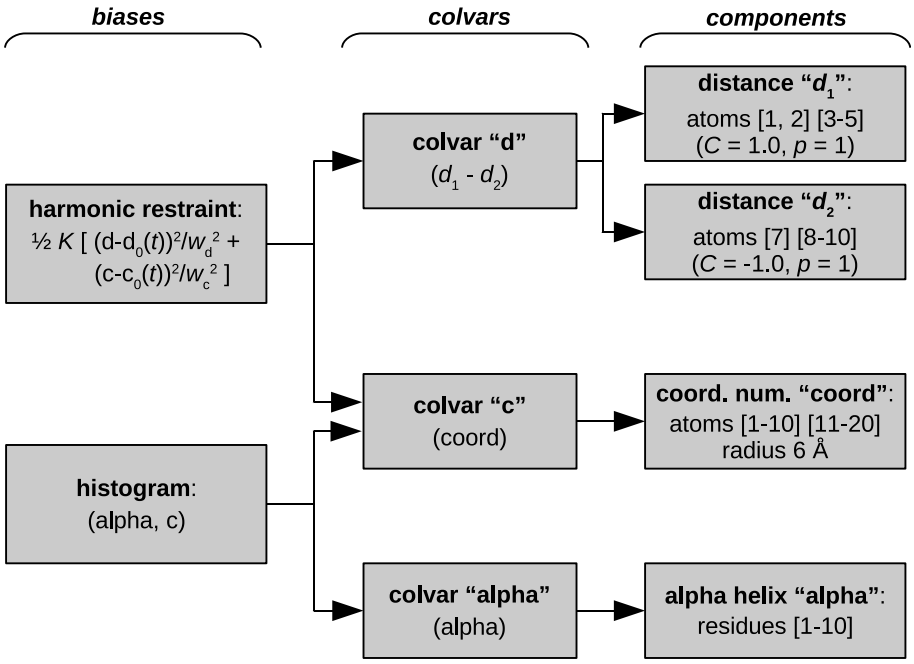


Figure 1: Graphical representation of a Colvars configuration. The colvar called "d" is defined as the difference between two distances: the first distance (d_1) is taken between the center of mass of atoms 1 and 2 and that of atoms 3 to 5, the second (d_2) between atom 7 and the center of mass of atoms 8 to 10. The difference $d = d_1 - d_2$ is obtained by multiplying the two by a coefficient $C = +1$ or $C = -1$, respectively. The colvar called "c" is the coordination number calculated between atoms 1 to 10 and atoms 11 to 20. A harmonic restraint is applied to both d and c: to allow using the same force constant K, both d and c are scaled by their respective fluctuation widths w_d and w_c. A third colvar "alpha" is defined as the α-helical content of residues 1 to 10. The values of "c" and "alpha" are also recorded throughout the simulation as a joint 2-dimensional histogram.

```
colvar {
  # difference of two distances
  name d
  width 0.2 # 0.2 Å of estimated fluctuation width
  distance {
    componentCoeff 1.0
    group1 { atomNumbers 1 2 }
    group2 { atomNumbers 3 4 5 }
  }
  distance {
    componentCoeff -1.0
    group1 { atomNumbers 7 }
    group2 { atomNumbers 8 9 10 }
  }
}

colvar {
  name c
```

```
  coordNum {
    cutoff 6.0
    group1 { atomNumbersRange 1-10 }
    group2 { atomNumbersRange 11-20 }
  }
}

colvar {
  name alpha
  alpha {
    psfSegID PROT
    residueRange 1-10
  }
}

harmonic {
  colvars d c
  centers 3.0 4.0
  forceConstant 5.0
}

histogram {
  colvars c alpha
}
```

Section 5 explains how to define a colvar and its behavior, regardless of its specific functional form. To define colvars that are appropriate to a specific physical system, Section 6 documents how to select atoms, and section 5 lists all of the available functional forms, which we call "colvar components". Finally, section 7 lists the available methods and algorithms to perform biased simulations and multidimensional analysis of colvars.

4.5 Input state file

Because many of the methods implemented in Colvars are history-dependent, a *state file* is often needed to continue a long simulation over consecutive runs. Such state file is written automatically at the end of any simulation with Colvars, and contains data accumulated during that simulation along with the step number at the end of it. The step number read from the state file is then used to control such time-dependent biases: because of this essential role, the step number internal to Colvars may not always match the step number reported by the MD program that carried during the simulation (which may instead restart from zero each time).

Depending on the configuration, a state file may need to be loaded issued at the beginning of a new simulation when time-dependent biasing methods are applied (moving restraints, metadynamics, ABF, ...). . After initialization, a state file may be loaded at any time with the Tcl command `cv load`.

It is possible to load a state file even if the configuration has changed: for example, new variables may be defined or restraints be added in between consecutive runs. For each newly defined variable or bias, no information will be read from the state file if this is unavailable: such new objects will remain uninitialized until the first compute step. Conversely, any information that the state file has about variables or biases that

are not defined any longer is silently ignored. *Because these checks are done by the names of variables or biases, it is the user's responsibility to ensure that these are consistent between runs.*

4.6 Output files

During a simulation with collective variables defined, the following three output files are written:

- A *state file*, named *outputName*`.colvars.state`; this file is in ASCII (plain text) format. This file is written at the end of the specified run, but can also be written at any time with the command `cv save` (**??**).
  This is the only Colvars output file needed to continue a simulation.
- If the parameter `colvarsRestartFrequency` is larger than zero, a *restart file* is written every that many steps: this file is fully equivalent to the final state file. The name of this file is *restartName*`.colvars.state`.
- If the parameter `colvarsTrajFrequency` is greater than 0 (default: 100), a *trajectory file* is written during the simulation: its name is *outputName*`.colvars.traj`; unlike the state file, it is not needed to restart a simulation, but can be used later for post-processing and analysis.

Other output files may also be written by specific methods, e.g. the ABF or metadynamics methods (7.2, 7.4). Like the trajectory file, they are needed only for analyzing, not continuing a simulation. All such files' names also begin with the prefix *outputName*.

5 Defining collective variables

A collective variable is defined by the keyword `colvar` followed by its configuration options contained within curly braces:

```
colvar {
  name xi
  <other options>
  function_name {
    <parameters>
    <atom selection>
  }
}
```

There are multiple ways of defining a variable:

- The *simplest and most common way* way is using one of the precompiled functions (here called "components"), which are listed in section 5.1. For example, using the keyword `rmsd` (section 5.5.1) defines the variable as the root mean squared deviation (RMSD) of the selected atoms.
- A new variable may also be constructed as a linear or polynomial combination of the components listed in section 5.1 (see 5.14 for details).
- A user-defined Tcl function of the existing components (see list in section 5.1), or of the atomic coordinates directly (see the `cartesian` keyword in 5.8.1). The function is provided by a separate Tcl script, and referenced through the keyword `scriptedFunction` (see 5.15 for details).

Choosing a component (function) is the only parameter strictly required to define a collective variable. It is also highly recommended to specify a name for the variable:

- `name` ⟨Name of this colvar⟩
  Context: `colvar`
  Acceptable values: string
  Default value: "`colvar`" + numeric id
  Description: The name is an unique case-sensitive string which allows the Colvars module to identify this colvar unambiguously; it is also used in the trajectory file to label to the columns corresponding to this colvar.

5.1 Choosing a function

In this context, the function that computes a colvar is called a *component*. A component's choice and definition consists of including in the variable's configuration a keyword indicating the type of function (e.g. `rmsd`), followed by a definition block specifying the atoms involved (see 6) and any additional parameters (cutoffs, "reference" values, ...). *At least one component must be chosen to define a variable:* if none of the keywords listed below is found, an error is raised.

The following components implement functions with a scalar value (i.e. a real number):

- distance: distance between two groups;
- distanceZ: projection of a distance vector on an axis;
- distanceXY: projection of a distance vector on a plane;
- distanceInv: mean distance between two groups of atoms (e.g. NOE-based distance);
- angle: angle between three groups;
- dihedral: torsional (dihedral) angle between four groups;
- dipoleAngle: angle between two groups and dipole of a third group;
- dipoleMagnitude magnitude of the dipole of a group of atoms;
- polarTheta: polar angle of a group in spherical coordinates;
- polarPhi: azimuthal angle of a group in spherical coordinates;
- coordNum: coordination number between two groups;
- selfCoordNum: coordination number of atoms within a group;
- hBond: hydrogen bond between two atoms;
- rmsd: root mean square deviation (RMSD) from a set of reference coordinates;
- eigenvector: projection of the atomic coordinates on a vector;
- orientationAngle: angle of the best-fit rotation from a set of reference coordinates;
- orientationProj: cosine of orientationProj;
- spinAngle: projection orthogonal to an axis of the best-fit rotation from a set of reference coordinates;
- tilt: projection on an axis of the best-fit rotation from a set of reference coordinates;
- gyration: radius of gyration of a group of atoms;
- inertia: moment of inertia of a group of atoms;
- inertiaZ: moment of inertia of a group of atoms around a chosen axis;
- alpha: α-helix content of a protein segment.
- dihedralPC: projection of protein backbone dihedrals onto a dihedral principal component.

Some components do not return scalar, but vector values:

- distanceVec: distance vector between two groups (length: 3);
- distanceDir: unit vector parallel to distanceVec (length: 3);
- cartesian: vector of atomic Cartesian coordinates (length: N times the number of Cartesian components requested, X, Y or Z);
- distancePairs: vector of mutual distances (length: $N_1 \times N_2$);

- `orientation`: best-fit rotation, expressed as a unit quaternion (length: 4).

The types of components used in a colvar (scalar or not) determine the properties of that colvar, and particularly which biasing or analysis methods can be applied.

What if "X" is not listed? If a function type is not available on this list, it may be possible to define it as a polynomial superposition of existing ones (see 5.14), or a scripted function (see 5.15).

In the rest of this section, all available component types are listed, along with their physical units and the ranges of values, if limited. Such limiting values can be used to define `lowerBoundary` and `upperBoundary` in the parent colvar.

For each type of component, the available configurations keywords are listed: when two components share certain keywords, the second component references to the documentation of the first one that uses that keyword. The very few keywords that are available for all types of components are listed in a separate section 5.11.

5.2 Distances

5.2.1 `distance`: center-of-mass distance between two groups.

The `distance` {...} block defines a distance component between the two atom groups, `group1` and `group2`.

List of keywords (see also 5.14 for additional options):

- `group1` ⟨First group of atoms⟩
  Context: `distance`
  Acceptable values: Block `group1` {...}
  Description: First group of atoms.

- `group2`: analogous to `group1`

- `forceNoPBC` ⟨Calculate absolute rather than minimum-image distance?⟩
  Context: `distance`
  Acceptable values: boolean
  Default value: `no`
  Description: By default, in calculations with periodic boundary conditions, the `distance` component returns the distance according to the minimum-image convention. If this parameter is set to `yes`, PBC will be ignored and the distance between the coordinates as maintained internally will be used. This is only useful in a limited number of special cases, e.g. to describe the distance between remote points of a single macromolecule, which cannot be split across periodic cell boundaries, and for which the minimum-image distance might give the wrong result because of a relatively small periodic cell.

- `oneSiteTotalForce` ⟨Measure total force on group 1 only?⟩
  Context: `angle`, `dipoleAngle`, `dihedral`
  Acceptable values: boolean
  Default value: `no`
  Description: If this is set to `yes`, the total force is measured along a vector field (see equation (25) in section 7.2) that only involves atoms of `group1`. This option is only useful for ABF, or custom biases that compute total forces. See section 7.2 for details.

The value returned is a positive number (in Å), ranging from 0 to the largest possible interatomic distance within the chosen boundary conditions (with PBCs, the minimum image convention is used unless the `forceNoPBC` option is set).

5.2.2 `distanceZ`: projection of a distance vector on an axis.

The `distanceZ` {...} block defines a distance projection component, which can be seen as measuring the distance between two groups projected onto an axis, or the position of a group along such an axis. The axis can be defined using either one reference group and a constant vector, or dynamically based on two reference groups. One of the groups can be set to a dummy atom to allow the use of an absolute Cartesian coordinate.
List of keywords (see also 5.14 for additional options):

- `main` ⟨Main group of atoms⟩
  Context: `distanceZ`
  Acceptable values: Block `main` {...}
  Description: Group of atoms whose position r is measured.

- `ref` ⟨Reference group of atoms⟩
  Context: `distanceZ`
  Acceptable values: Block `ref` {...}
  Description: Reference group of atoms. The position of its center of mass is noted r_1 below.

- `ref2` ⟨Secondary reference group⟩
  Context: `distanceZ`
  Acceptable values: Block `ref2` {...}
  Default value: none
  Description: Optional group of reference atoms, whose position r_2 can be used to define a dynamic projection axis: $e = (\|r_2 - r_1\|)^{-1} \times (r_2 - r_1)$. In this case, the origin is $r_m = 1/2(r_1 + r_2)$, and the value of the component is $e \cdot (r - r_m)$.

- `axis` ⟨Projection axis (Å)⟩
  Context: `distanceZ`
  Acceptable values: `(x, y, z)` triplet
  Default value: `(0.0, 0.0, 1.0)`
  Description: The three components of this vector define a projection axis e for the distance vector $r - r_1$ joining the centers of groups `ref` and `main`. The value of the component is then $e \cdot (r - r_1)$. The vector should be written as three components separated by commas and enclosed in parentheses.

- `forceNoPBC`: see definition of `forceNoPBC` in sec. 5.2.1 (`distance` component)

- `oneSiteTotalForce`: see definition of `oneSiteTotalForce` in sec. 5.2.1 (`distance` component)

This component returns a number (in Å) whose range is determined by the chosen boundary conditions. For instance, if the z axis is used in a simulation with periodic boundaries, the returned value ranges between $-b_z/2$ and $b_z/2$, where b_z is the box length along z (this behavior is disabled if `forceNoPBC` is set).

5.2.3 `distanceXY`: modulus of the projection of a distance vector on a plane.

The `distanceXY` {...} block defines a distance projected on a plane, and accepts the same keywords as the component `distanceZ`, i.e. `main`, `ref`, either `ref2` or `axis`, and `oneSiteTotalForce`. It returns the norm of the projection of the distance vector between `main` and `ref` onto the plane orthogonal to the axis. The axis is defined using the `axis` parameter or as the vector joining `ref` and `ref2` (see `distanceZ` above).
List of keywords (see also 5.14 for additional options):

- `main`: see definition of `main` in sec. 5.2.2 (`distanceZ` component)
- `ref`: see definition of `ref` in sec. 5.2.2 (`distanceZ` component)
- `ref2`: see definition of `ref2` in sec. 5.2.2 (`distanceZ` component)
- `axis`: see definition of `axis` in sec. 5.2.2 (`distanceZ` component)
- `forceNoPBC`: see definition of `forceNoPBC` in sec. 5.2.1 (`distance` component)
- `oneSiteTotalForce`: see definition of `oneSiteTotalForce` in sec. 5.2.1 (`distance` component)

5.2.4 `distanceVec`: distance vector between two groups.

The `distanceVec` {...} block defines a distance vector component, which accepts the same keywords as the component `distance`: `group1`, `group2`, and `forceNoPBC`. Its value is the 3-vector joining the centers of mass of `group1` and `group2`.
List of keywords (see also 5.14 for additional options):

- `group1`: see definition of `group1` in sec. 5.2.1 (`distance` component)
- `group2`: analogous to `group1`
- `forceNoPBC`: see definition of `forceNoPBC` in sec. 5.2.1 (`distance` component)
- `oneSiteTotalForce`: see definition of `oneSiteTotalForce` in sec. 5.2.1 (`distance` component)

5.2.5 `distanceDir`: distance unit vector between two groups.

The `distanceDir` {...} block defines a distance unit vector component, which accepts the same keywords as the component `distance`: `group1`, `group2`, and `forceNoPBC`. It returns a 3-dimensional unit vector $\mathbf{d} = (d_x, d_y, d_z)$, with $|\mathbf{d}| = 1$.
List of keywords (see also 5.14 for additional options):

- `group1`: see definition of `group1` in sec. 5.2.1 (`distance` component)
- `group2`: analogous to `group1`
- `forceNoPBC`: see definition of `forceNoPBC` in sec. 5.2.1 (`distance` component)
- `oneSiteTotalForce`: see definition of `oneSiteTotalForce` in sec. 5.2.1 (`distance` component)

5.2.6 `distanceInv`: mean distance between two groups of atoms.

The `distanceInv` {...} block defines a generalized mean distance between two groups of atoms 1 and 2, weighted with exponent $1/n$:

$$d_{1,2}^{[n]} = \left(\frac{1}{N_1 N_2} \sum_{i,j} \left(\frac{1}{\|\mathbf{d}^{ij}\|} \right)^n \right)^{-1/n} \tag{2}$$

where $\|\mathbf{d}^{ij}\|$ is the distance between atoms i and j in groups 1 and 2 respectively, and n is an even integer.
List of keywords (see also 5.14 for additional options):

- `group1`: see definition of `group1` in sec. 5.2.1 (`distance` component)
- `group2`: analogous to `group1`
- `oneSiteTotalForce`: see definition of `oneSiteTotalForce` in sec. 5.2.1 (`distance` component)
- `exponent` ⟨Exponent n in equation 2⟩
  Context: `distanceInv`
  Acceptable values: positive even integer
  Default value: 6
  Description: Defines the exponent to which the individual distances are elevated before averaging. The default value of 6 is useful for example to applying restraints based on NOE-measured distances.

This component returns a number in Å, ranging from 0 to the largest possible distance within the chosen boundary conditions.

5.3 Angles

5.3.1 `angle`: angle between three groups.

The `angle` {...} block defines an angle, and contains the three blocks `group1`, `group2` and `group3`, defining the three groups. It returns an angle (in degrees) within the interval $[0:180]$.
List of keywords (see also 5.14 for additional options):

- `group1`: see definition of `group1` in sec. 5.2.1 (`distance` component)
- `group2`: analogous to `group1`
- `group3`: analogous to `group1`
- `forceNoPBC`: see definition of `forceNoPBC` in sec. 5.2.1 (`distance` component)
- `oneSiteTotalForce`: see definition of `oneSiteTotalForce` in sec. 5.2.1 (`distance` component)

5.3.2 `dipoleAngle`: angle between two groups and dipole of a third group.

The `dipoleAngle` {...} block defines an angle, and contains the three blocks `group1`, `group2` and `group3`, defining the three groups, being `group1` the group where dipole is calculated. It returns an angle (in degrees) within the interval $[0:180]$.
List of keywords (see also 5.14 for additional options):

- `group1`: see definition of `group1` in sec. 5.2.1 (`distance` component)
- `group2`: analogous to `group1`
- `group3`: analogous to `group1`
- `forceNoPBC`: see definition of `forceNoPBC` in sec. 5.2.1 (`distance` component)
- `oneSiteTotalForce`: see definition of `oneSiteTotalForce` in sec. 5.2.1 (`distance` component)

5.3.3 `dihedral`: torsional angle between four groups.

The `dihedral` {...} block defines a torsional angle, and contains the blocks `group1`, `group2`, `group3` and `group4`, defining the four groups. It returns an angle (in degrees) within the interval $[-180:180]$. The Colvars module calculates all the distances between two angles taking into account periodicity. For instance, reference values for restraints or range boundaries can be defined by using any real number of choice.
List of keywords (see also 5.14 for additional options):

- `group1`: see definition of `group1` in sec. 5.2.1 (`distance` component)
- `group2`: analogous to `group1`
- `group3`: analogous to `group1`
- `group4`: analogous to `group1`
- `forceNoPBC`: see definition of `forceNoPBC` in sec. 5.2.1 (`distance` component)
- `oneSiteTotalForce`: see definition of `oneSiteTotalForce` in sec. 5.2.1 (`distance` component)

5.3.4 `polarTheta`: polar angle in spherical coordinates.

The `polarTheta` {...} block defines the polar angle in spherical coordinates, for the center of mass of a group of atoms described by the block `atoms`. It returns an angle (in degrees) within the interval $[0:180]$. To obtain spherical coordinates in a frame of reference tied to another group of atoms, use the `fittingGroup` (6.2) option within the `atoms` block. An example is provided in file `examples/11_polar_angles.in` of the Colvars public repository.
List of keywords (see also 5.14 for additional options):

- `atoms` ⟨Atom group⟩
  Context: `polarPhi`
  Acceptable values: `atoms` {...} block
  Description: Defines the group of atoms for the COM of which the angle should be calculated.

5.3.5 `polarPhi`: azimuthal angle in spherical coordinates.

The `polarPhi` {...} block defines the azimuthal angle in spherical coordinates, for the center of mass of a group of atoms described by the block `atoms`. It returns an angle (in degrees) within the interval $[-180:180]$. The Colvars module calculates all the distances between two angles taking into account periodicity. For instance, reference values for restraints or range boundaries can be defined by using any real number of choice. To obtain spherical coordinates in a frame of reference tied to another group of atoms, use the `fittingGroup` (6.2) option within the `atoms` block. An example is provided in file `examples/11_polar_angles.in` of the Colvars public repository.
List of keywords (see also 5.14 for additional options):

- `atoms` ⟨Atom group⟩
  Context: `polarPhi`
  Acceptable values: `atoms` {...} block
  Description: Defines the group of atoms for the COM of which the angle should be calculated.

5.4 Contacts

5.4.1 `coordNum`: coordination number between two groups.

The `coordNum` {...} block defines a coordination number (or number of contacts), which calculates the function $(1-(d/d_0)^n)/(1-(d/d_0)^m)$, where d_0 is the "cutoff" distance, and n and m are exponents that can control its long range behavior and stiffness [2]. This function is summed over all pairs of atoms in `group1` and `group2`:

$$C(\texttt{group1},\texttt{group2}) = \sum_{i\in\texttt{group1}} \sum_{j\in\texttt{group2}} \frac{1-(|\mathbf{x}_i-\mathbf{x}_j|/d_0)^n}{1-(|\mathbf{x}_i-\mathbf{x}_j|/d_0)^m} \tag{3}$$

List of keywords (see also 5.14 for additional options):

- `group1`: see definition of `group1` in sec. 5.2.1 (`distance` component)
- `group2`: analogous to `group1`
- `cutoff` ⟨"Interaction" distance (Å)⟩
  Context: `coordNum`
  Acceptable values: positive decimal
  Default value: 4.0
  Description: This number defines the switching distance to define an interatomic contact: for $d \ll d_0$, the switching function $(1-(d/d_0)^n)/(1-(d/d_0)^m)$ is close to 1, at $d = d_0$ it has a value of n/m ($1/2$ with the default n and m), and at $d \gg d_0$ it goes to zero approximately like d^{m-n}. Hence, for a proper behavior, m must be larger than n.
- `cutoff3` ⟨Reference distance vector (Å)⟩
  Context: `coordNum`
  Acceptable values: "`(x, y, z)`" triplet of positive decimals
  Default value: `(4.0, 4.0, 4.0)`
  Description: The three components of this vector define three different cutoffs d_0 for each direction. This option is mutually exclusive with `cutoff`.

- `expNumer` ⟨Numerator exponent⟩
  Context: `coordNum`
  Acceptable values: positive even integer
  Default value: 6
  Description: This number defines the n exponent for the switching function.

- `expDenom` ⟨Denominator exponent⟩
  Context: `coordNum`
  Acceptable values: positive even integer
  Default value: 12
  Description: This number defines the m exponent for the switching function.

- `group2CenterOnly` ⟨Use only `group2`'s center of mass⟩
  Context: `coordNum`
  Acceptable values: boolean
  Default value: `off`
  Description: If this option is `on`, only contacts between each atoms in `group1` and the center of mass of `group2` are calculated (by default, the sum extends over all pairs of atoms in `group1` and `group2`). If `group2` is a `dummyAtom`, this option is set to `yes` by default.

- `tolerance` ⟨Pairlist control⟩
  Context: `coordNum`
  Acceptable values: decimal
  Default value: 0.0
  Description: This controls the pairlist feature, dictating the minimum value for each summation element in Eq. 3 such that the pair that contributed the summation element is included in subsequent simulation timesteps until the next pairlist recalculation. For most applications, this value should be small (eg. 0.001) to avoid missing important contributions to the overall sum. Higher values will improve performance by reducing the number of pairs that contribute to the sum. Values above 1 will exclude all possible pair interactions. Similarly, values below 0 will never exclude a pair from consideration. To ensure continuous forces, Eq. 3 is further modified by subtracting the tolerance and then rescaling so that each pair covers the range $[0,1]$.

- `pairListFrequency` ⟨Pairlist regeneration frequency⟩
  Context: `coordNum`
  Acceptable values: positive integer
  Default value: 100
  Description: This controls the pairlist feature, dictating how many steps are taken between regenerating pairlists if the tolerance is greater than 0.

This component returns a dimensionless number, which ranges from approximately 0 (all interatomic distances are much larger than the cutoff) to $N_{\texttt{group1}} \times N_{\texttt{group2}}$ (all distances are less than the cutoff), or $N_{\texttt{group1}}$ if `group2CenterOnly` is used. For performance reasons, at least one of `group1` and `group2` should be of limited size or `group2CenterOnly` should be used: the cost of the loop over all pairs grows as $N_{\texttt{group1}} \times N_{\texttt{group2}}$. Setting `tolerance` > 0 ameliorates this to some degree, although every pair is still checked to regenerate the pairlist.

5.4.2 `selfCoordNum`: coordination number between atoms within a group.

The `selfCoordNum` {...} block defines a coordination number similarly to the component `coordNum`, but the function is summed over atom pairs within `group1`:

$$C(\texttt{group1}) = \sum_{i \in \texttt{group1}} \sum_{j>i} \frac{1-(|\mathbf{x}_i - \mathbf{x}_j|/d_0)^n}{1-(|\mathbf{x}_i - \mathbf{x}_j|/d_0)^m} \tag{4}$$

The keywords accepted by `selfCoordNum` are a subset of those accepted by `coordNum`, namely `group1` (here defining *all* of the atoms to be considered), `cutoff`, `expNumer`, and `expDenom`.
List of keywords (see also 5.14 for additional options):

- `group1`: see definition of `group1` in sec. 5.4.1 (`coordNum` component)
- `cutoff`: see definition of `cutoff` in sec. 5.4.1 (`coordNum` component)
- `cutoff3`: see definition of `cutoff3` in sec. 5.4.1 (`coordNum` component)
- `expNumer`: see definition of `expNumer` in sec. 5.4.1 (`coordNum` component)
- `expDenom`: see definition of `expDenom` in sec. 5.4.1 (`coordNum` component)
- `tolerance`: see definition of `tolerance` in sec. 5.4.1 (`coordNum` component)
- `pairListFrequency`: see definition of `pairListFrequency` in sec. 5.4.1 (`coordNum` component)

This component returns a dimensionless number, which ranges from approximately 0 (all interatomic distances much larger than the cutoff) to $N_{\texttt{group1}} \times (N_{\texttt{group1}} - 1)/2$ (all distances within the cutoff). For performance reasons, `group1` should be of limited size, because the cost of the loop over all pairs grows as $N^2_{\texttt{group1}}$.

5.4.3 `hBond`: hydrogen bond between two atoms.

The `hBond` {...} block defines a hydrogen bond, implemented as a coordination number (eq. 3) between the donor and the acceptor atoms. Therefore, it accepts the same options `cutoff` (with a different default value of 3.3 Å), `expNumer` (with a default value of 6) and `expDenom` (with a default value of 8). Unlike `coordNum`, it requires two atom numbers, `acceptor` and `donor`, to be defined. It returns an adimensional number, with values between 0 (acceptor and donor far outside the cutoff distance) and 1 (acceptor and donor much closer than the cutoff).
List of keywords (see also 5.14 for additional options):

- `acceptor` ⟨Number of the acceptor atom⟩
  Context: `hBond`
  Acceptable values: positive integer
  Description: Number that uses the same convention as `atomNumbers`.
- `donor`: analogous to `acceptor`
- `cutoff`: see definition of `cutoff` in sec. 5.4.1 (`coordNum` component)
  Note: default value is 3.3 Å.
- `expNumer`: see definition of `expNumer` in sec. 5.4.1 (`coordNum` component)
  Note: default value is 6.

- `expDenom`: see definition of `expDenom` in sec. 5.4.1 (`coordNum` component)
  Note: default value is 8.

5.5 Collective metrics

5.5.1 `rmsd`: root mean square displacement (RMSD) from reference positions.

The block `rmsd` $\{\ldots\}$ defines the root mean square replacement (RMSD) of a group of atoms with respect to a reference structure. For each set of coordinates $\{\mathbf{x}_1(t), \mathbf{x}_2(t), \ldots \mathbf{x}_N(t)\}$, the colvar component `rmsd` calculates the optimal rotation $U^{\{\mathbf{x}_i(t)\}\to\{\mathbf{x}_i^{(\mathrm{ref})}\}}$ that best superimposes the coordinates $\{\mathbf{x}_i(t)\}$ onto a set of reference coordinates $\{\mathbf{x}_i^{(\mathrm{ref})}\}$. Both the current and the reference coordinates are centered on their centers of geometry, $\mathbf{x}_{\mathrm{cog}}(t)$ and $\mathbf{x}_{\mathrm{cog}}^{(\mathrm{ref})}$. The root mean square displacement is then defined as:

$$\mathrm{RMSD}(\{\mathbf{x}_i(t)\}, \{\mathbf{x}_i^{(\mathrm{ref})}\}) \;=\; \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left|U\left(\mathbf{x}_i(t)-\mathbf{x}_{\mathrm{cog}}(t)\right)-\left(\mathbf{x}_i^{(\mathrm{ref})}-\mathbf{x}_{\mathrm{cog}}^{(\mathrm{ref})}\right)\right|^2} \tag{5}$$

The optimal rotation $U^{\{\mathbf{x}_i(t)\}\to\{\mathbf{x}_i^{(\mathrm{ref})}\}}$ is calculated within the formalism developed in reference [3], which guarantees a continuous dependence of $U^{\{\mathbf{x}_i(t)\}\to\{\mathbf{x}_i^{(\mathrm{ref})}\}}$ with respect to $\{\mathbf{x}_i(t)\}$.
List of keywords (see also 5.14 for additional options):

- `atoms` ⟨Atom group⟩
  Context: `rmsd`
  Acceptable values: `atoms` $\{\ldots\}$ block
  Description: Defines the group of atoms of which the RMSD should be calculated. Optimal fit options (such as `refPositions` and `rotateReference`) should typically NOT be set within this block. Exceptions to this rule are the special cases discussed in the *Advanced usage* paragraph below.

- `refPositions` ⟨Reference coordinates⟩
  Context: `rmsd`
  Acceptable values: space-separated list of `(x, y, z)` triplets
  Description: This option (mutually exclusive with `refPositionsFile`) sets the reference coordinates for RMSD calculation, and uses these to compute the roto-translational fit. It is functionally equivalent to the option `refPositions` in the atom group definition, which also supports more advanced fitting options.

- `refPositionsFile` ⟨Reference coordinates file⟩
  Context: `rmsd`
  Acceptable values: UNIX filename
  Description: This option (mutually exclusive with `refPositions`) sets the reference coordinates for RMSD calculation, and uses these to compute the roto-translational fit. It is functionally equivalent to the option `refPositionsFile` in the atom group definition, which also supports more advanced fitting options.

- `refPositionsCol` ⟨PDB column containing atom flags⟩
  Context: `rmsd`

Acceptable values: `O`, `B`, `X`, `Y`, or `Z`
Description: If `refPositionsFile` is a PDB file that contains all the atoms in the topology, this option may be provided to set which PDB field is used to flag the reference coordinates for `atoms`.

- `refPositionsColValue` ⟨Atom selection flag in the PDB column⟩
  Context: `rmsd`
  Acceptable values: positive decimal
  Description: If defined, this value identifies in the PDB column `refPositionsCol` of the file `refPositionsFile` which atom positions are to be read. Otherwise, all positions with a non-zero value are read.

This component returns a positive real number (in Å).

5.5.2 Advanced usage of the `rmsd` component.

In the standard usage as described above, the `rmsd` component calculates a minimum RMSD, that is, current coordinates are optimally fitted onto the same reference coordinates that are used to compute the RMSD value. The fit itself is handled by the atom group object, whose parameters are automatically set by the `rmsd` component. For very specific applications, however, it may be useful to control the fitting process separately from the definition of the reference coordinates, to evaluate various types of non-minimal RMSD values. This can be achieved by setting the related options (`refPositions`, etc.) explicitly in the atom group block. This allows for the following non-standard cases:

1. applying the optimal translation, but no rotation (`rotateReference off`), to bias or restrain the shape and orientation, but not the position of the atom group;
2. applying the optimal rotation, but no translation (`centerReference off`), to bias or restrain the shape and position, but not the orientation of the atom group;
3. disabling the application of optimal roto-translations, which lets the RMSD component decribe the deviation of atoms from fixed positions in the laboratory frame: this allows for custom positional restraints within the Colvars module;
4. fitting the atomic positions to different reference coordinates than those used in the RMSD calculation itself (by specifying `refPositions` or `refPositionsFile` within the atom group as well as within the `rmsd` block);
5. applying the optimal rotation and/or translation from a separate atom group, defined through `fittingGroup`: the RMSD then reflects the deviation from reference coordinates in a separate, moving reference frame (see example in the section on `fittingGroup`).

5.5.3 `eigenvector`: projection of the atomic coordinates on a vector.

The block `eigenvector` `{...}` defines the projection of the coordinates of a group of atoms (or more precisely, their deviations from the reference coordinates) onto a vector in $\mathbb{R}^{3n}$, where n is the number of atoms in the group. The computed quantity is the total projection:

$$p(\{\mathbf{x}_i(t)\},\{\mathbf{x}_i^{(\mathrm{ref})}\}) \;=\; \sum_{i=1}^{n} \mathbf{v}_i \cdot \Big(U(\mathbf{x}_i(t) - \mathbf{x}_{\mathrm{cog}}(t)) - (\mathbf{x}_i^{(\mathrm{ref})} - \mathbf{x}_{\mathrm{cog}}^{(\mathrm{ref})}) \Big)\,, \tag{6}$$

where, as in the `rmsd` component, U is the optimal rotation matrix, $\mathbf{x}_{\mathrm{cog}}(t)$ and $\mathbf{x}_{\mathrm{cog}}^{(\mathrm{ref})}$ are the centers of geometry of the current and reference positions respectively, and $\mathbf{v}_i$ are the components of the vector for each atom. Example choices for $(\mathbf{v}_i)$ are an eigenvector of the covariance matrix (essential mode), or a normal mode of the system. It is assumed that $\sum_i \mathbf{v}_i = 0$: otherwise, the Colvars module centers the $\mathbf{v}_i$ automatically when reading them from the configuration.
List of keywords (see also 5.14 for additional options):

- `atoms`: see definition of `atoms` in sec. 5.5.1 (`rmsd` component)
- `refPositions`: see definition of `refPositions` in sec. 5.5.1 (`rmsd` component)
- `refPositionsFile`: see definition of `refPositionsFile` in sec. 5.5.1 (`rmsd` component)
- `refPositionsCol`: see definition of `refPositionsCol` in sec. 5.5.1 (`rmsd` component)
- `refPositionsColValue`: see definition of `refPositionsColValue` in sec. 5.5.1 (`rmsd` component)
- `vector` ⟨Vector components⟩
  Context: `eigenvector`
  Acceptable values: space-separated list of `(x, y, z)` triplets
  Description: This option (mutually exclusive with `vectorFile`) sets the values of the vector components.
- `vectorFile` ⟨file containing vector components⟩
  Context: `eigenvector`
  Acceptable values: UNIX filename
  Description: This option (mutually exclusive with `vector`) sets the name of a coordinate file containing the vector components; the file is read according to the same format used for `refPositionsFile`. For a PDB file specifically, the components are read from the X, Y and Z fields. **Note:** *The PDB file has limited precision and fixed-point numbers: in some cases, the vector components may not be accurately represented; a XYZ file should be used instead, containing floating-point numbers.*
- `vectorCol` ⟨PDB column used to flag participating atoms⟩
  Context: `eigenvector`
  Acceptable values: `O` or `B`
  Description: Analogous to `atomsCol`.
- `vectorColValue` ⟨Value used to flag participating atoms in the PDB file⟩
  Context: `eigenvector`
  Acceptable values: positive decimal
  Description: Analogous to `atomsColValue`.
- `differenceVector` ⟨The $3n$-dimensional vector is the difference between `vector` and `refPositions`⟩
  Context: `eigenvector`
  Acceptable values: boolean
  Default value: `off`
  Description: If this option is on, the numbers provided by `vector` or `vectorFile` are interpreted as another set of positions, $\mathbf{x}'_i$: the vector $\mathbf{v}_i$ is then defined as $\mathbf{v}_i = \left(\mathbf{x}'_i - \mathbf{x}_i^{(\mathrm{ref})}\right)$. This allows to conveniently define a colvar ξ as a projection on the linear transformation between two sets of positions, "A" and "B". For convenience, the vector is also normalized so that $\xi = 0$ when the atoms are at the set of positions "A" and $\xi = 1$ at the set of positions "B".

This component returns a number (in Å), whose value ranges between the smallest and largest absolute positions in the unit cell during the simulations (see also `distanceZ`). Due to the normalization in eq. 6, this range does not depend on the number of atoms involved.

5.5.4 `gyration`: radius of gyration of a group of atoms.

The block `gyration` {...} defines the parameters for calculating the radius of gyration of a group of atomic positions $\{\mathbf{x}_1(t), \mathbf{x}_2(t), \ldots \mathbf{x}_N(t)\}$ with respect to their center of geometry, $\mathbf{x}_{\mathrm{cog}}(t)$:

$$R_{\mathrm{gyr}} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left|\mathbf{x}_i(t) - \mathbf{x}_{\mathrm{cog}}(t)\right|^2} \tag{7}$$

This component must contain one `atoms` {...} block to define the atom group, and returns a positive number, expressed in Å.
List of keywords (see also 5.14 for additional options):

- `atoms`: see definition of `atoms` in sec. 5.5.1 (`rmsd` component)

5.5.5 `inertia`: total moment of inertia of a group of atoms.

The block `inertia` {...} defines the parameters for calculating the total moment of inertia of a group of atomic positions $\{\mathbf{x}_1(t), \mathbf{x}_2(t), \ldots \mathbf{x}_N(t)\}$ with respect to their center of geometry, $\mathbf{x}_{\mathrm{cog}}(t)$:

$$I = \sum_{i=1}^{N}\left|\mathbf{x}_i(t) - \mathbf{x}_{\mathrm{cog}}(t)\right|^2 \tag{8}$$

Note that all atomic masses are set to 1 for simplicity. This component must contain one `atoms` {...} block to define the atom group, and returns a positive number, expressed in Å^2.
List of keywords (see also 5.14 for additional options):

- `atoms`: see definition of `atoms` in sec. 5.5.1 (`rmsd` component)

5.5.6 `dipoleMagnitude`: dipole magnitude of a group of atoms.

The `dipoleMagnitude` {...} block defines the dipole magnitude of a group of atoms (norm of the dipole moment's vector), being `atoms` the group where dipole magnitude is calculated. It returns the magnitude in elementary charge e times Å.
List of keywords (see also 5.14 for additional options):

- `atoms`: see definition of `atoms` in sec. 5.5.1 (`rmsd` component)

5.5.7 `inertiaZ`: total moment of inertia of a group of atoms around a chosen axis.

The block `inertiaZ {...}` defines the parameters for calculating the component along the axis **e** of the moment of inertia of a group of atomic positions $\{\mathbf{x}_1(t), \mathbf{x}_2(t), \ldots \mathbf{x}_N(t)\}$ with respect to their center of geometry, $\mathbf{x}_{\mathrm{cog}}(t)$:

$$I_{\mathbf{e}} = \sum_{i=1}^{N} \left(\left(\mathbf{x}_i(t) - \mathbf{x}_{\mathrm{cog}}(t) \right) \cdot \mathbf{e} \right)^2 \tag{9}$$

Note that all atomic masses are set to 1 for simplicity. This component must contain one `atoms {...}` block to define the atom group, and returns a positive number, expressed in Å^2.
List of keywords (see also 5.14 for additional options):

- `atoms`: see definition of `atoms` in sec. 5.5.1 (`rmsd` component)
- `axis` ⟨Projection axis (Å)⟩
  Context: `inertiaZ`
  Acceptable values: `(x, y, z)` triplet
  Default value: `(0.0, 0.0, 1.0)`
  Description: The three components of this vector define (when normalized) the projection axis **e**.

5.6 Rotations

5.6.1 `orientation`: orientation from reference coordinates.

The block `orientation {...}` returns the same optimal rotation used in the `rmsd` component to superimpose the coordinates $\{\mathbf{x}_i(t)\}$ onto a set of reference coordinates $\{\mathbf{x}_i^{(\mathrm{ref})}\}$. Such component returns a four dimensional vector $\mathtt{q} = (q_0, q_1, q_2, q_3)$, with $\sum_i q_i^2 = 1$; this *quaternion* expresses the optimal rotation $\{\mathbf{x}_i(t)\} \rightarrow \{\mathbf{x}_i^{(\mathrm{ref})}\}$ according to the formalism in reference [3]. The quaternion (q_0, q_1, q_2, q_3) can also be written as $(\cos(\theta/2), \sin(\theta/2)\mathbf{u})$, where θ is the angle and **u** the normalized axis of rotation; for example, a rotation of 90° around the z axis is expressed as "`(0.707, 0.0, 0.0, 0.707)`". The script `quaternion2rmatrix.tcl` provides Tcl functions for converting to and from a 4×4 rotation matrix in a format suitable for usage in VMD.

As for the component `rmsd`, the available options are `atoms`, `refPositionsFile`, `refPositionsCol` and `refPositionsColValue`, and `refPositions`.

Note: `refPositions`and `refPositionsFile` define the set of positions *from which* the optimal rotation is calculated, but this rotation is not applied to the coordinates of the atoms involved: it is used instead to define the variable itself.
List of keywords (see also 5.14 for additional options):

- `atoms`: see definition of `atoms` in sec. 5.5.1 (`rmsd` component)
- `refPositions`: see definition of `refPositions` in sec. 5.5.1 (`rmsd` component)
- `refPositionsFile`: see definition of `refPositionsFile` in sec. 5.5.1 (`rmsd` component)
- `refPositionsCol`: see definition of `refPositionsCol` in sec. 5.5.1 (`rmsd` component)

- `refPositionsColValue`: see definition of `refPositionsColValue` in sec. 5.5.1 (`rmsd` component)

- `closestToQuaternion` ⟨Reference rotation⟩
  Context: `orientation`
  Acceptable values: "`(q0, q1, q2, q3)`" quadruplet
  Default value: `(1.0, 0.0, 0.0, 0.0)` ("null" rotation)
  Description: Between the two equivalent quaternions (q_0, q_1, q_2, q_3) and $(-q_0, -q_1, -q_2, -q_3)$, the closer to `(1.0, 0.0, 0.0, 0.0)` is chosen. This simplifies the visualization of the colvar trajectory when sampled values are a smaller subset of all possible rotations. **Note:** *this only affects the output, never the dynamics*.

Tip: stopping the rotation of a protein. To stop the rotation of an elongated macromolecule in solution (and use an anisotropic box to save water molecules), it is possible to define a colvar with an `orientation` component, and restrain it throuh the `harmonic` bias around the identity rotation, `(1.0, 0.0, 0.0, 0.0)`. Only the overall orientation of the macromolecule is affected, and *not* its internal degrees of freedom. The user should also take care that the macromolecule is composed by a single chain, or disable `wrapAll` otherwise.

5.6.2 `orientationAngle`: angle of rotation from reference coordinates.

The block `orientationAngle` {...} accepts the same base options as the component `orientation`: `atoms`, `refPositions`, `refPositionsFile`, `refPositionsCol` and `refPositionsColValue`. The returned value is the angle of rotation θ between the current and the reference positions. This angle is expressed in degrees within the range [0°:180°].
List of keywords (see also 5.14 for additional options):

- `atoms`: see definition of `atoms` in sec. 5.5.1 (`rmsd` component)

- `refPositions`: see definition of `refPositions` in sec. 5.5.1 (`rmsd` component)

- `refPositionsFile`: see definition of `refPositionsFile` in sec. 5.5.1 (`rmsd` component)

- `refPositionsCol`: see definition of `refPositionsCol` in sec. 5.5.1 (`rmsd` component)

- `refPositionsColValue`: see definition of `refPositionsColValue` in sec. 5.5.1 (`rmsd` component)

5.6.3 `orientationProj`: cosine of the angle of rotation from reference coordinates.

The block `orientationProj` {...} accepts the same base options as the component `orientation`: `atoms`, `refPositions`, `refPositionsFile`, `refPositionsCol` and `refPositionsColValue`. The returned value is the cosine of the angle of rotation θ between the current and the reference positions. The range of values is [-1:1].
List of keywords (see also 5.14 for additional options):

- `atoms`: see definition of `atoms` in sec. 5.5.1 (`rmsd` component)

- `refPositions`: see definition of `refPositions` in sec. 5.5.1 (`rmsd` component)

- `refPositionsFile`: see definition of `refPositionsFile` in sec. 5.5.1 (`rmsd` component)
- `refPositionsCol`: see definition of `refPositionsCol` in sec. 5.5.1 (`rmsd` component)
- `refPositionsColValue`: see definition of `refPositionsColValue` in sec. 5.5.1 (`rmsd` component)

5.6.4 `spinAngle`: angle of rotation around a given axis.

The complete rotation described by `orientation` can optionally be decomposed into two sub-rotations: one is a "*spin*" rotation around **e**, and the other a "*tilt*" rotation around an axis orthogonal to **e**. The component `spinAngle` measures the angle of the "spin" sub-rotation around **e**.
List of keywords (see also 5.14 for additional options):

- `atoms`: see definition of `atoms` in sec. 5.5.1 (`rmsd` component)
- `refPositions`: see definition of `refPositions` in sec. 5.5.1 (`rmsd` component)
- `refPositionsFile`: see definition of `refPositionsFile` in sec. 5.5.1 (`rmsd` component)
- `refPositionsCol`: see definition of `refPositionsCol` in sec. 5.5.1 (`rmsd` component)
- `refPositionsColValue`: see definition of `refPositionsColValue` in sec. 5.5.1 (`rmsd` component)
- `axis` ⟨Special rotation axis (Å)⟩
  Context: `tilt`
  Acceptable values: `(x, y, z)` triplet
  Default value: `(0.0, 0.0, 1.0)`
  Description: The three components of this vector define (when normalized) the special rotation axis used to calculate the `tilt` and `spinAngle` components.

The component `spinAngle` returns an angle (in degrees) within the periodic interval $[-180 : 180]$.

Note: the value of `spinAngle` is a continuous function almost everywhere, with the exception of configurations with the corresponding "tilt" angle equal to 180° (i.e. the `tilt` component is equal to -1): in those cases, `spinAngle` is undefined. If such configurations are expected, consider defining a `tilt` colvar using the same axis **e**, and restraining it with a lower wall away from -1.

5.6.5 `tilt`: cosine of the rotation orthogonal to a given axis.

The component `tilt` measures the cosine of the angle of the "tilt" sub-rotation, which combined with the "spin" sub-rotation provides the complete rotation of a group of atoms. The cosine of the tilt angle rather than the tilt angle itself is implemented, because the latter is unevenly distributed even for an isotropic system: consider as an analogy the angle θ in the spherical coordinate system. The component `tilt` relies on the same options as `spinAngle`, including the definition of the axis **e**. The values of `tilt` are real numbers in the interval $[-1 : 1]$: the value 1 represents an orientation fully parallel to **e** (tilt angle = 0°), and the value -1 represents an anti-parallel orientation.
List of keywords (see also 5.14 for additional options):

- `atoms`: see definition of `atoms` in sec. 5.5.1 (`rmsd` component)
- `refPositions`: see definition of `refPositions` in sec. 5.5.1 (`rmsd` component)
- `refPositionsFile`: see definition of `refPositionsFile` in sec. 5.5.1 (`rmsd` component)
- `refPositionsCol`: see definition of `refPositionsCol` in sec. 5.5.1 (`rmsd` component)
- `refPositionsColValue`: see definition of `refPositionsColValue` in sec. 5.5.1 (`rmsd` component)
- `axis`: see definition of `axis` in sec. 5.6.4 (`spinAngle` component)

5.7 Protein structure descriptors

5.7.1 `alpha`: α-helix content of a protein segment.

The block `alpha` `{...}` defines the parameters to calculate the helical content of a segment of protein residues. The α-helical content across the $N+1$ residues N_0 to N_0+N is calculated by the formula:

$$\alpha\left(\mathrm{C}_\alpha^{(N_0)},\mathrm{O}^{(N_0)},\mathrm{C}_\alpha^{(N_0+1)},\mathrm{O}^{(N_0+1)},\ldots\mathrm{N}^{(N_0+5)},\mathrm{C}_\alpha^{(N_0+5)},\mathrm{O}^{(N_0+5)},\ldots\mathrm{N}^{(N_0+N)},\mathrm{C}_\alpha^{(N_0+N)}\right) = \tag{10}$$

$$\frac{1}{2(N-2)}\sum_{n=N_0}^{N_0+N-2}\mathrm{angf}\left(\mathrm{C}_\alpha^{(n)},\mathrm{C}_\alpha^{(n+1)},\mathrm{C}_\alpha^{(n+2)}\right) + \frac{1}{2(N-4)}\sum_{n=N_0}^{N_0+N-4}\mathrm{hbf}\left(\mathrm{O}^{(n)},\mathrm{N}^{(n+4)}\right), \tag{11}$$

where the score function for the $\mathrm{C}_\alpha-\mathrm{C}_\alpha-\mathrm{C}_\alpha$ angle is defined as:

$$\mathrm{angf}\left(\mathrm{C}_\alpha^{(n)},\mathrm{C}_\alpha^{(n+1)},\mathrm{C}_\alpha^{(n+2)}\right) = \frac{1-\left(\theta(\mathrm{C}_\alpha^{(n)},\mathrm{C}_\alpha^{(n+1)},\mathrm{C}_\alpha^{(n+2)})-\theta_0\right)^2/(\Delta\theta_{\mathrm{tol}})^2}{1-\left(\theta(\mathrm{C}_\alpha^{(n)},\mathrm{C}_\alpha^{(n+1)},\mathrm{C}_\alpha^{(n+2)})-\theta_0\right)^4/(\Delta\theta_{\mathrm{tol}})^4}, \tag{12}$$

and the score function for the $\mathrm{O}^{(n)} \leftrightarrow \mathrm{N}^{(n+4)}$ hydrogen bond is defined through a `hBond` colvar component on the same atoms.

List of keywords (see also 5.14 for additional options):

- `residueRange` ⟨Potential α-helical residues⟩
  Context: `alpha`
  Acceptable values: "<Initial residue number>-<Final residue number>"
  Description: This option specifies the range of residues on which this component should be defined. The Colvars module looks for the atoms within these residues named "`CA`", "`N`" and "`O`", and raises an error if any of those atoms is not found.
- `psfSegID` ⟨PSF segment identifier⟩
  Context: `alpha`
  Acceptable values: string (max 4 characters)
  Description: This option sets the PSF segment identifier for the residues specified in `residueRange`. This option is only required when PSF topologies are used.

- `hBondCoeff` ⟨Coefficient for the hydrogen bond term⟩
  Context: `alpha`
  Acceptable values: positive between 0 and 1
  Default value: 0.5
  Description: This number specifies the contribution to the total value from the hydrogen bond terms. 0 disables the hydrogen bond terms, 1 disables the angle terms.

- `angleRef` ⟨Reference $C_\alpha - C_\alpha - C_\alpha$ angle⟩
  Context: `alpha`
  Acceptable values: positive decimal
  Default value: 88°
  Description: This option sets the reference angle used in the score function (12).

- `angleTol` ⟨Tolerance in the $C_\alpha - C_\alpha - C_\alpha$ angle⟩
  Context: `alpha`
  Acceptable values: positive decimal
  Default value: 15°
  Description: This option sets the angle tolerance used in the score function (12).

- `hBondCutoff` ⟨Hydrogen bond cutoff⟩
  Context: `alpha`
  Acceptable values: positive decimal
  Default value: 3.3 Å
  Description: Equivalent to the `cutoff` option in the `hBond` component.

- `hBondExpNumer` ⟨Hydrogen bond numerator exponent⟩
  Context: `alpha`
  Acceptable values: positive integer
  Default value: 6
  Description: Equivalent to the `expNumer` option in the `hBond` component.

- `hBondExpDenom` ⟨Hydrogen bond denominator exponent⟩
  Context: `alpha`
  Acceptable values: positive integer
  Default value: 8
  Description: Equivalent to the `expDenom` option in the `hBond` component.

This component returns positive values, always comprised between 0 (lowest α-helical score) and 1 (highest α-helical score).

5.7.2 `dihedralPC`: protein dihedral pricipal component

The block `dihedralPC` {...} defines the parameters to calculate the projection of backbone dihedral angles within a protein segment onto a *dihedral principal component*, following the formalism of dihedral principal component analysis (dPCA) proposed by Mu et al.[4] and documented in detail by Altis et al.[5]. Given a peptide or protein segment of N residues, each with Ramachandran angles ϕ_i and ψ_i, dPCA rests on a variance/covariance analysis of the $4(N-1)$ variables $\cos(\psi_1), \sin(\psi_1), \cos(\phi_2), \sin(\phi_2) \cdots \cos(\phi_N), \sin(\phi_N)$. Note that angles ϕ_1 and ψ_N have little impact on chain conformation, and are therefore discarded, following the implementation of dPCA in the analysis software Carma.[6]

For a given principal component (eigenvector) of coefficients $(k_i)_{1\leq i\leq 4(N-1)}$, the projection of the current backbone conformation is:

$$\xi = \sum_{n=1}^{N-1} k_{4n-3}\cos(\psi_n) + k_{4n-2}\sin(\psi_n) + k_{4n-1}\cos(\phi_{n+1}) + k_{4n}\sin(\phi_{n+1}) \tag{13}$$

`dihedralPC` expects the same parameters as the `alpha` component for defining the relevant residues (`residueRange` and `psfSegID`) in addition to the following:
List of keywords (see also 5.14 for additional options):

- `residueRange`: see definition of `residueRange` in sec. 5.7.1 (`alpha` component)
- `psfSegID`: see definition of `psfSegID` in sec. 5.7.1 (`alpha` component)
- `vectorFile` ⟨File containing dihedral PCA eigenvector(s)⟩
  Context: `dihedralPC`
  Acceptable values: file name
  Description: A text file containing the coefficients of dihedral PCA eigenvectors on the cosine and sine coordinates. The vectors should be arranged in columns, as in the files output by Carma.[6]
- `vectorNumber` ⟨File containing dihedralPCA eigenvector(s)⟩
  Context: `dihedralPC`
  Acceptable values: positive integer
  Description: Number of the eigenvector to be used for this component.

5.8 Raw data: building blocks for custom functions

5.8.1 `cartesian`: vector of atomic Cartesian coordinates.

The `cartesian` {...} block defines a component returning a flat vector containing the Cartesian coordinates of all participating atoms, in the order $(x_1, y_1, z_1, \cdots, x_n, y_n, z_n)$.
List of keywords (see also 5.14 for additional options):

- `atoms` ⟨Group of atoms⟩
  Context: `cartesian`
  Acceptable values: Block `atoms` {...}
  Description: Defines the atoms whose coordinates make up the value of the component. If `rotateReference` or `centerReference` are defined, coordinates are evaluated within the moving frame of reference.

5.8.2 `distancePairs`: set of pairwise distances between two groups.

The `distancePairs` {...} block defines a $N_1 \times N_2$-dimensional variable that includes all mutual distances between the atoms of two groups. This can be useful, for example, to develop a new variable defined over two groups, by using the `scriptedFunction` feature.
List of keywords (see also 5.14 for additional options):

- `group1`: see definition of `group1` in sec. 5.2.1 (`distance` component)
- `group2`: analogous to `group1`
- `forceNoPBC`: see definition of `forceNoPBC` in sec. 5.2.1 (`distance` component)

This component returns a $N_1 \times N_2$-dimensional vector of numbers, each ranging from 0 to the largest possible distance within the chosen boundary conditions.

5.9 Geometric path collective variables

The geometric path collective variables define the progress along a path, s, and the distance from the path, z. These CVs are proposed by Leines and Ensing[7] , which differ from that[8] proposed by Branduardi et al., and utilize a set of geometric algorithms. The path is defined as a series of frames in the atomic Cartesian coordinate space or the CV space. s and z are computed as

$$s = \frac{m}{M} \pm \frac{1}{2M}\left(\frac{\sqrt{(\mathbf{v}_1\cdot\mathbf{v}_3)^2 - |\mathbf{v}_3|^2(|\mathbf{v}_1|^2 - |\mathbf{v}_2|^2)} - (\mathbf{v}_1\cdot\mathbf{v}_3)}{|\mathbf{v}_3|^2} - 1\right) \tag{14}$$

$$z = \sqrt{\left(\mathbf{v}_1 + \frac{1}{2}\left(\frac{\sqrt{(\mathbf{v}_1\cdot\mathbf{v}_3)^2 - |\mathbf{v}_3|^2(|\mathbf{v}_1|^2 - |\mathbf{v}_2|^2)} - (\mathbf{v}_1\cdot\mathbf{v}_3)}{|\mathbf{v}_3|^2} - 1\right)\mathbf{v}_4\right)^2} \tag{15}$$

where $\mathbf{v}_1 = \mathbf{s}_m - \mathbf{z}$ is the vector connecting the current position to the closest frame, $\mathbf{v}_2 = \mathbf{z} - \mathbf{s}_{m-1}$ is the vector connecting the second closest frame to the current position, $\mathbf{v}_3 = \mathbf{s}_{m+1} - \mathbf{s}_m$ is the vector connecting the closest frame to the third closest frame, and $\mathbf{v}_4 = \mathbf{s}_m - \mathbf{s}_{m-1}$ is the vector connecting the second closest frame to the closest frame. m and M are the current index of the closest frame and the total number of frames, respectively. If the current position is on the left of the closest reference frame, the $\pm$ in s turns to the positive sign. Otherwise it turns to the negative sign.

The equations above assume: (i) the frames are equidistant and (ii) the second and the third closest frames are neighbouring to the closest frame. When these assumptions are not satisfied, this set of path CV should be used carefully.

5.9.1 `gspath`: progress along a path defined in atomic Cartesian coordinate space.

In the `gspath` {...} and the `gzpath` {...} block all vectors, namely $\mathbf{z}$ and $\mathbf{s}_k$ are defined in atomic Cartesian coordinate space. More specifically, $\mathbf{z} = [\mathbf{r}_1, \cdots, \mathbf{r}_n]$, where $\mathbf{r}_i$ is the i-th atom specified in the `atoms` block. $\mathbf{s}_k = [\mathbf{r}_{k,1}, \cdots, \mathbf{r}_{k,n}]$, where $\mathbf{r}_{k,i}$ means the i-th atom of the k-th reference frame.
List of keywords (see also 5.14 for additional options):

- `atoms` ⟨Group of atoms⟩
  Context: `gspath` and `gzpath`
  Acceptable values: Block `atoms` {...}
  Description: Defines the atoms whose coordinates make up the value of the component.
- `refPositionsCol` ⟨PDB column containing atom flags⟩
  Context: `gspath` and `gzpath`
  Acceptable values: `O`, `B`, `X`, `Y`, or `Z`
  Description: If `refPositionsFileN` is a PDB file that contains all the atoms in the topology, this option may be provided to set which PDB field is used to flag the reference coordinates for `atoms`.

- `refPositionsFileN` ⟨File containing the reference positions for fitting⟩
  Context: `gspath` and `gzpath`
  Acceptable values: UNIX filename
  Description: The path is defined by multiple `refPositionsFiles` which are similiar to `refPositionsFile` in the `rmsd` CV. If your path consists of 10 nodes, you can list the coordinate file (in PDB or XYZ format) from `refPositionsFile1` to `refPositionsFile10`.

- `useSecondClosestFrame` ⟨Define $\mathbf{s}_{m-1}$ as the second closest frame?⟩
  Context: `gspath` and `gzpath`
  Acceptable values: boolean
  Default value: `on`
  Description: The definition assumes the second closest frame is neighbouring to the closest frame. This is not always true especially when the path is crooked. If this option is set to `on` (default), $\mathbf{s}_{m-1}$ is defined as the second closest frame. If this option is set to `off`, $\mathbf{s}_{m-1}$ is defined as the left or right neighbouring frame of the closest frame.

- `useThirdClosestFrame` ⟨Define $\mathbf{s}_{m+1}$ as the third closest frame?⟩
  Context: `gspath` and `gzpath`
  Acceptable values: boolean
  Default value: `off`
  Description: The definition assumes the third closest frame is neighbouring to the closest frame. This is not always true especially when the path is crooked. If this option is set to `on`, $\mathbf{s}_{m+1}$ is defined as the third closest frame. If this option is set to `off` (default), $\mathbf{s}_{m+1}$ is defined as the left or right neighbouring frame of the closest frame.

- `fittingAtoms` ⟨The atoms that are used for alignment⟩
  Context: `gspath` and `gspath`
  Acceptable values: Group of atoms
  Description: Before calculating $\mathbf{v}_1$, $\mathbf{v}_2$, $\mathbf{v}_3$ and $\mathbf{v}_4$, the current frame need to be aligned to the corresponding reference frames. This option specifies which atoms are used to do alignment.

5.9.2 `gzpath`: distance from a path defined in atomic Cartesian coordinate space.

List of keywords (see also 5.14 for additional options):

- `useZsquare` ⟨Compute z^2 instead of z⟩
  Context: `gzpath`
  Acceptable values: boolean
  Default value: `off`
  Description: z is not differentiable when it is zero. This implementation workarounds it by setting the derivative of z to zero when $z = 0$. Another workaround is set this option to `on`, which computes z^2 instead of z, and then z^2 is differentiable when it is zero.

The usage of `gzpath` and `gspath` is illustrated below:

```
colvar {
  # Progress along the path
  name gs
```

```
  # The path is defined by 5 reference frames (from string-00.pdb to string-04.pdb)
  # Use atomic coordinate from atoms 1, 2 and 3 to compute the path
  gspath {
    atoms {atomnumbers { 1 2 3 }}
    refPositionsFile1 string-00.pdb
    refPositionsFile2 string-01.pdb
    refPositionsFile3 string-02.pdb
    refPositionsFile4 string-03.pdb
    refPositionsFile5 string-04.pdb
  }
}
colvar {
  # Distance from the path
  name gz
  # The path is defined by 5 reference frames (from string-00.pdb to string-04.pdb)
  # Use atomic coordinate from atoms 1, 2 and 3 to compute the path
  gzpath {
    atoms {atomnumbers { 1 2 3 }}
    refPositionsFile1 string-00.pdb
    refPositionsFile2 string-01.pdb
    refPositionsFile3 string-02.pdb
    refPositionsFile4 string-03.pdb
    refPositionsFile5 string-04.pdb
  }
}
```

5.9.3 `linearCombination`: Helper CV to define a linear combination of other CVs

This is a helper CV which can be defined as a linear combination of other CVs. It maybe useful when you want to define the `gspathCV { ... }` and the `gzpathCV { ... }` as combinations of other CVs.

5.9.4 `gspathCV`: progress along a path defined in CV space.

In the `gspathCV { ... }` and the `gzpathCV { ... }` block all vectors, namely $\mathbf{z}$ and $\mathbf{s}_k$ are defined in CV space. More specifically, $\mathbf{z} = [\xi_1, \cdots, \xi_n]$, where ξ_i is the i-th CV. $\mathbf{s}_k = [\xi_{k,1}, \cdots, \xi_{k,n}]$, where $\xi_{k,i}$ means the i-th CV of the k-th reference frame. It should be note that these two CVs requires the `pathFile` option, which specifies a path file. Each line in the path file contains a set of space-seperated CV value of the reference frame. The sequence of reference frames matches the sequence of the lines.
List of keywords (see also 5.14 for additional options):

- `useSecondClosestFrame` ⟨Define $\mathbf{s}_{m-1}$ as the second closest frame?⟩
  Context: `gspathCV` and `gzpathCV`
  Acceptable values: boolean
  Default value: `on`
  Description: The definition assumes the second closest frame is neighbouring to the closest frame. This is not always true especially when the path is crooked. If this option is set to `on` (default), $\mathbf{s}_{m-1}$

is defined as the second closest frame. If this option is set to `off`, $\mathbf{s}_{m-1}$ is defined as the left or right neighbouring frame of the closest frame.

- `useThirdClosestFrame` ⟨Define $\mathbf{s}_{m+1}$ as the third closest frame?⟩
  Context: `gspathCV` and `gzpathCV`
  Acceptable values: boolean
  Default value: `off`
  Description: The definition assumes the third closest frame is neighbouring to the closest frame. This is not always true especially when the path is crooked. If this option is set to `on`, $\mathbf{s}_{m+1}$ is defined as the third closest frame. If this option is set to `off` (default), $\mathbf{s}_{m+1}$ is defined as the left or right neighbouring frame of the closest frame.

- `pathFile` ⟨The file name of the path file.⟩
  Context: `gspathCV` and `gzpathCV`
  Acceptable values: UNIX filename
  Description: Defines the nodes or images that constitutes the path in CV space. The CVs of an image are listed in a line of `pathFile` using space-seperated format. Lines from top to button in `pathFile` corresponds images from initial to last.

5.9.5 `gzpathCV`: distance from a path defined in CV space.

List of keywords (see also 5.14 for additional options):

- `useZsquare` ⟨Compute z^2 instead of z⟩
  Context: `gzpathCV`
  Acceptable values: boolean
  Default value: `off`
  Description: z is not differentiable when it is zero. This implementation workarounds it by setting the derivative of z to zero when $z = 0$. Another workaround is set this option to `on`, which computes z^2 instead of z, and then z^2 is differentiable when it is zero.

The usage of `gzpathCV` and `gspathCV` is illustrated below:

```
colvar {
  # Progress along the path
  name gs
  # Path defined by the CV space of two dihedral angles
  gspathCV {
    pathFile ./path.txt
    dihedral {
      name 001
      group1 {atomNumbers {5}}
      group2 {atomNumbers {7}}
      group3 {atomNumbers {9}}
      group4 {atomNumbers {15}}
    }
    dihedral {
      name 002
```

```
      group1 {atomNumbers {7}}
      group2 {atomNumbers {9}}
      group3 {atomNumbers {15}}
      group4 {atomNumbers {17}}
    }
  }
}
colvar {
  # Distance from the path
  name gz
  gzpathCV {
    pathFile ./path.txt
    dihedral {
      name 001
      group1 {atomNumbers {5}}
      group2 {atomNumbers {7}}
      group3 {atomNumbers {9}}
      group4 {atomNumbers {15}}
    }
    dihedral {
      name 002
      group1 {atomNumbers {7}}
      group2 {atomNumbers {9}}
      group3 {atomNumbers {15}}
      group4 {atomNumbers {17}}
    }
  }
}
```

5.10 Arithmetic path collective variables

The arithmetic path collective variable in CV space uses the same formula as the one proposed by Branduardi[8] et al., except that it computes s and z in CV space instead of RMSDs in Cartesian space. Moreover, this implementation allows different coefficients for each CV components as described in [9]. Assuming a path is composed of N reference frames and defined in an M-dimensional CV space, then the equations of s and z of the path are

$$s = \frac{\sum_{i=1}^{N} i \exp\left(-\lambda \sum_{j=1}^{M} c_j^2 \left(x_j - x_{i,j}\right)^2\right)}{\sum_{i=1}^{N} \exp\left(-\lambda \sum_{j=1}^{M} c_j^2 \left(x_j - x_{i,j}\right)^2\right)} \tag{16}$$

$$z = -\frac{1}{\lambda} \ln\left(\sum_{i=1}^{N} \exp\left(-\lambda \sum_{j=1}^{M} c_j^2 \left(x_j - x_{i,j}\right)\right)\right) \tag{17}$$

where c_j is the coefficient(weight) of the j-th CV, $x_{i,j}$ is the value of j-th CV of i-th reference frame and x_j is the value of j-th CV of current frame. λ is a parameter to smooth the variation of s and z.

5.10.1 `aspathCV`: progress along a path defined in CV space.

This colvar component computes the s variable.
List of keywords (see also 5.14 for additional options):

- `weights` ⟨Coefficients of the collective variables⟩
  Context: `aspathCV` and `azpathCV`
  Acceptable values: space-seperated numbers in a {...} block
  Default value: {`1.0 ...`}
  Description: Define the coefficients. The j-th value in the {...} block corresponds the c_j in the equations.

- `lambda` ⟨Smoothness of the variation of s and z⟩
  Context: `aspathCV` and `azpathCV`
  Acceptable values: decimal
  Default value: inverse of the mean square displacements of successive reference frames
  Description: The value of λ in the equations.

- `pathFile` ⟨The file name of the path file.⟩
  Context: `aspathCV` and `azpathCV`
  Acceptable values: UNIX filename
  Description: Defines the nodes or images that constitutes the path in CV space. The CVs of an image are listed in a line of `pathFile` using space-seperated format. Lines from top to button in `pathFile` corresponds images from initial to last.

5.10.2 `azpathCV`: distance from a path defined in CV space.

This colvar component computes the z variable. Options are the same as in 5.10.1.

The usage of `azpathCV` and `aspathCV` is illustrated below:

```
colvar {
  # Progress along the path
  name as
  # Path defined by the CV space of two dihedral angles
  aspathCV {
    pathFile ./path.txt
    weights {1.0 1.0}
    lambda 0.005
    dihedral {
      name 001
      group1 {atomNumbers {5}}
      group2 {atomNumbers {7}}
      group3 {atomNumbers {9}}
      group4 {atomNumbers {15}}
    }
    dihedral {
      name 002
      group1 {atomNumbers {7}}
```

```
      group2 {atomNumbers {9}}
      group3 {atomNumbers {15}}
      group4 {atomNumbers {17}}
    }
  }
}
colvar {
  # Distance from the path
  name az
  azpathCV {
    pathFile ./path.txt
    weights {1.0 1.0}
    lambda 0.005
    dihedral {
      name 001
      group1 {atomNumbers {5}}
      group2 {atomNumbers {7}}
      group3 {atomNumbers {9}}
      group4 {atomNumbers {15}}
    }
    dihedral {
      name 002
      group1 {atomNumbers {7}}
      group2 {atomNumbers {9}}
      group3 {atomNumbers {15}}
      group4 {atomNumbers {17}}
    }
  }
}
```

5.10.3 Path collective variables in Cartesian coordinates

The path collective variables defined by Branduardi et al. [8] are based on RMSDs in Cartesian coordinates. Noting d_i the RMSD between the current set of Cartesian coordinates and those of image number i of the path:

$$s = \frac{1}{N-1} \frac{\sum_{i=1}^{N} (i-1) \exp\left(-\lambda d_i^2\right)}{\sum_{i=1}^{N} \exp\left(-\lambda d_i^2\right)} \tag{18}$$

$$z = -\frac{1}{\lambda} \ln \left(\sum_{i=1}^{N} \exp(-\lambda d_i^2) \right) \tag{19}$$

where λ is the smoothing parameter.

These coordinates are implemented as Tcl-scripted combinations of `rmsd` components. The implementation is available as file `colvartools/pathCV.tcl`, and an example is provided in file `examples/10_pathCV.namd` of the Colvars public repository. It implements an optimization procedure, whereby the distance to a given image is only calculated if its contribution to the sum is larger than a user-defined `tolerance` parameter. All distances are calculated every `freq` timesteps to update the list of nearby images.

5.11 Shared keywords for all components

The following options can be used for any of the above colvar components in order to obtain a polynomial combination or any user-supplied function provided by `scriptedFunction`.

- `name` ⟨Name of this component⟩
  Context: any component
  Acceptable values: string
  Default value: type of component + numeric id
  Description: The name is an unique case-sensitive string which allows the Colvars module to identify this component. This is useful, for example, when combining multiple components via a `scriptedFunction`.

- `scalable` ⟨Attempt to calculate this component in parallel?⟩
  Context: any component
  Acceptable values: boolean
  Default value: `on`, if available
  Description: If set to `on` (default), the Colvars module will attempt to calculate this component in parallel to reduce overhead. Whether this option is available depends on the type of component: currently supported are `distance`, `distanceZ`, `distanceXY`, `distanceVec`, `distanceDir`, `angle` and `dihedral`. This flag influences computational cost, but does not affect numerical results: therefore, it should only be turned off for debugging or testing purposes.

5.12 Periodic components

The following components returns real numbers that lie in a periodic interval:

- `dihedral`: torsional angle between four groups;
- `spinAngle`: angle of rotation around a predefined axis in the best-fit from a set of reference coordinates.

In certain conditions, `distanceZ` can also be periodic, namely when periodic boundary conditions (PBCs) are defined in the simulation and `distanceZ`'s axis is parallel to a unit cell vector.

In addition, a custom or scripted scalar colvar may be periodic depending on its user-defined expression. It will only be treated as such by the Colvars module if the period is specified using the `period` keyword, while `wrapAround` is optional.

The following keywords can be used within periodic components, or wthin scripted variables 5.15).

- `period` ⟨Period of the component⟩
  Context: `distanceZ`, custom colvars
  Acceptable values: positive decimal
  Default value: 0.0
  Description: Setting this number enables the treatment of `distanceZ` as a periodic component: by default, `distanceZ` is not considered periodic. The keyword is supported, but irrelevant within `dihedral` or `spinAngle`, because their period is always 360 degrees.

- `wrapAround` ⟨Center of the wrapping interval for periodic variables⟩
  Context: `distanceZ`, `dihedral`, `spinAngle`, custom colvars
  Acceptable values: decimal
  Default value: 0.0
  Description: By default, values of the periodic components are centered around zero, ranging from $-P/2$ to $P/2$, where P is the period. Setting this number centers the interval around this value. This can be useful for convenience of output, or to set the walls for a `harmonicWalls` in an order that would not otherwise be allowed.

Internally, all differences between two values of a periodic colvar follow the minimum image convention: they are calculated based on the two periodic images that are closest to each other.

Note: linear or polynomial combinations of periodic components (see 5.14) may become meaningless when components cross the periodic boundary. Use such combinations carefully: estimate the range of possible values of each component in a given simulation, and make use of `wrapAround` *to limit this problem whenever possible.*

5.13 Non-scalar components

When one of the following components are used, the defined colvar returns a value that is not a scalar number:

- `distanceVec`: 3-dimensional vector of the distance between two groups;
- `distanceDir`: 3-dimensional unit vector of the distance between two groups;
- `orientation`: 4-dimensional unit quaternion representing the best-fit rotation from a set of reference coordinates.

The distance between two 3-dimensional unit vectors is computed as the angle between them. The distance between two quaternions is computed as the angle between the two 4-dimensional unit vectors: because the orientation represented by `q` is the same as the one represented by $-$`q`, distances between two quaternions are computed considering the closest of the two symmetric images.

Non-scalar components carry the following restrictions:

- Calculation of total forces (`outputTotalForce` option) is currently not implemented.
- Each colvar can only contain one non-scalar component.
- Binning on a grid (`abf`, `histogram` and `metadynamics` with `useGrids` enabled) is currently not implemented for colvars based on such components.

Note: while these restrictions apply to individual colvars based on non-scalar components, no limit is set to the number of scalar colvars. To compute multi-dimensional histograms and PMFs, use sets of scalar colvars of arbitrary size.

5.13.1 Calculating total forces

In addition to the restrictions due to the type of value computed (scalar or non-scalar), a final restriction can arise when calculating total force (`outputTotalForce` option or application of a `abf` bias). total forces are available currently only for the following components: `distance`, `distanceZ`, `distanceXY`, `angle`, `dihedral`, `rmsd`, `eigenvector` and `gyration`.

5.14 Linear and polynomial combinations of components

To extend the set of possible definitions of colvars $\xi(\mathbf{r})$, multiple components $q_i(\mathbf{r})$ can be summed with the formula:

$$\xi(\mathbf{r}) = \sum_i c_i [q_i(\mathbf{r})]^{n_i} \tag{20}$$

where each component appears with a unique coefficient c_i (1.0 by default) the positive integer exponent n_i (1 by default).

Any set of components can be combined within a colvar, provided that they return the same type of values (scalar, unit vector, vector, or quaternion). By default, the colvar is the sum of its components. Linear or polynomial combinations (following equation (20)) can be obtained by setting the following parameters, which are common to all components:

- `componentCoeff` ⟨Coefficient of this component in the colvar⟩
  Context: any component
  Acceptable values: decimal
  Default value: `1.0`
  Description: Defines the coefficient by which this component is multiplied (after being raised to `componentExp`) before being added to the sum.

- `componentExp` ⟨Exponent of this component in the colvar⟩
  Context: any component
  Acceptable values: integer
  Default value: `1`
  Description: Defines the power at which the value of this component is raised before being added to the sum. When this exponent is different than 1 (non-linear sum), total forces and the Jacobian force are not available, making the colvar unsuitable for ABF calculations.

Example: To define the *average* of a colvar across different parts of the system, simply define within the same colvar block a series of components of the same type (applied to different atom groups), and assign to each component a `componentCoeff` of $1/N$.

5.15 Scripted functions

When scripting is supported (default in VMD), a colvar may be defined as a scripted function of its components, rather than a linear or polynomial combination. When implementing generic functions of Cartesian coordinates rather than functions of existing components, the `cartesian` component may be particularly useful. A scalar-valued scripted variable may be manually defined as periodic by providing the keyword `period`, and the optional keyword `wrapAround`, with the same meaning as in periodic components (see 5.12 for details).

An example of elaborate scripted colvar is given in example 10, in the form of path-based collective variables as defined by Branduardi et al[8] (Section 5.10.3).

- `scriptedFunction` ⟨Compute colvar as a scripted function of its components⟩
  Context: `colvar`
  Acceptable values: string
  Description: If this option is specified, the colvar will be computed as a scripted function of the values of its components. To that effect, the user should define two Tcl procedures: `calc_<scriptedFunction>` and `calc_<scriptedFunction>_gradient`, both accepting as many parameters as the colvar has

components. Values of the components will be passed to those procedures in the order defined by their sorted `name` strings. Note that if all components are of the same type, their default names are sorted in the order in which they are defined, so that names need only be specified for combinations of components of different types. `calc_<scriptedFunction>` should return one value of type <scriptedFunctionType>, corresponding to the colvar value. `calc_<scriptedFunction>_gradient` should return a Tcl list containing the derivatives of the function with respect to each component. If both the function and some of the components are vectors, the gradient is really a Jacobian matrix that should be passed as a linear vector in row-major order, i.e. for a function $f_i(x_j)$: $\nabla_x f_1 \nabla_x f_2 \cdots$.

- `scriptedFunctionType` ⟨Type of value returned by the scripted colvar⟩
  Context: `colvar`
  Acceptable values: string
  Default value: `scalar`
  Description: If a colvar is defined as a scripted function, its type is not constrained by the types of its components. With this flag, the user may specify whether the colvar is a scalar or one of the following vector types: `vector3` (a 3D vector), `unit_vector3` (a normalized 3D vector), or `unit_quaternion` (a normalized quaternion), or `vector` (a vector whose size is specified by `scriptedFunctionVectorSize`). Non-scalar values should be passed as space-separated lists.

- `scriptedFunctionVectorSize` ⟨Dimension of the vector value of a scripted colvar⟩
  Context: `colvar`
  Acceptable values: positive integer
  Description: This parameter is only valid when `scriptedFunctionType` is set to `vector`. It defines the vector length of the colvar value returned by the function.

5.16 Defining grid parameters

Many algorithms require the definition of boundaries and/or characteristic spacings that can be used to define discrete "states" in the collective variable, or to combine variables with very different units. The parameters described below offer a way to specify these parameters only once for each variable, while using them multiple times in restraints, time-dependent biases or analysis methods.

- `width` ⟨Unit of the variable, or grid spacing⟩
  Context: `colvar`
  Acceptable values: positive decimal
  Default value: 1.0
  Description: This number defines the effective unit of measurement for the collective variable, and is used by the biasing methods for the following purposes. Harmonic (7.5), harmonic walls (7.7) and linear restraints (7.8) use it to set the physical unit of the force constant, which is useful for multidimensional restraints involving multiple variables with very different units (for examples, $\AA$ or degrees $^\circ$) with a single, scaled force constant. The values of the scaled force constant in the units of each variable are printed at initialization time. Histograms (7.10), ABF (7.2) and metadynamics (7.4) all use this number as the initial choice for the grid spacing along this variable: for this reason, `width` should generally be no larger than the standard deviation of the colvar in an unbiased simulation. Unless it is required to control the spacing, it is usually simplest to keep the default value of 1, so that restraint force constants are provided with their full physical unit.

- `lowerBoundary` ⟨Lower boundary of the colvar⟩
  Context: `colvar`
  Acceptable values: decimal
  Default value: natural boundary of the function
  Description: Defines the lowest end of the interval of "relevant" values for the variable. This number can be, for example, a true physical boundary imposed by the choice of function (e.g. the `distance` function is always larger than zero): if this is the case, and only one function is used to define the variable, the default value of this number is set to the lowest end of the range of values of that function, if available (see Section 5.1). Alternatively, this value may be provided by the user, to represent for example the left-most point of a PMF calculation along this variable. In the latter case, it is the user's responsibility to either *(a)* ensure the variable does not go significantly beyond the boundary (for example by adding a `harmonicWalls` restraint, 7.7), or *(b)* instruct the code that this is a true physical boundary by setting `hardLowerBoundary`.

- `upperBoundary` ⟨Upper boundary of the colvar⟩
  Context: `colvar`
  Acceptable values: decimal
  Default value: natural boundary of the function
  Description: Similarly to `lowerBoundary`, defines the highest of the "relevant" values of the variable.

- `hardLowerBoundary` ⟨Whether the lower boundary is the physical lower limit⟩
  Context: `colvar`
  Acceptable values: boolean
  Default value: provided by the component
  Description: When the colvar has a "natural" boundary (for example, a `distance` colvar cannot go below 0) this flag is automatically enabled. For more complex variable definitions, or when `lowerBoundary` is provided directly by the user, it may be useful to set this flag explicitly. This option does not affect simulation results, but enables some internal optimizations by letting the code know that the variable is unable to cross the lower boundary, regardless of whether restraints are applied to it.

- `hardUpperBoundary` ⟨Whether the upper boundary is the physical upper limit of the colvar's values⟩
  Context: `colvar`
  Acceptable values: boolean
  Default value: provided by the component
  Description: Analogous to `hardLowerBoundary`.

- `expandBoundaries` ⟨Allow to expand the two boundaries if needed⟩
  Context: `colvar`
  Acceptable values: boolean
  Default value: `off`
  Description: If defined, `lowerBoundary` and `upperBoundary` may be automatically expanded to accommodate colvar values that do not fit in the initial range. Currently, this option is used by the metadynamics bias (7.4) to keep all of its hills fully within the grid. This option cannot be used when the initial boundaries already span the full period of a periodic colvar.

5.16.1 Grid files: multicolumn text format

Many simulation methods and analysis tools write files that contain functions of the collective variables tabulated on a grid (e.g. potentials of mean force or multidimentional histograms) for the purpose of analyzing results. Such files are produced by ABF (7.2), metadynamics (7.4), multidimensional histograms (7.10), as well as any restraint with optional thermodynamic integration support (7.1).

In some cases, these files may also be read as input of a new simulation. Suitable input files for this purpose are typically generated as output files of previous simulations, or directly by the user in the specific case of ensemble-biased metadynamics (7.4.5). This section explains the "multicolumn" format used by these files. For a multidimensional function $f(\xi_1, \xi_2, \dots)$ the multicolumn grid format is defined as follows:

```
#  N_cv
#  min(ξ_1)      width(ξ_1)      npoints(ξ_1)      periodic(ξ_1)
#  min(ξ_2)      width(ξ_2)      npoints(ξ_2)      periodic(ξ_2)
#  ...           ...             ...               ...
#  min(ξ_Ncv)    width(ξ_Ncv)    npoints(ξ_Ncv)    periodic(ξ_Ncv)

   ξ_1^1         ξ_2^1           ...               ξ_Ncv^1           f(ξ_1^1, ξ_2^1, ..., ξ_Ncv^1)
   ξ_1^1         ξ_2^1           ...               ξ_Ncv^2           f(ξ_1^1, ξ_2^1, ..., ξ_Ncv^2)
   ...           ...             ...               ...               ...
```

Lines beginning with the character "#" are the header of the file. N_{cv} is the number of collective variables sampled by the grid. For each variable ξ_i, $\mathtt{min}(\xi_i)$ is the lowest value sampled by the grid (i.e. the left-most boundary of the grid along ξ_i), $\mathtt{width}(\xi_i)$ is the width of each grid step along ξ_i, $\mathtt{npoints}(\xi_i)$ is the number of points and $\mathtt{periodic}(\xi_i)$ is a flag whose value is 1 or 0 depending on whether the grid is periodic along ξ_i. In most situations:

- $\mathtt{min}(\xi_i)$ is given by the `lowerBoundary` keyword of the variable ξ_i;
- $\mathtt{width}(\xi_i)$ is given by the `width` keyword;
- $\mathtt{npoints}(\xi_i)$ is calculated from the two above numbers and the `upperBoundary` keyword;
- $\mathtt{periodic}(\xi_i)$ is set to 1 if and only if ξ_i is periodic and the grids' boundaries cover its period.

Exception: there is a slightly different header in PMF files computed by ABF (7.2) or by other biases with an optional thermodynamic integration (TI) estimator (7.1). In this case, free-energy gradients are accumulated on an `(npoints)`-long grid along each variable ξ: after these gradients are integrated, the resulting PMF is discretized on a grid with `(npoints+1)` points along ξ. Therefore, the edges of the PMF's grid extend $\mathtt{width}/2$ above and below the original boundaries (unless these are periodic). The format of the file's header is otherwise unchanged.

After the header, the rest of the file contains values of the tabulated function $f(\xi_1, \xi_2, \dots \xi_{N_{\mathrm{cv}}})$, one for each line. The first N_{cv} columns contain values of $\xi_1, \xi_2, \dots \xi_{N_{\mathrm{cv}}}$ and the last column contains the value of the function f. Points are sorted in ascending order with the fastest-changing values at the right ("C-style" order). Each sweep of the right-most variable $\xi_{N_{\mathrm{cv}}}$ is terminated by an empty line. For two dimensional grid files, this allows quick visualization by programs such as GNUplot.

Example 1: multicolumn text file for a one-dimensional histogram with `lowerBoundary` = 15, `upperBoundary` = 48 and `width` = 0.1.

```
#  1
#  15     0.1          330  0

   15.05  6.14012e-07
   15.15  7.47644e-07
   ...    ...
   47.85  1.65944e-06
   47.95  1.46712e-06
```

Example 2: multicolumn text file for a two-dimensional histogram of two dihedral angles (periodic interval with 6° bins):

```
#  2
#  -180.0  6.0     30           1
#  -180.0  6.0     30           1

   -177.0  -177.0  8.97117e-06
   -177.0  -171.0  1.53525e-06
   ...     ...     ...
   -177.0   177.0  2.442956-06

   -171.0  -177.0  2.04702e-05
   ...     ...     ...
```

5.17 Trajectory output

- `outputValue` ⟨Output a trajectory for this colvar⟩
  Context: `colvar`
  Acceptable values: boolean
  Default value: `on`
  Description: If `colvarsTrajFrequency` is non-zero, the value of this colvar is written to the trajectory file every `colvarsTrajFrequency` steps in the column labeled "<name>".

- `outputVelocity` ⟨Output a velocity trajectory for this colvar⟩
  Context: `colvar`
  Acceptable values: boolean
  Default value: `off`
  Description: If `colvarsTrajFrequency` is defined, the finite-difference calculated velocity of this colvar are written to the trajectory file under the label "v_<name>".

- `outputEnergy` ⟨Output an energy trajectory for this colvar⟩
  Context: `colvar`
  Acceptable values: boolean
  Default value: `off`
  Description: This option applies only to extended Lagrangian colvars. If `colvarsTrajFrequency` is defined, the kinetic energy of the extended degree and freedom and the potential energy of the restraining spring are are written to the trajectory file under the labels "Ek_<name>" and "Ep_<name>".

- `outputTotalForce` ⟨Output a total force trajectory for this colvar⟩
  Context: `colvar`
  Acceptable values: boolean
  Default value: `off`
  Description: If `colvarsTrajFrequency` is defined, the total force on this colvar (i.e. the projection of all atomic total forces onto this colvar — see equation (25) in section 7.2) are written to the trajectory file under the label "`fs_<name>`". For extended Lagrangian colvars, the "total force" felt by the extended degree of freedom is simply the force from the harmonic spring. **Note:** not all components support this option. The physical unit for this force is kcal/mol, divided by the colvar unit U.

- `outputAppliedForce` ⟨Output an applied force trajectory for this colvar⟩
  Context: `colvar`
  Acceptable values: boolean
  Default value: `off`
  Description: If `colvarsTrajFrequency` is defined, the total force applied on this colvar by Colvars biases are written to the trajectory under the label "`fa_<name>`". For extended Lagrangian colvars, this force is actually applied to the extended degree of freedom rather than the geometric colvar itself. The physical unit for this force is kcal/mol divided by the colvar unit.

5.18 Extended Lagrangian

The following options enable extended-system dynamics, where a colvar is coupled to an additional degree of freedom (fictitious particle) by a harmonic spring. This extended coordinate masks the colvar and replaces it transparently from the perspective of biasing and analysis methods. Biasing forces are then applied to the extended degree of freedom, and the actual geometric colvar (function of Cartesian coordinates) only feels the force from the harmonic spring. This is particularly useful when combined with an `abf` bias to perform eABF simulations (7.3).

Note that for some biases (`harmonicWalls`, `histogram`), this masking behavior is controlled by the keyword `bypassExtendedLagrangian`. Specifically for `harmonicWalls`, the default behavior is to bypass extended Lagrangian coordinates and act directly on the actual colvars.

- `extendedLagrangian` ⟨Add extended degree of freedom⟩
  Context: `colvar`
  Acceptable values: boolean
  Default value: `off`
  Description: Adds a fictitious particle to be coupled to the colvar by a harmonic spring. The fictitious mass and the force constant of the coupling potential are derived from the parameters `extendedTimeConstant` and `extendedFluctuation`, described below. Biasing forces on the colvar are applied to this fictitious particle, rather than to the atoms directly. This implements the extended Lagrangian formalism used in some metadynamics simulations [2]. .

- `extendedFluctuation` ⟨Standard deviation between the colvar and the fictitious particle (colvar unit)⟩
  Context: `colvar`
  Acceptable values: positive decimal
  Description: Defines the spring stiffness for the `extendedLagrangian` mode, by setting the typical deviation between the colvar and the extended degree of freedom due to thermal fluctuation. The spring force constant is calculated internally as $k_B T / \sigma^2$, where σ is the value of `extendedFluctuation`.

- `extendedTimeConstant` ⟨Oscillation period of the fictitious particle (fs)⟩
  Context: `colvar`
  Acceptable values: positive decimal
  Default value: `200`
  Description: Defines the inertial mass of the fictitious particle, by setting the oscillation period of the harmonic oscillator formed by the fictitious particle and the spring. The period should be much larger than the MD time step to ensure accurate integration of the extended particle's equation of motion. The fictitious mass is calculated internally as $k_B T(\tau/2\pi\sigma)^2$, where τ is the period and σ is the typical fluctuation (see above).

- `extendedTemp` ⟨Temperature for the extended degree of freedom (K)⟩
  Context: `colvar`
  Acceptable values: positive decimal
  Default value: thermostat temperature
  Description: Temperature used for calculating the coupling force constant of the extended variable (see `extendedFluctuation`) and, if needed, as a target temperature for extended Langevin dynamics (see `extendedLangevinDamping`). This should normally be left at its default value.

- `extendedLangevinDamping` ⟨Damping factor for extended Langevin dynamics (ps^{-1})⟩
  Context: `colvar`
  Acceptable values: positive decimal
  Default value: `1.0`
  Description: If this is non-zero, the extended degree of freedom undergoes Langevin dynamics at temperature `extendedTemp`. The friction force is minus `extendedLangevinDamping` times the velocity. This is useful because the extended dynamics coordinate may heat up in the transient non-equilibrium regime of ABF. Use moderate damping values, to limit viscous friction (potentially slowing down diffusive sampling) and stochastic noise (increasing the variance of statistical measurements). In doubt, use the default value.

5.19 Multiple time-step variables

- `timeStepFactor` ⟨Compute this colvar once in a certain number of timesteps⟩
  Context: `colvar`
  Acceptable values: positive integer
  Default value: `1`
  Description: Instructs this colvar to activate at a time interval equal to the base (MD) timestep times `timeStepFactor`.[10] At other time steps, the value of the variable is not updated, and no biasing forces are applied. Any forces exerted by biases are accumulated over the given time interval, then applied as an impulse at the next update.

5.20 Backward-compatibility

- `subtractAppliedForce` ⟨Do not include biasing forces in the total force for this colvar⟩
  Context: `colvar`
  Acceptable values: boolean

Default value: `off`
Description: If the colvar supports total force calculation (see 5.13.1), all forces applied to this colvar by biases will be removed from the total force. This keyword allows to recover some of the "system force" calculation available in the Colvars module before version 2016-08-10. Please note that removal of *all* other external forces (including biasing forces applied to a different colvar) is *no longer supported*, due to changes in the underlying simulation engines (primarily NAMD). This option may be useful when continuing a previous simulation where the removal of external/applied forces is essential. *For all new simulations, the use of this option is not recommended.*

5.21 Statistical analysis

Run-time calculations of statistical properties that depend explicitly on time can be performed for individual collective variables. Currently, several types of time correlation functions, running averages and running standard deviations are implemented. For run-time computation of histograms, please see the histogram bias (7.10).

- `corrFunc` ⟨Calculate a time correlation function?⟩
  Context: `colvar`
  Acceptable values: boolean
  Default value: `off`
  Description: Whether or not a time correlaction function should be calculated for this colvar.

- `corrFuncWithColvar` ⟨Colvar name for the correlation function⟩
  Context: `colvar`
  Acceptable values: string
  Description: By default, the auto-correlation function (ACF) of this colvar, ξ_i, is calculated. When this option is specified, the correlation function is calculated instead with another colvar, ξ_j, which must be of the same type (scalar, vector, or quaternion) as ξ_i.

- `corrFuncType` ⟨Type of the correlation function⟩
  Context: `colvar`
  Acceptable values: `velocity`, `coordinate` or `coordinate_p2`
  Default value: `velocity`
  Description: With `coordinate` or `velocity`, the correlation function $C_{i,j}(t) = \left\langle \Pi\left(\xi_i(t_0), \xi_j(t_0+t)\right)\right\rangle$ is calculated between the variables ξ_i and ξ_j, or their velocities. $\Pi(\xi_i, \xi_j)$ is the scalar product when calculated between scalar or vector values, whereas for quaternions it is the cosine between the two corresponding rotation axes. With `coordinate_p2`, the second order Legendre polynomial, $(3\cos(\theta)^2 - 1)/2$, is used instead of the cosine.

- `corrFuncNormalize` ⟨Normalize the time correlation function?⟩
  Context: `colvar`
  Acceptable values: boolean
  Default value: `on`
  Description: If enabled, the value of the correlation function at $t = 0$ is normalized to 1; otherwise, it equals to $\left\langle O(\xi_i, \xi_j)\right\rangle$.

- `corrFuncLength` ⟨Length of the time correlation function⟩
  Context: `colvar`

Acceptable values: positive integer
Default value: `1000`
Description: Length (in number of points) of the time correlation function.

- `corrFuncStride` ⟨Stride of the time correlation function⟩
  Context: `colvar`
  Acceptable values: positive integer
  Default value: `1`
  Description: Number of steps between two values of the time correlation function.

- `corrFuncOffset` ⟨Offset of the time correlation function⟩
  Context: `colvar`
  Acceptable values: positive integer
  Default value: `0`
  Description: The starting time (in number of steps) of the time correlation function (default: $t = 0$).
  Note: *the value at t = 0 is always used for the normalization.*

- `corrFuncOutputFile` ⟨Output file for the time correlation function⟩
  Context: `colvar`
  Acceptable values: UNIX filename
  Default value: *outputName*`.<name>.corrfunc.dat`
  Description: The time correlation function is saved in this file.

- `runAve` ⟨Calculate the running average and standard deviation⟩
  Context: `colvar`
  Acceptable values: boolean
  Default value: `off`
  Description: Whether or not the running average and standard deviation should be calculated for this colvar.

- `runAveLength` ⟨Length of the running average window⟩
  Context: `colvar`
  Acceptable values: positive integer
  Default value: `1000`
  Description: Length (in number of points) of the running average window.

- `runAveStride` ⟨Stride of the running average window values⟩
  Context: `colvar`
  Acceptable values: positive integer
  Default value: `1`
  Description: Number of steps between two values within the running average window.

- `runAveOutputFile` ⟨Output file for the running average and standard deviation⟩
  Context: `colvar`
  Acceptable values: UNIX filename
  Default value: *outputName*`.<name>.runave.traj`
  Description: The running average and standard deviation are saved in this file.

6 Selecting atoms

To define collective variables, atoms are usually selected as groups. Each group is defined using an identifier that is unique in the context of the specific colvar component (e.g. for a distance component, the two groups are `group1` and `group2`). The identifier is followed by a brace-delimited block containing selection keywords and other parameters, including an optional `name`:

- `name` ⟨Unique name for the atom group⟩
  Context: atom group
  Acceptable values: string
  Description: This parameter defines a unique name for this atom group, which can be referred to in the definition of other atom groups (including in other colvars) by invoking `atomsOfGroup` as a selection keyword.

6.1 Atom selection keywords

Selection keywords may be used individually or in combination with each other, and each can be repeated any number of times. Selection is incremental: each keyword adds the corresponding atoms to the selection, so that different sets of atoms can be combined. However, atoms included by multiple keywords are only counted once. Below is an example configuration for an atom group called "`atoms`". **Note:** *this is an unusually varied combination of selection keywords, demonstrating how they can be combined together: most simulations only use one of them.*

```
atoms {

  # add atoms 1 and 3 to this group (note: the first atom in the system is 1)
  atomNumbers {
    1 3
  }

  # add atoms starting from 20 up to and including 50
  atomNumbersRange 20-50

  # add all the atoms with occupancy 2 in the file atoms.pdb
  atomsFile atoms.pdb
  atomsCol O
  atomsColValue 2.0

  # add all the C-alphas within residues 11 to 20 of segments "PR1" and "PR2"
  psfSegID PR1 PR2
  atomNameResidueRange CA 11-20
  atomNameResidueRange CA 11-20

  # add index group (requires a .ndx file to be provided globally)
```

```
  indexGroup Water
}
```

The resulting selection includes atoms 1 and 3, those between 20 and 50, the C_α atoms between residues 11 and 20 of the two segments PR1 and PR2, and those in the index group called "Water". The indices of this group are read from the file provided by the global keyword indexFile.

In the current version, the Colvars module does not manipulate VMD atom selections directly: however, these can be converted to atom groups within the Colvars configuration string, using selection keywords such as atomNumbers. The complete list of selection keywords available in VMD is:

- atomNumbers ⟨List of atom numbers⟩
  Context: atom group
  Acceptable values: space-separated list of positive integers
  Description: This option adds to the group all the atoms whose numbers are in the list. *The number of the first atom in the system is 1: to convert from a VMD selection, use "atomselect get serial".*

- indexGroup ⟨Name of index group to be used (GROMACS format)⟩
  Context: atom group
  Acceptable values: string
  Description: If the name of an index file has been provided by indexFile, this option allows to select one index group from that file: the atoms from that index group will be used to define the current group.

- atomsOfGroup ⟨Name of group defined previously⟩
  Context: atom group
  Acceptable values: string
  Description: Refers to a group defined previously using its user-defined name. This adds all atoms of that named group to the current group.

- atomNumbersRange ⟨Atoms within a number range⟩
  Context: atom group
  Acceptable values: <Starting number>-<Ending number>
  Description: This option includes in the group all atoms whose numbers are within the range specified. *The number of the first atom in the system is 1.*

- atomNameResidueRange ⟨Named atoms within a range of residue numbers⟩
  Context: atom group
  Acceptable values: <Atom name> <Starting residue>-<Ending residue>
  Description: This option adds to the group all the atoms with the provided name, within residues in the given range.

- psfSegID ⟨PSF segment identifier⟩
  Context: atom group
  Acceptable values: space-separated list of strings (max 4 characters)
  Description: This option sets the PSF segment identifier for atomNameResidueRange. Multiple values may be provided, which correspond to multiple instances of atomNameResidueRange, in order of their occurrence. This option is only necessary if a PSF topology file is used.

- atomsFile ⟨PDB file name for atom selection⟩
  Context: atom group

Acceptable values: UNIX filename
Description: This option selects atoms from the PDB file provided and adds them to the group according to numerical flags in the column `atomsCol`. **Note:** *the sequence of atoms in the PDB file provided must match that in the system's topology*.

- `atomsCol` ⟨PDB column to use for atom selection flags⟩
  Context: atom group
  Acceptable values: `O`, `B`, `X`, `Y`, or `Z`
  Description: This option specifies which PDB column in `atomsFile` is used to determine which atoms are to be included in the group.

- `atomsColValue` ⟨Atom selection flag in the PDB column⟩
  Context: atom group
  Acceptable values: positive decimal
  Description: If defined, this value in `atomsCol` identifies atoms in `atomsFile` that are included in the group. If undefined, all atoms with a non-zero value in `atomsCol` are included.

- `dummyAtom` ⟨Dummy atom position (Å)⟩
  Context: atom group
  Acceptable values: `(x, y, z)` triplet
  Description: Instead of selecting any atom, this option makes the group a virtual particle at a fixed position in space. This is useful e.g. to replace a group's center of geometry with a user-defined position.

6.2 Moving frame of reference.

The following options define an automatic calculation of an optimal translation (`centerReference`) or optimal rotation (`rotateReference`), that superimposes the positions of this group to a provided set of reference coordinates. This can allow, for example, to effectively remove from certain colvars the effects of molecular tumbling and of diffusion. Given the set of atomic positions $\mathbf{x}_i$, the colvar ξ can be defined on a set of roto-translated positions $\mathbf{x}'_i = R(\mathbf{x}_i - \mathbf{x}^{\mathrm{C}}) + \mathbf{x}^{\mathrm{ref}}$. $\mathbf{x}^{\mathrm{C}}$ is the geometric center of the $\mathbf{x}_i$, R is the optimal rotation matrix to the reference positions and $\mathbf{x}^{\mathrm{ref}}$ is the geometric center of the reference positions.

Components that are defined based on pairwise distances are naturally invariant under global roto-translations. Other components are instead affected by global rotations or translations: however, they can be made invariant if they are expressed in the frame of reference of a chosen group of atoms, using the `centerReference` and `rotateReference` options. Finally, a few components are defined by convention using a roto-translated frame (e.g. the minimal RMSD): for these components, `centerReference` and `rotateReference` are enabled by default. In typical applications, the default settings result in the expected behavior.

Warning on rotating frames of reference and periodic boundary conditions. `rotateReference` affects coordinates that depend on minimum-image distances in periodic boundary conditions (PBC). After rotation of the coordinates, the periodic cell vectors become irrelevant: the rotated system is effectively non-periodic. A safe way to handle this is to ensure that the relevant inter-group distance vectors remain smaller than the half-size of the periodic cell. If this is not desirable, one should avoid the rotating frame of reference, and apply orientational restraints to the reference group instead, in order to keep the orientation of the reference group consistent with the orientation of the periodic cell.

Warning on rotating frames of reference and ABF. Note that `centerReference` and `rotateReference` may affect the Jacobian derivative of colvar components in a way that is not taken into account by default. Be careful when using these options in ABF simulations or when using total force values.

- `centerReference` ⟨Implicitly remove translations for this group⟩
  Context: atom group
  Acceptable values: boolean
  Default value: `off`
  Description: If this option is `on`, the center of geometry of the group will be aligned with that of the reference positions provided by either `refPositions` or `refPositionsFile`. Colvar components will only have access to the aligned positions. **Note**: unless otherwise specified, `rmsd` and `eigenvector` set this option to `on` *by default*.

- `rotateReference` ⟨Implicitly remove rotations for this group⟩
  Context: atom group
  Acceptable values: boolean
  Default value: `off`
  Description: If this option is `on`, the coordinates of this group will be optimally superimposed to the reference positions provided by either `refPositions` or `refPositionsFile`. The rotation will be performed around the center of geometry if `centerReference` is `on`, or around the origin otherwise. The algorithm used is the same employed by the `orientation` colvar component [3]. Forces applied to the atoms of this group will also be implicitly rotated back to the original frame. **Note**: unless otherwise specified, `rmsd` and `eigenvector` set this option to `on` *by default*.

- `refPositions` ⟨Reference positions for fitting (Å)⟩
  Context: atom group
  Acceptable values: space-separated list of `(x, y, z)` triplets
  Description: This option provides a list of reference coordinates for `centerReference` and/or `rotateReference`, and is mutually exclusive with `refPositionsFile`. If only `centerReference` is `on`, the list may contain a single (x, y, z) triplet; if also `rotateReference` is `on`, the list should be as long as the atom group, and *its order must match the order in which atoms were defined*.

- `refPositionsFile` ⟨File containing the reference positions for fitting⟩
  Context: atom group
  Acceptable values: UNIX filename
  Description: This option provides a list of reference coordinates for `centerReference` and/or `rotateReference`, and is mutually exclusive with `refPositions`. The acceptable file format is XYZ, which is read in double precision, or PDB; *the latter is discouraged if the precision of the reference coordinates is a concern*. Atomic positions are read differently depending on the following scenarios: **(i)** the file contains exactly as many records as the atoms in the group: all positions are read in sequence; **(ii)** (most common case) the file contains coordinates for the entire system: only the positions corresponding to the numeric indices of the atom group are read; **(iii)** if the file is a PDB file and `refPositionsCol` is specified, positions are read according to the value of the column `refPositionsCol` (which may be the same as `atomsCol`). In each case, atoms are read from the file *in order of increasing number*.

- `refPositionsCol` ⟨PDB column containing atom flags⟩
  Context: atom group
  Acceptable values: `O`, `B`, `X`, `Y`, or `Z`

Description: Like `atomsCol` for `atomsFile`, indicates which column to use to identify the atoms in `refPositionsFile` (if this is a PDB file).

- `refPositionsColValue` ⟨Atom selection flag in the PDB column⟩
  Context: atom group
  Acceptable values: positive decimal
  Description: Analogous to `atomsColValue`, but applied to `refPositionsCol`.

- `fittingGroup` ⟨Use an alternate set of atoms to define the roto-translation⟩
  Context: atom group
  Acceptable values: Block `fittingGroup { ... }`
  Default value: This atom group itself
  Description: If either `centerReference` or `rotateReference` is defined, this keyword defines an alternate atom group to calculate the optimal roto-translation. Use this option to define a continuous rotation if the structure of the group involved changes significantly (a typical symptom would be the message "Warning: discontinuous rotation!").

  The following example illustrates the use of `fittingGroup` as part of a Distance to Bound Configuration (DBC) coordinate for use in ligand restraints for binding affinity calculations.[11] The group called "`atoms`" describes coordinates of a ligand's atoms, expressed in a moving frame of reference tied to a binding site (here within a protein). An optimal roto-translation is calculated automatically by fitting the C_α trace of the rest of the protein onto the coordinates provided by a PDB file. To define a DBC coordinate, this atom group would be used within an `rmsd` function.

```
# Example: defining a group "atoms" (the ligand) whose coordinates are expressed
# in a roto-translated frame of reference defined by a second group (the receptor)
atoms {

  atomNumbers 1 2 3 4 5 6 7 # atoms of the ligand (1-based)

  centerReference yes
  rotateReference yes
  fittingGroup {
    # define the frame by fitting alpha carbon atoms
    # in 2 protein segments close to the site
    psfSegID PROT PROT
    atomNameResidueRange CA 1-40
    atomNameResidueRange CA 59-100
  }
  refPositionsFile all.pdb # can be the entire system
}
```

The following two options have default values appropriate for the vast majority of applications, and are only provided to support rare, special cases.

- `enableFitGradients` ⟨Include the roto-translational contribution to colvar gradients⟩
  Context: atom group
  Acceptable values: boolean
  Default value: `on`

Description: When either `centerReference` or `rotateReference` is on, the gradients of some colvars include terms proportional to $\partial R/\partial \mathbf{x}_i$ (rotational gradients) and $\partial \mathbf{x}^{\mathrm{C}}/\partial \mathbf{x}_i$ (translational gradients). By default, these terms are calculated and included in the total gradients; if this option is set to `off`, they are neglected. In the case of a minimum RMSD component, this flag is automatically disabled because the contributions of those derivatives to the gradients cancel out.

- `enableForces` ⟨Apply forces from this colvar to this group⟩
  Context: atom group
  Acceptable values: boolean
  Default value: `on`
  Description: If this option is `off`, no forces are applied the atoms in the group. Other forces are not affected (i.e. those from the MD engine, from other colvars, and other external forces). For dummy atoms, this option is `off` by default.

6.3 Treatment of periodic boundary conditions.

When periodic boundary conditions are defined, the Colvars module requires that the coordinates of each molecular fragment are contiguous, without "jumps" when a fragment is partially wrapped near a periodic boundary. The Colvars module relies on this assumption when calculating a group's center of geometry, but the condition may fail if the group spans different molecules. In general, coordinate wrapping does not affect the calculation of colvars if each atom group satisfies one or more of the following:

i) it is composed by only one atom;

ii) it is used by a colvar component which does not make use of its center of geometry, but only of pairwise distances (`distanceInv`, `coordNum`, `hBond`, `alpha`, `dihedralPC`);

iii) it is used by a colvar component that ignores the ill-defined Cartesian components of its center of mass (such as the x and y components of a membrane's center of mass modeled with `distanceZ`).

If none of these conditions are met, wrapping may affect the calculation of collective variables: a possible solution is to use `pbc wrap` or `pbc unwrap` prior to processing a trajectory with the Colvars module.

6.4 Performance of a Colvars calculation based on group size.

In simulations performed with message-passing programs (such as NAMD or LAMMPS), the calculation of energy and forces is distributed (i.e., parallelized) across multiple nodes, as well as over the processor cores of each node. When Colvars is enabled, certain atomic coordinates are collected on a single node, where the calculation of collective variables and of their biases is executed. This means that for simulations over large numbers of nodes, a Colvars calculation may produce a significant overhead, coming from the costs of transmitting atomic coordinates to one node and of processing them.

Performance can be improved in multiple ways:

- The calculation of variables, components and biases can be distributed over the processor cores of the node where the Colvars module is executed. Currently, an equal weight is assigned to each colvar, or to each component of those colvars that include more than one component. The performance of simulations that use many colvars or components is improved automatically. For simulations that

use a single large colvar, it may be advisable to partition it in multiple components, which will be then distributed across the available cores. If printed, the message "SMP parallelism is available." indicates the availability of the option (will be supported in a future relase of VMD). If available, the option is turned on by default, but may be disabled using the keyword `smp` if required for debugging.

- As a general rule, the size of atom groups should be kept relatively small (up to a few thousands of atoms, depending on the size of the entire system in comparison). To gain an estimate of the computational cost of a large colvar, one can use a test calculation of the same colvar in VMD (hint: use the `time` Tcl command to measure the cost of running `cv update`).

7 Biasing and analysis methods

A biasing or analysis method can be applied to existing collective variables by using the following configuration:

```
<biastype> {
  name <name>
  colvars <xi1> <xi2> ...
  <parameters>
}
```

The keyword <biastype> indicates the method of choice. There can be multiple instances of the same method, e.g. using multiple harmonic blocks allows defining multiple restraints.

All biasing and analysis methods implemented recognize the following options:

- name ⟨Identifier for the bias⟩
  Context: colvar bias
  Acceptable values: string
  Default value: <type of bias><bias index>
  Description: This string is used to identify the bias or analysis method in the output, and to name some output files. **Tip:** because the default name depends on the order of definition, but the outcome of the simulation does not, it may be convenient to assign consistent names for certain biases; for example, you may want to name a moving harmonic restraint smd, so that it can always be identified regardless of the presence of other restraints.

- colvars ⟨Collective variables involved⟩
  Context: colvar bias
  Acceptable values: space-separated list of colvar names
  Description: This option selects by name all the variables to which this bias or analysis will be applied.

- outputEnergy ⟨Write the current bias energy to the trajectory file⟩
  Context: colvar bias
  Acceptable values: boolean
  Default value: off
  Description: If this option is chosen and colvarsTrajFrequency is not zero, the current value of the biasing energy will be written to the trajectory file during the simulation.

- bypassExtendedLagrangian ⟨Apply bias to actual colvars, bypassing extended coordinates⟩
  Context: colvar bias
  Acceptable values: boolean
  Default value: off
  Description: This option is implemented by the harmonicWalls and histogram biases. It is only relevant if the bias is applied to one or several extended-Lagrangian colvars (5.18), for example within an eABF (7.3) simulation. Usually, biases use the value of the extended coordinate as a proxy for the actual colvar, and their biasing forces are applied to the extended coordinates as well. If bypassExtendedLagrangian is enabled, the bias behaves as if there were no extended coordinates,

and accesses the value of the underlying colvars, applying any biasing forces along the gradients of those variables.

7.1 Thermodynamic integration

The methods implemented here provide a variety of estimators of conformational free-energies. These are carried out at run-time, or with the use of post-processing tools over the generated output files. The specifics of each estimator are discussed in the documentation of each biasing or analysis method.

A special case is the traditional thermodynamic integration (TI) method, used for example to compute potentials of mean force (PMFs). Most types of restraints (7.5, 7.7, 7.8, ...) as well as metadynamics (7.4) can optionally use TI alongside their own estimator, based on the keywords documented below.

- `writeTIPMF` ⟨Write the PMF computed by thermodynamic integration⟩
  Context: colvar bias
  Acceptable values: boolean
  Default value: `off`
  Description: If the bias is applied to a variable that supports the calculation of total forces (see `outputtotalForce` and 5.13.1), this option allows calculating the corresponding PMF by thermodynamic integration, and writing it to the file *outputName*`.<name>.ti.pmf`, where `<name>` is the name of the bias and the contents of the file are in multicolumn text format (5.16.1). The total force includes the forces applied to the variable by all bias, except those from this bias itself. If any bias applies time-dependent forces besides the one using this option, an error is raised.

- `writeTISamples` ⟨Write the free-energy gradient samples⟩
  Context: colvar bias
  Acceptable values: boolean
  Default value: `off`
  Description: This option allows to compute total forces for use with thermodynamic integration as done by the keyword `writeTIPMF`. The names of the files containing the variables' histogram and mean thermodynamic forces are *outputName*`.<name>.ti.count` and *outputName*`.<name>.ti.force`, respectively: these can be used by `abf_integrate` (see 7.2.4) or similar utility. Note that because the `.force` file contains mean forces instead of free-energy gradients, `abf_integrate <filename> -s -1.0` should be used. This option is on by default when `writeTIPMF` is on, but can be enabled separately if the bias is applied to more than one variable, making not possible the direct integration of the PMF at runtime. If any bias applies time-dependent forces besides the one using this option, an error is raised.

In adaptive biasing force (ABF) (7.2) the above keywords are not recognized, because their functionality is either included already (conventional ABF) or not available (extended-system ABF).

7.2 Adaptive Biasing Force

For a full description of the Adaptive Biasing Force method, see reference [12]. For details about this implementation, see references [13] and [14]. **When publishing research that makes use of this functionality, please cite references [12] and [14].**

An alternate usage of this feature is the application of custom tabulated biasing potentials to one or more colvars. See `inputPrefix` and `updateBias` below.

Combining ABF with the extended Lagrangian feature (5.18) of the variables produces the extended-system ABF variant of the method (7.3).

ABF is based on the thermodynamic integration (TI) scheme for computing free energy profiles. The free energy as a function of a set of collective variables $\xi = (\xi_i)_{i\in[1,n]}$ is defined from the canonical distribution of ξ, $\mathscr{P}(\xi)$:

$$A(\xi) = -\frac{1}{\beta}\ln \mathscr{P}(\xi) + A_0 \tag{21}$$

In the TI formalism, the free energy is obtained from its gradient, which is generally calculated in the form of the average of a force F_ξ exerted on ξ, taken over an iso-ξ surface:

$$\nabla_\xi A(\xi) = \left\langle -F_\xi \right\rangle_\xi \tag{22}$$

Several formulae that take the form of (22) have been proposed. This implementation relies partly on the classic formulation [15], and partly on a more versatile scheme originating in a work by Ruiz-Montero et al. [16], generalized by den Otter [17] and extended to multiple variables by Ciccotti et al. [18]. Consider a system subject to constraints of the form $\sigma_k(x) = 0$. Let $(v_i)_{i\in[1,n]}$ be arbitrarily chosen vector fields ($\mathbb{R}^{3N} \to \mathbb{R}^{3N}$) verifying, for all i, j, and k:

$$v_i \cdot \nabla_x \xi_j = \delta_{ij} \tag{23}$$
$$v_i \cdot \nabla_x \sigma_k = 0 \tag{24}$$

then the following holds [18]:

$$\frac{\partial A}{\partial \xi_i} = \langle v_i \cdot \nabla_x V - k_B T \nabla_x \cdot v_i \rangle_\xi \tag{25}$$

where V is the potential energy function. v_i can be interpreted as the direction along which the force acting on variable ξ_i is measured, whereas the second term in the average corresponds to the geometric entropy contribution that appears as a Jacobian correction in the classic formalism [15]. Condition (23) states that the direction along which the total force on ξ_i is measured is orthogonal to the gradient of ξ_j, which means that the force measured on ξ_i does not act on ξ_j.

Equation (24) implies that constraint forces are orthogonal to the directions along which the free energy gradient is measured, so that the measurement is effectively performed on unconstrained degrees of freedom.

In the framework of ABF, $\mathbf{F}_\xi$ is accumulated in bins of finite size $\delta\xi$, thereby providing an estimate of the free energy gradient according to equation (22). The biasing force applied along the collective variables to overcome free energy barriers is calculated as:

$$\mathbf{F}^{\mathrm{ABF}} = \alpha(N_\xi) \times \nabla_x \widetilde{A}(\xi) \tag{26}$$

where $\nabla_x \widetilde{A}$ denotes the current estimate of the free energy gradient at the current point ξ in the collective variable subspace, and $\alpha(N_\xi)$ is a scaling factor that is ramped from 0 to 1 as the local number of samples N_ξ increases to prevent nonequilibrium effects in the early phase of the simulation, when the gradient estimate has a large variance. See the `fullSamples` parameter below for details.

As sampling of the phase space proceeds, the estimate $\nabla_x \widetilde{A}$ is progressively refined. The biasing force introduced in the equations of motion guarantees that in the bin centered around ξ, the forces acting along the selected collective variables average to zero over time. Eventually, as the undelying free energy surface is canceled by the adaptive bias, evolution of the system along ξ is governed mainly by diffusion. Although

this implementation of ABF can in principle be used in arbitrary dimension, a higher-dimension collective variable space is likely to be difficult to sample and visualize. Most commonly, the number of variables is one or two, sometimes three.

7.2.1 ABF requirements on collective variables

The following conditions must be met for an ABF simulation to be possible and to produce an accurate estimate of the free energy profile. Note that these requirements do not apply when using the extended-system ABF method (7.3).

1. *Only linear combinations* of colvar components can be used in ABF calculations.
2. *Availability of total forces* is necessary. The following colvar components can be used in ABF calculations: `distance`, `distance_xy`, `distance_z`, `angle`, `dihedral`, `gyration`, `rmsd` and `eigenvector`. Atom groups may not be replaced by dummy atoms, unless they are excluded from the force measurement by specifying `oneSiteTotalForce`, if available.
3. *Mutual orthogonality of colvars*. In a multidimensional ABF calculation, equation (23) must be satisfied for any two colvars ξ_i and ξ_j. Various cases fulfill this orthogonality condition:
   - ξ_i and ξ_j are based on non-overlapping sets of atoms.
   - atoms involved in the force measurement on ξ_i do not participate in the definition of ξ_j. This can be obtained using the option `oneSiteTotalForce` of the `distance`, `angle`, and `dihedral` components (example: Ramachandran angles ϕ, ψ).
   - ξ_i and ξ_j are orthogonal by construction. Useful cases are the sum and difference of two components, or `distance_z` and `distance_xy` using the same axis.
4. *Mutual orthogonality of components*: when several components are combined into a colvar, it is assumed that their vectors v_i (equation (25)) are mutually orthogonal. The cases described for colvars in the previous paragraph apply.
5. *Orthogonality of colvars and constraints*: equation 24 can be satisfied in two simple ways, if either no constrained atoms are involved in the force measurement (see point 3 above) or pairs of atoms joined by a constrained bond are part of an *atom group* which only intervenes through its center (center of mass or geometric center) in the force measurement. In the latter case, the contributions of the two atoms to the left-hand side of equation 24 cancel out. For example, all atoms of a rigid TIP3P water molecule can safely be included in an atom group used in a `distance` component.

7.2.2 Parameters for ABF

ABF depends on parameters from collective variables to define the grid on which free energy gradients are computed. In the direction of each colvar, the grid ranges from `lowerBoundary` to `upperBoundary`, and the bin width (grid spacing) is set by the `width` parameter. The following specific parameters can be set in the ABF configuration block:

- `name`: see definition of `name` in sec. 7 (biasing and analysis methods)
- `colvars`: see definition of `colvars` in sec. 7 (biasing and analysis methods)

- `outputEnergy`: see definition of `outputEnergy` in sec. 7 (biasing and analysis methods)

- `fullSamples` ⟨Number of samples in a bin prior to application of the ABF⟩
  Context: `abf`
  Acceptable values: positive integer
  Default value: 200
  Description: To avoid nonequilibrium effects due to large fluctuations of the force exerted along the colvars, it is recommended to apply a biasing force only after a the estimate has started converging. If `fullSamples` is non-zero, the applied biasing force is scaled by a factor $\alpha(N_\xi)$ between 0 and 1. If the number of samples N_ξ in the current bin is higher than `fullSamples`, the factor is one. If it is less than half of `fullSamples`, the factor is zero and no bias is applied. Between those two thresholds, the factor follows a linear ramp from 0 to 1: $\alpha(N_\xi) = (2N_\xi/\text{fullSamples}) - 1$.

- `maxForce` ⟨Maximum magnitude of the ABF force⟩
  Context: `abf`
  Acceptable values: positive decimals (one per colvar)
  Default value: disabled
  Description: This option enforces a cap on the magnitude of the biasing force effectively applied by this ABF bias on each colvar. This can be useful in the presence of singularities in the PMF such as hard walls, where the discretization of the average force becomes very inaccurate, causing the colvar's diffusion to get "stuck" at the singularity. To enable this cap, provide one non-negative value for each colvar. The unit of force is kcal/mol divided by the colvar unit.

- `hideJacobian` ⟨Remove geometric entropy term from calculated free energy gradient?⟩
  Context: `abf`
  Acceptable values: boolean
  Default value: `no`
  Description: In a few special cases, most notably distance-based variables, an alternate definition of the potential of mean force is traditionally used, which excludes the Jacobian term describing the effect of geometric entropy on the distribution of the variable. This results, for example, in particle-particle potentials of mean force being flat at large separations. Setting this parameter to `yes` causes the output data to follow that convention, by removing this contribution from the output gradients while applying internally the corresponding correction to ensure uniform sampling. It is not allowed for colvars with multiple components.

- `outputFreq` ⟨Frequency (in timesteps) at which ABF data files are refreshed⟩
  Context: `abf`
  Acceptable values: positive integer
  Default value: Colvars module restart frequency
  Description: The files containing the free energy gradient estimate and sampling histogram (and the PMF in one-dimensional calculations) are written on disk at the given time interval.

- `historyFreq` ⟨Frequency (in timesteps) at which ABF history files are accumulated⟩
  Context: `abf`
  Acceptable values: positive integer
  Default value: 0
  Description: If this number is non-zero, the free energy gradient estimate and sampling histogram (and the PMF in one-dimensional calculations) are appended to files on disk at the given time interval. History file names use the same prefix as output files, with "`.hist`" appended.

- `inputPrefix` ⟨Filename prefix for reading ABF data⟩
  Context: `abf`
  Acceptable values: list of strings
  Description: If this parameter is set, for each item in the list, ABF tries to read a gradient and a sampling files named <`inputPrefix`>`.grad` and <`inputPrefix`>`.count`. This is done at startup and sets the initial state of the ABF algorithm. The data from all provided files is combined appropriately. Also, the grid definition (min and max values, width) need not be the same that for the current run. This command is useful to piece together data from simulations in different regions of collective variable space, or change the colvar boundary values and widths. Note that it is not recommended to use it to switch to a smaller width, as that will leave some bins empty in the finer data grid. This option is NOT compatible with reading the data from a restart file (`cv load` command).

- `applyBias` ⟨Apply the ABF bias?⟩
  Context: `abf`
  Acceptable values: boolean
  Default value: `yes`
  Description: If this is set to no, the calculation proceeds normally but the adaptive biasing force is not applied. Data is still collected to compute the free energy gradient. This is mostly intended for testing purposes, and should not be used in routine simulations.

- `updateBias` ⟨Update the ABF bias?⟩
  Context: `abf`
  Acceptable values: boolean
  Default value: `yes`
  Description: If this is set to no, the initial biasing force (e.g. read from a restart file or through `inputPrefix`) is not updated during the simulation. As a result, a constant bias is applied. This can be used to apply a custom, tabulated biasing potential to any combination of colvars. To that effect, one should prepare a gradient file containing the gradient of the potential to be applied (negative of the bias force), and a count file containing only values greater than `fullSamples`. These files must match the grid parameters of the colvars.

7.2.3 Output files

The ABF bias produces the following files, all in multicolumn text format (5.16.1):

- *outputName*`.grad`: current estimate of the free energy gradient (grid), in multicolumn;
- *outputName*`.count`: histogram of samples collected, on the same grid;
- *outputName*`.pmf`: integrated free energy profile or PMF (for dimensions 1, 2 or 3).

Also in the case of one-dimensional calculations, the ABF bias can report its current energy via `outputEnergy`; in higher dimensions, such computation is not implemented and the energy reported is zero.

If several ABF biases are defined concurrently, their name is inserted to produce unique filenames for output, as in *outputName*`.abf1.grad`. This should not be done routinely and could lead to meaningless results: only do it if you know what you are doing!

If the colvar space has been partitioned into sections (*windows*) in which independent ABF simulations have been run, the resulting data can be merged using the `inputPrefix` option described above (a run of 0 steps is enough).

7.2.4 Multidimensional free energy surfaces

If a one-dimensional calculation is performed, the estimated free energy gradient is integrated using a simple rectangle rule. In dimension 2 or 3, it is calculated as the solution of a Poisson equation:

$$\Delta A(\xi) = -\nabla \cdot \langle F_\xi \rangle \tag{27}$$

wehere ΔA is the Laplacian of the free energy. The potential of mean force is written under the file name `<outputName>.pmf`, in a plain text format (see 5.16.1) that can be read by most data plotting and analysis programs (e.g. Gnuplot). This applies periodic boundary conditions to periodic coordinates, and Neumann boundary conditions otherwise (imposed free energy gradient at the boundary of the domain). Note that the grid used for free energy discretization is extended by one point along non-periodic coordinates, but not along periodic coordinates.

In dimension 4 or greater, integrating the discretized gradient becomes non-trivial. The standalone utility `abf_integrate` is provided to perform that task. Because 4D ABF calculations are uncommon, **this tool is practically deprecated** by the Poisson integration described above.

`abf_integrate` reads the gradient data and uses it to perform a Monte-Carlo (M-C) simulation in discretized collective variable space (specifically, on the same grid used by ABF to discretize the free energy gradient). By default, a history-dependent bias (similar in spirit to metadynamics) is used: at each M-C step, the bias at the current position is incremented by a preset amount (the *hill height*). Upon convergence, this bias counteracts optimally the underlying gradient; it is negated to obtain the estimate of the free energy surface.

`abf_integrate` is invoked using the command-line:

```
abf_integrate <gradient_file> [-n <nsteps>] [-t <temp>] [-m (0|1)] [-h <hill_height>] [-f <factor>]
```

The gradient file name is provided first, followed by other parameters in any order. They are described below, with their default value in square brackets:

- `-n`: number of M-C steps to be performed; by default, a minimal number of steps is chosen based on the size of the grid, and the integration runs until a convergence criterion is satisfied (based on the RMSD between the target gradient and the real PMF gradient)
- `-t`: temperature for M-C sampling (unrelated to the simulation temperature) [500 K]
- `-s`: scaling factor for the gradients; when using a histogram of total forces obtained from `outputTotalForce` or the `.force` file written by `writeTISamples`, a scaling factor of -1 should be used [1.0]
- `-m`: use metadynamics-like biased sampling? (0 = false) [1]
- `-h`: increment for the history-dependent bias ("hill height") [0.01 kcal/mol]
- `-f`: if non-zero, this factor is used to scale the increment stepwise in the second half of the M-C sampling to refine the free energy estimate [0.5]

Using the default values of all parameters should give reasonable results in most cases.

`abf_integrate` produces the following output files:

- `<gradient_file>.pmf`: computed free energy surface
- `<gradient_file>.histo`: histogram of M-C sampling (not usable in a straightforward way if the history-dependent bias has been applied)
- `<gradient_file>.est`: estimated gradient of the calculated free energy surface (from finite differences)

- `<gradient_file>.dev`: deviation between the user-provided numerical gradient and the actual gradient of the calculated free energy surface. The RMS norm of this vector field is used as a convergence criteria and displayed periodically during the integration.

Note: Typically, the "deviation" vector field does not vanish as the integration converges. This happens because the numerical estimate of the gradient does not exactly derive from a potential, due to numerical approximations used to obtain it (finite sampling and discretization on a grid).

7.3 Extended-system Adaptive Biasing Force (eABF)

Extended-system ABF (eABF) is a variant of ABF (7.2) where the bias is not applied directly to the collective variable, but to an extended coordinate ("fictitious variable") λ that evolves dynamically according to Newtonian or Langevin dynamics. Such an extended coordinate is enabled for a given colvar using the `extendedLagrangian` and associated keywords (5.18). The theory of eABF and the present implementation are documented in detail in reference [19].

Defining an ABF bias on a colvar wherein the `extendedLagrangian` option is active will perform eABF automatically; there is no dedicated option.

The extended variable λ is coupled to the colvar $z = \xi(q)$ by the harmonic potential $(k/2)(z-\lambda)^2$. Under eABF dynamics, the adaptive bias on λ is the running estimate of the average spring force:

$$F^{\text{bias}}(\lambda^*) = \langle k(\lambda - z)\rangle_{\lambda^*} \tag{28}$$

where the angle brackets indicate a canonical average conditioned by $\lambda = \lambda^*$. At long simulation times, eABF produces a flat histogram of the extended variable λ, and a flattened histogram of ξ, whose exact shape depends on the strength of the coupling as defined by `extendedFluctuation` in the colvar. Coupling should be somewhat loose for faster exploration and convergence, but strong enough that the bias does help overcome barriers along the colvar ξ.[19] Distribution of the colvar may be assessed by plotting its histogram, which is written to the *outputName*`.zcount` file in every eABF simulation. Note that a `histogram` bias (7.10) applied to an extended-Lagrangian colvar will access the extended degree of freedom λ, not the original colvar ξ; however, the joint histogram may be explicitly requested by listing the name of the colvar twice in a row within the `colvars` parameter of the `histogram` block.

The eABF PMF is that of the coordinate λ, it is not exactly the free energy profile of ξ. That quantity can be calculated based on the CZAR estimator.

7.3.1 CZAR estimator of the free energy

The *corrected z-averaged restraint* (CZAR) estimator is described in detail in reference [19]. It is computed automatically in eABF simulations, regardless of the number of colvars involved. Note that ABF may also be applied on a combination of extended and non-extended colvars; in that case, CZAR still provides an unbiased estimate of the free energy gradient.

CZAR estimates the free energy gradient as:

$$A'(z) = -\frac{1}{\beta}\frac{d\ln\tilde{\rho}(z)}{dz} + k(\langle\lambda\rangle_z - z). \tag{29}$$

where $z = \xi(q)$ is the colvar, λ is the extended variable harmonically coupled to z with a force constant k, and $\tilde{\rho}(z)$ is the observed distribution (histogram) of z, affected by the eABF bias.

Parameters for the CZAR estimator are:

- `CZARestimator` ⟨Calculate CZAR estimator of the free energy?⟩
  Context: `abf`
  Acceptable values: boolean
  Default value: `yes`
  Description: This option is only available when ABF is performed on extended-Lagrangian colvars. When enabled, it triggers calculation of the free energy following the CZAR estimator.

- `writeCZARwindowFile` ⟨Write internal data from CZAR to a separate file?⟩
  Context: `abf`
  Acceptable values: boolean
  Default value: `no`
  Description: When this option is enabled, eABF simulations will write a file containing the z-averaged restraint force under the name *outputName*`.zgrad`. The same information is always included in the colvars state file, which is sufficient for restarting an eABF simulation. These separate file is only useful when joining adjacent windows from a stratified eABF simulation, either to continue the simulation in a broader window or to compute a CZAR estimate of the PMF over the full range of the coordinate(s). **Important warning.** Unbiased free-energy estimators from eABF dynamics rely on some form of sampling histogram. When running stratified (windowed) calculations this histogram becomes discontinuous, and as a result the free energy gradient estimated by CZAR is inaccurate at the window boundary, resulting in visible "blips" in the PMF. As a workaround, we recommend manually replacing the two free energy gradient values at the boundary, either with the ABF values from .grad files (accurate in the limit of tight coupling), or with values interpolated for the neighboring values of the CZAR gradient.

Similar to ABF, the CZAR estimator produces two output files in multicolumn text format (5.16.1):

- *outputName*`.czar.grad`: current estimate of the free energy gradient (grid), in multicolumn;
- *outputName*`.czar.pmf`: only for one-dimensional calculations, integrated free energy profile or PMF.

The sampling histogram associated with the CZAR estimator is the z-histogram, which is written in the file *outputName*`.zcount`.

7.4 Metadynamics

The metadynamics method uses a history-dependent potential [20] that generalizes to any type of colvars the conformational flooding [21] and local elevation [22] methods, originally formulated to use as colvars the principal components of a covariance matrix or a set of dihedral angles, respectively. The metadynamics potential on the colvars $\xi = (\xi_1, \xi_2, \ldots, \xi_{N_{\text{cv}}})$ is defined as:

$$V_{\text{meta}}(\xi(t)) = \sum_{t'=\delta t, 2\delta t, \ldots}^{t'<t} W \prod_{i=1}^{N_{\text{cv}}} \exp\left(-\frac{(\xi_i(t) - \xi_i(t'))^2}{2\sigma_{\xi_i}^2}\right), \tag{30}$$

where V_{meta} is the history-dependent potential acting on the *current* values of the colvars ξ, and depends only parametrically on the *previous* values of the colvars. V_{meta} is constructed as a sum of N_{cv}-dimensional repulsive Gaussian "hills", whose height is a chosen energy constant W, and whose centers are the previously explored configurations $(\xi(\delta t), \xi(2\delta t), \ldots)$.

During the simulation, the system evolves towards the nearest minimum of the "effective" potential of mean force $\tilde{A}(\xi)$, which is the sum of the "real" underlying potential of mean force $A(\xi)$ and the the metadynamics potential, $V_{\text{meta}}(\xi)$. Therefore, at any given time the probability of observing the configuration ξ^*

is proportional to $\exp\left(-\tilde{A}(\xi^*)/\kappa_B T\right)$: this is also the probability that a new Gaussian "hill" is added at that configuration. If the simulation is run for a sufficiently long time, each local minimum is canceled out by the sum of the Gaussian "hills". At that stage the "effective" potential of mean force $\tilde{A}(\xi)$ is constant, and $-V_{\text{meta}}(\xi)$ is an estimator of the "real" potential of mean force $A(\xi)$, save for an additive constant:

$$A(\xi) \simeq -V_{\text{meta}}(\xi) + K \tag{31}$$

Such estimate of the free energy can be provided by enabling `writeFreeEnergyFile`. Assuming that the set of collective variables includes all relevant degrees of freedom, the predicted error of the estimate is a simple function of the correlation times of the colvars τ_{ξ_i}, and of the user-defined parameters W, σ_{ξ_i} and δt [23]. In typical applications, a good rule of thumb can be to choose the ratio $W/\delta t$ much smaller than $\kappa_B T/\tau_\xi$, where τ_ξ is the longest among ξ's correlation times: σ_{ξ_i} then dictates the resolution of the calculated PMF.

If the metadynamics parameters are chosen correctly, after an equilibration time, t_e, the estimator provided by eq. 31 oscillates on time around the "real" free energy, thereby a better estimate of the latter can be obtained as the time average of the bias potential after t_e [24, 25]:

$$A(\xi) = -\frac{1}{t_{tot} - t_e} \int_{t_e}^{t_{tot}} V_{\text{meta}}(\xi, t)\, dt \tag{32}$$

where t_e is the time after which the bias potential grows (approximately) evenly during the simulation and t_{tot} is the total simulation time. The free energy calculated according to eq. 32 can thus be obtained averaging on time mutiple time-dependent free energy estimates, that can be printed out through the keyword `keepFreeEnergyFiles`. An alternative is to obtain the free energy profiles by summing the hills added during the simulation; the hills trajectory can be printed out by enabling the option `writeHillsTrajectory`.

7.4.1 Treatment of the PMF boundaries

In typical scenarios the Gaussian hills of a metadynamics potential are interpolated and summed together onto a grid, which is much more efficient than computing each hill independently at every step (the keyword `useGrids` is on by default). This numerical approximation typically yields neglibile errors in the resulting PMF [1]. However, due to the finite thickness of the Gaussian function, the metadynamics potential would suddenly vanish each time a variable exceeds its grid boundaries.

To avoid such discontinuity the Colvars metadynamics code will keep an explicit copy of each hill that straddles a grid's boundary, and will use it to compute metadynamics forces outside the grid. This measure is taken to protect the accuracy and stability of a metadynamics simulation, except in cases of "natural" boundaries (for example, the $[0:180]$ interval of an `angle` colvar) or when the flags `hardLowerBoundary` and `hardUpperBoundary` are explicitly set by the user. Unfortunately, processing explicit hills alongside the potential and force grids could easily become inefficient, slowing down the simulation and increasing the state file's size.

In general, it is a good idea to *define a repulsive potential to avoid hills from coming too close to the grid's boundaries*, for example as a `harmonicWalls` restraint (see 7.7).

Example: Using harmonic walls to protect the grid's boundaries.

```
colvar {
  name r
  distance { ... }
  upperBoundary 15.0
```

```
  width 0.2
}

metadynamics {
  name meta_r
  colvars r
  hillWeight 0.001
  hillWidth 2.0
}

harmonicWalls {
  name wall_r
  colvars r
  upperWall 13.0
  upperWallConstant 2.0
}
```

In the colvar `r`, the `distance` function used has a `lowerBoundary` automatically set to 0 Å by default, thus the keyword `lowerBoundary` itself is not mandatory and `hardLowerBoundary` is set to `yes` internally. However, `upperBoundary` does not have such a "natural" choice of value. The metadynamics potential `meta_r` will individually process any hill whose center is too close to the `upperBoundary`, more precisely within fewer grid points than 6 times the Gaussian σ parameter plus one. It goes without saying that if the colvar `r` represents a distance between two freely-moving molecules, it will cross this "threshold" rather frequently.

In this example, where the value of `hillWidth` (2σ) amounts to 2 grid points, the threshold is 6+1 = 7 grid points away from `upperBoundary`: in explicit units, the threshold is 15.0 - 7×0.2 = 13.6 Å.

The `wall_r` restraint included in the example prevents this: the position of its `upperWall` is 13 Å, i.e. 3 grid points below the buffer's threshold (13.6 Å). For the chosen value of `upperWallConstant`, the energy of the `wall_r` bias at `r` = r_{upper} = 13.6 Å is:

$$E(\text{wall\_r}) = \frac{1}{2}k\left(\frac{r - r_{\text{upper}}}{\texttt{width}(r)}\right)^2 = \frac{1}{2}2.0\,(-3)^2 = 9 \text{ kcal/mol}$$

which results in a relative probability $\exp(-E(\text{wall\_r})/\kappa_B T) \simeq 3 \times 10^{-7}$ that `r` crosses the threshold. The probability that `r` exceeds `upperBoundary`, which is further away, has also become vanishingly small. At that point, you may want to set `hardUpperBoundary` to `yes` for `r`, and let `meta_r` know that no special treatment near the grid's boundaries will be needed.

What is the impact of the wall restraint onto the PMF? Not a very complicated one: the PMF reconstructed by metadynamics will simply show a sharp increase in free-energy where the wall potential kicks in (`r` > 13 Å). You may then choose between using the PMF only up until that point and discard the rest, or subtracting the energy of the `harmonicWalls` restraint from the PMF itself. Keep in mind, however, that the statistical convergence of metadynamics may be less accurate where the wall potential is strong.

In summary, although it would be simpler to set the wall's position `upperWall` and the grid's boundary `upperBoundary` to the same number, the finite width of the Gaussian hills calls for setting the former strictly within the latter.

7.4.2 Basic configuration keywords

To enable a metadynamics calculation, a `metadynamics` {...} block must be defined in the Colvars configuration file. Its mandatory keywords are `colvars`, the variables involved, `hillWeight`, the weight parameter W, and the widths 2σ of the Gaussian hills in each dimension given by the single dimensionless parameter `hillWidth`, or more explicitly by the `gaussianSigmas`.

- `name`: see definition of `name` in sec. 7 (biasing and analysis methods)
- `colvars`: see definition of `colvars` in sec. 7 (biasing and analysis methods)
- `outputEnergy`: see definition of `outputEnergy` in sec. 7 (biasing and analysis methods)
- `writeTIPMF`: see definition of `writeTIPMF` in sec. 7 (biasing and analysis methods)
- `writeTISamples`: see definition of `writeTISamples` in sec. 7 (biasing and analysis methods)
- `hillWeight` ⟨Height of each hill (kcal/mol)⟩
  Context: `metadynamics`
  Acceptable values: positive decimal
  Description: This option sets the height W of the Gaussian hills that are added during this run. Lower values provide more accurate sampling of the system's degrees of freedom at the price of longer simulation times to complete a PMF calculation based on metadynamics.
- `hillWidth` ⟨Width 2σ of a Gaussian hill, measured in number of grid points⟩
  Context: `metadynamics`
  Acceptable values: positive decimal
  Description: This keyword sets the Gaussian width $2\sigma_{\xi_i}$ for all colvars, expressed in *number of grid points*, with the grid spacing along each colvar ξ determined by the respective value of `width`. Values between 1 and 3 are recommended for this option: smaller numbers will fail to adequately interpolate each Gaussian function [1], while larger values may be unable to account for steep free-energy gradients. The values of each half-width σ_{ξ_i} in the physical units of ξ_i are also printed by VMD at initialization time; alternatively, they may be set explicitly via `gaussianSigmas`.
- `gaussianSigmas` ⟨Half-widths σ of the Gaussian hill (one for each colvar)⟩
  Context: `metadynamics`
  Acceptable values: space-separated list of decimals
  Description: This option sets the parameters σ_{ξ_i} of the Gaussian hills along each colvar ξ_i, expressed in *the same unit of* ξ_i. No restrictions are placed on each value, but a warning will be printed if `useGrids` is on and the Gaussian width $2\sigma_{\xi_i}$ is smaller than the corresponding grid spacing, `width`(ξ_i). If not given, default values will be computed from the dimensionless number `hillWidth`.
- `newHillFrequency` ⟨Frequency of hill creation⟩
  Context: `metadynamics`
  Acceptable values: positive integer
  Default value: `1000`
  Description: This option sets the number of steps after which a new Gaussian hill is added to the metadynamics potential. The product of this number and the integration time-step defines the parameter δt in eq. 30. Higher values provide more accurate statistical sampling, at the price of longer simulation times to complete a PMF calculation. When analyzing data from a previous simulation in VMD, the metadynamics potential does not need to be updated, and it is useful to set this number to 0.

7.4.3 Output files

When interpolating grids are enabled (default behavior), the PMF is written by default every `colvarsRestartFrequency` steps to the file *outputName*`.pmf` in multicolumn text format (5.16.1). The following two options allow to disable or control this behavior and to track statistical convergence:

- `writeFreeEnergyFile` ⟨Periodically write the PMF for visualization⟩
  Context: `metadynamics`
  Acceptable values: boolean
  Default value: `on`
  Description: When `useGrids` and this option are `on`, the PMF is written every `colvarsRestartFrequency` steps.

- `keepFreeEnergyFiles` ⟨Keep all the PMF files⟩
  Context: `metadynamics`
  Acceptable values: boolean
  Default value: `off`
  Description: When `writeFreeEnergyFile` and this option are `on`, the step number is included in the file name, thus generating a series of PMF files. Activating this option can be useful to follow more closely the convergence of the simulation, by comparing PMFs separated by short times.

7.4.4 Performance optimization

The following options control the computational cost of metadynamics calculations, but do not affect results. Default values are chosen to minimize such cost with no loss of accuracy.

- `useGrids` ⟨Interpolate the hills with grids⟩
  Context: `metadynamics`
  Acceptable values: boolean
  Default value: `on`
  Description: This option discretizes all hills for improved performance, accumulating their energy and their gradients on two separate grids of equal spacing. Grids are defined by the values of `lowerBoundary`, `upperBoundary` and `width` for each colvar. Currently, this option is implemented for all types of variables except the non-scalar types (`distanceDir` or `orientation`). If `expandBoundaries` is defined in one of the colvars, grids are automatically expanded along the direction of that colvar.

- `rebinGrids` ⟨Recompute the grids when reading a state file⟩
  Context: `metadynamics`
  Acceptable values: boolean
  Default value: `off`
  Description: When restarting from a state file, the grid's parameters (boundaries and widths) saved in the state file override those in the configuration file. Enabling this option forces the grids to match those in the current configuration file.

7.4.5 Ensemble-Biased Metadynamics

The ensemble-biased metadynamics (EBMetaD) approach [26] is designed to reproduce a target probability distribution along selected collective variables. Standard metadynamics can be seen as a special case of EBMetaD with a flat distribution as target. This is achieved by weighing the Gaussian functions used in the metadynamics approach by the inverse of the target probability distribution:

$$V_{\mathrm{EBmetaD}}(\xi(t)) = \sum_{t'=\delta t,2\delta t,\ldots}^{t'<t} \frac{W}{\exp(S_\rho)\,\rho_{exp}(\xi(t'))} \prod_{i=1}^{N_{\mathrm{cv}}} \exp\left(-\frac{(\xi_i(t)-\xi_i(t'))^2}{2\sigma_{\xi_i}^2}\right), \tag{33}$$

where $\rho_{exp}(\xi)$ is the target probability distribution and $S_\rho = -\int \rho_{exp}(\xi)\log\rho_{exp}(\xi)\,\mathrm{d}\xi$ its corresponding differential entropy. The method is designed so that during the simulation the resulting distribution of the collective variable ξ converges to $\rho_{exp}(\xi)$. A practical application of EBMetaD is to reproduce an "experimental" probability distribution, for example the distance distribution between spectroscopic labels inferred from Förster resonance energy transfer (FRET) or double electron-electron resonance (DEER) experiments [26].

The PMF along ξ can be estimated from the bias potential and the target ditribution [26]:

$$A(\xi) \simeq -V_{\mathrm{EBmetaD}}(\xi) - \kappa_{\mathrm{B}} T \log\rho_{exp}(\xi) \tag{34}$$

and obtained by enabling `writeFreeEnergyFile`. Similarly to eq. 32, a more accurate estimate of the free energy can be obtained by averaging (after an equilibration time) multiple time-dependent free energy estimates (see `keepFreeEnergyFiles`).

The following additional options define the configuration for the ensemble-biased metadynamics approach:

- `ebMeta` ⟨Perform ensemble-biased metadynamics⟩
  Context: `metadynamics`
  Acceptable values: boolean
  Default value: `off`
  Description: If enabled, this flag activates the ensemble-biased metadynamics as described by Marinelli et al.[26]. The target distribution file, `targetdistfile`, is then required. The keywords `lowerBoundary`, `upperBoundary` and `width` for the respective variables are also needed to set the binning (grid) of the target distribution file.

- `targetDistFile` ⟨Target probability distribution file for ensemble-biased metadynamics⟩
  Context: `metadynamics`
  Acceptable values: multicolumn text file
  Description: This file provides the target probability distribution, $\rho_{exp}(\xi)$, reported in eq. 33. The latter distribution must be a tabulated function provided in a multicolumn text format (see 5.16.1). The provided distribution is then normalized.

- `ebMetaEquilSteps` ⟨Number of equilibration steps for ensemble-biased metadynamics⟩
  Context: `metadynamics`
  Acceptable values: positive integer
  Description: The EBMetaD approach may introduce large hills in regions with small values of the target probability distribution (eq. 33). This happens, for example, if the probability distribution sampled by a conventional molecular dynamics simulation is significantly different from the target distribution. This may lead to instabilities at the beginning of the simulation related to large biasing forces. In this case, it is useful to introduce an equilibration stage in which the bias potential

gradually switches from standard metadynamics (eq. 30) to EBmetaD (eq. 33) as $\lambda V_{\text{meta}}(\xi) + (1 - \lambda)V_{\text{EBmetaD}}(\xi)$, where $\lambda = (\texttt{ebMetaEquilSteps} - \texttt{step})/\texttt{ebMetaEquilSteps}$ and `step` is the current simulation step number.

- `targetDistMinVal` ⟨Minimum value of the target distribution in reference to its `maximum value`⟩
  Context: `metadynamics`
  Acceptable values: positive decimal
  Description: It is useful to set a minimum value of the target probability distribution to avoid values of the latter that are nearly zero, leading to very large hills. This parameter sets the minimum value of the target probability distribution that is expressed as a fraction of its `maximum value`: `minumum value = maximum value X targetDistMinVal`. This implies that 0 < `targetDistMinVal` < 1 and its default value is set to 1/1000000. To avoid divisions by zero (see eq. 33), if `targetDistMinVal` is set as zero, values of ρ_{exp} equal to zero are replaced by the smallest positive value read in the same file.

As with standard metadynamics, multidimensional probability distributions can be targeted using a single `metadynamics` block using multiple `colvars` and a multidimensional target distribution file (see 5.16.1). Instead, multiple probability distributions on different variables can be targeted separately in the same simulation by introducing multiple `metadynamics` blocks with the `ebMeta` option.

Example: EBmetaD configuration for a single variable.

```
colvar {
  name r
  distance {
    group1 { atomNumbers 991 992 }
    group2 { atomNumbers 1762 1763 }
  }
  upperBoundary 100.0
  width 0.1
}

metadynamics {
  name ebmeta
  colvars r
  hillWeight 0.01
  hillWidth 3.0
  ebMeta on
  targetDistFile targetdist1.dat
  ebMetaEquilSteps 500000
}
```

where `targetdist1.dat` is a text file in "multicolumn" format (5.16.1) with the same `width` as the variable `r` (0.1 in this case):

```
#  1
#  0.0    0.1     1000  0

    0.05  0.0012
    0.15  0.0014
   ...    ...
   99.95  0.0010
```

Tip: Besides setting a meaninful value for `targetDistMinVal`, the exploration of unphysically low values of the target distribution (which would lead to very large hills and possibly numerical instabilities) can be also prevented by restricting sampling to a given interval, using e.g. `harmonicWalls` restraint (7.7).

7.4.6 Well-tempered metadynamics

The following options define the configuration for the "well-tempered" metadynamics approach [27]:

- `wellTempered` ⟨Perform well-tempered metadynamics⟩
  Context: `metadynamics`
  Acceptable values: boolean
  Default value: `off`
  Description: If enabled, this flag causes well-tempered metadynamics as described by Barducci et al.[27] to be performed, rather than standard metadynamics. The parameter `biasTemperature` is then required. This feature was contributed by Li Li (Luthey-Schulten group, Department of Chemistry, UIUC).

- `biasTemperature` ⟨Temperature bias for well-tempered metadynamics⟩
  Context: `metadynamics`
  Acceptable values: positive decimal
  Description: When running metadynamics in the long time limit, collective variable space is sampled to a modified temperature $T + \Delta T$. In conventional metadynamics, the temperature "boost" ΔT would constantly increases with time. Instead, in well-tempered metadynamics ΔT must be defined by the user via `biasTemperature`. The written PMF includes the scaling factor $(T + \Delta T)/\Delta T$ [27]. A careful choice of ΔT determines the sampling and convergence rate, and is hence crucial to the success of a well-tempered metadynamics simulation.

7.4.7 Multiple-walker metadynamics

Metadynamics calculations can be performed concurrently by multiple replicas that share a common history. This variant of the method is called multiple-walker metadynamics [28]: the Gaussian hills of all replicas are periodically combined into a single biasing potential, intended to converge to a single PMF.

In the implementation here described [1], replicas communicate through files. This arrangement allows launching the replicas either (1) as a bundle (i.e. a single job in a cluster's queueing system) or (2) as fully independent runs (i.e. as separate jobs for the queueing system). One advantage of the use case (1) is that an identical Colvars configuration can be used for all replicas (otherwise, `replicaID` needs to be manually

set to a different string for each replica). However, the use case (2) is less demanding in terms of high-performance computing resources: a typical scenario would be a computer cluster (including virtual servers from a cloud provider) where not all nodes are connected to each other at high speed, and thus each replica runs on a small group of nodes or a single node.

Whichever way the replicas are started (coupled or not), a shared filesystem is needed so that each replica can read the files created by the others: paths to these files are stored in the shared file `replicasRegistry`. This file, and those listed in it, are read every `replicaUpdateFrequency` steps. Each time the Colvars state file is written (for example, `colvarsRestartFrequency` steps), the file named:
outputName`.colvars.`*name*`.`*replicaID*`.state`
is written as well; this file contains only the state of the metadynamics bias, which the other replicas will read in turn. In between the times when this file is modified/replaced, new hills are also temporarily written to the file named:
outputName`.colvars.`*name*`.`*replicaID*`.hills`
Both files are only used for communication, and may be deleted after the replica begins writing files with a new *outputName*.

Example: Multiple-walker metadynamics with file-based communication.

```
metadynamics {
  name mymtd
  colvars x
  hillWeight 0.001
  newHillFrequency 1000
  hillWidth 3.0

  multipleReplicas on
  replicasRegistry /shared-folder/mymtd-replicas.txt
  replicaUpdateFrequency 50000 # Best if larger than newHillFrequency
}
```

The following are the multiple-walkers related options:

- `multipleReplicas` ⟨Enable multiple-walker metadynamics⟩
  Context: `metadynamics`
  Acceptable values: boolean
  Default value: `off`
  Description: This option turns on multiple-walker communication between replicas.

- `replicasRegistry` ⟨Multiple replicas database file⟩
  Context: `metadynamics`
  Acceptable values: UNIX filename
  Description: If `multipleReplicas` is on, this option sets the path to the replicas' shared database file. It is best to use an absolute path (especially when running individual replicas in separate folders).

- `replicaUpdateFrequency` ⟨How often hills are shared between replicas⟩
  Context: `metadynamics`
  Acceptable values: positive integer
  Description: If `multipleReplicas` is on, this option sets the number of steps after which each replica tries to read the other replicas' files. On a networked file system, it is best to use a number of steps that corresponds to at least a minute of wall time.

- `replicaID` ⟨Set the identifier for this replica⟩
  Context: `metadynamics`
  Acceptable values: string
  Default value: replica index (only if a shared communicator is used)
  Description: If `multipleReplicas` is `on`, this option sets a unique identifier for this replicas. When the replicas are launched in a single command (i.e. they share a parallel communicator and are tightly synchronized) this value is optional, and defaults to the replica's numeric index (starting at zero). However, when the replicas are launched as independent runs this option is required.

- `writePartialFreeEnergyFile` ⟨Periodically write the contribution to the PMF from this replica⟩
  Context: `metadynamics`
  Acceptable values: boolean
  Default value: `off`
  Description: If `multipleReplicas` is `on`, enabling this option produces an additional file *outputName*`.partial.pmf`, which can be useful to monitor the contribution of each replica to the total PMF (which is written to the file *outputName*`.pmf`). **Note:** the name of this file is chosen for consistency and convenience, *but its content is not a PMF* and it is not expected to converge, even if the total PMF does.

7.4.8 Compatibility and post-processing

The following options may be useful only for applications that go beyond the calculation of a PMF by metadynamics:

- `keepHills` ⟨Write each individual hill to the state file⟩
  Context: `metadynamics`
  Acceptable values: boolean
  Default value: `off`
  Description: When `useGrids` and this option are `on`, all hills are saved to the state file in their analytic form, alongside their grids. This makes it possible to later use exact analytic Gaussians for `rebinGrids`. To only keep track of the history of the added hills, `writeHillsTrajectory` is preferable.

- `writeHillsTrajectory` ⟨Write a log of new hills⟩
  Context: `metadynamics`
  Acceptable values: boolean
  Default value: `off`
  Description: If this option is `on`, a logfile is written by the `metadynamics` bias, with the name:
  "*outputName*`.colvars.<name>.hills.traj`",
  which can be useful to post-process the time series of the Gassian hills. When `multipleReplicas` is `on`, its name is changed to:
  "*outputName*`.colvars.<name>.<replicaID>.hills.traj`".
  The columns of this file are the centers of the hills, $\xi_i(t')$, followed by the half-widths, σ_{ξ_i}, and the weight, W. **Note:** prior to version 2020-02-24, the full-width 2σ of the Gaussian was reported in lieu of σ.

7.5 Harmonic restraints

The harmonic biasing method may be used to enforce fixed or moving restraints, including variants of Steered and Targeted MD. Within energy minimization runs, it allows for restrained minimization, e.g. to calculate relaxed potential energy surfaces. In the context of the Colvars module, harmonic potentials are meant according to their textbook definition:

$$V(\xi) = \frac{1}{2}k\left(\frac{\xi - \xi_0}{w_\xi}\right)^2 \tag{35}$$

Note that this differs from harmonic bond and angle potentials in common force fields, where the factor of one half is typically omitted, resulting in a non-standard definition of the force constant.

The formula above includes the characteristic length scale w_ξ of the colvar ξ (keyword `width`) to allow the definition of a multi-dimensional restraint with a unified force constant:

$$V(\xi_1, \ldots, \xi_M) = \frac{1}{2}k\sum_{i=1}^{M}\left(\frac{\xi_i - \xi_0}{w_\xi}\right)^2 \tag{36}$$

If one-dimensional or homogeneous multi-dimensional restraints are defined, and there are no other uses for the parameter w_ξ, `width` *can be left at its default value of* 1.

A harmonic restraint is set up by a `harmonic` { . . . } block, which may contain the following keywords:

- `name`: see definition of `name` in sec. 7 (biasing and analysis methods)
- `colvars`: see definition of `colvars` in sec. 7 (biasing and analysis methods)
- `outputEnergy`: see definition of `outputEnergy` in sec. 7 (biasing and analysis methods)
- `writeTIPMF`: see definition of `writeTIPMF` in sec. 7 (biasing and analysis methods)
- `writeTISamples`: see definition of `writeTISamples` in sec. 7 (biasing and analysis methods)
- `forceConstant` ⟨Scaled force constant (kcal/mol)⟩
  Context: `harmonic`
  Acceptable values: positive decimal
  Default value: `1.0`
  Description: This option defines a *scaled* force constant k for the harmonic potential (eq. 36). To ensure consistency for multidimensional restraints, it is divided internally by the square of the specific `width` of each variable (which is 1 by default). This makes all values effectively dimensionless and of commensurate size. For instance, if this force constant is set to the thermal energy $\kappa_B T$ (equal to RT if molar units are used), then the amplitude of the thermal fluctuations of each variable ξ will be on the order of its `width`, w_ξ. This can be used to estimate the optimal spacing of umbrella-sampling windows (under the assumption that the force constant is larger than the curvature of the underlying free energy). *The values of the actual force constants k/w_ξ^2 are always printed when the restraint is defined.*
- `centers` ⟨Initial harmonic restraint centers⟩
  Context: `harmonic`
  Acceptable values: space-separated list of colvar values
  Description: The centers (equilibrium values) of the restraint, ξ_0, are entered here. The number of values must be the number of requested colvars. Each value is a decimal number if the corresponding colvar returns a scalar, a "`(x, y, z)`" triplet if it returns a unit vector or a vector, and a "`(q0, q1, q2, q3)`" quadruplet if it returns a rotational quaternion. If a colvar has periodicities or symmetries, its closest image to the restraint center is considered when calculating the harmonic potential.

Tip: A complex set of restraints can be applied to a system, by defining several colvars, and applying one or more harmonic restraints to different groups of colvars. In some cases, dozens of colvars can be defined, but their value may not be relevant: to limit the size of the colvars trajectory file, it may be wise to disable `outputValue` for such "ancillary" variables, and leave it enabled only for "relevant" ones.

7.5.1 Moving restraints: steered molecular dynamics

The following options allow to change gradually the centers of the harmonic restraints during a simulations. When the centers are changed continuously, a steered MD in a collective variable space is carried out.

- `targetCenters` ⟨Steer the restraint centers towards these targets⟩
  Context: `harmonic`
  Acceptable values: space-separated list of colvar values
  Description: When defined, the current `centers` will be moved towards these values during the simulation. By default, the centers are moved over a total of `targetNumSteps` steps by a linear interpolation, in the spirit of Steered MD. If `targetNumStages` is set to a nonzero value, the change is performed in discrete stages, lasting `targetNumSteps` steps *each*. This second mode may be used to sample successive windows in the context of an Umbrella Sampling simulation. When continuing a simulation run, the `centers` specified in the configuration file <`colvarsConfig`> are overridden by those saved in the restart file <`colvarsInput`>. To perform Steered MD in an arbitrary space of colvars, it is sufficient to use this option and enable `outputAccumulatedWork` and/or `outputAppliedForce` within each of the colvars involved.

- `targetNumSteps` ⟨Number of steps for steering⟩
  Context: `harmonic`
  Acceptable values: positive integer
  Description: In single-stage (continuous) transformations, defines the number of MD steps required to move the restraint centers (or force constant) towards the values specified with `targetCenters` or `targetForceConstant`. After the target values have been reached, the centers (resp. force constant) are kept fixed. In multi-stage transformations, this sets the number of MD steps *per stage*.

- `outputCenters` ⟨Write the current centers to the trajectory file⟩
  Context: `harmonic`
  Acceptable values: boolean
  Default value: `off`
  Description: If this option is chosen and `colvarsTrajFrequency` is not zero, the positions of the restraint centers will be written to the trajectory file during the simulation. This option allows to conveniently extract the PMF from the colvars trajectory files in a steered MD calculation.

Note on restarting moving restraint simulations: Information about the current step and stage of a simulation with moving restraints is stored in the restart file (state file). Thus, such simulations can be run in several chunks, and restarted directly using the same colvars configuration file. In case of a restart, the values of parameters such as `targetCenters`, `targetNumSteps`, etc. should not be changed manually.

7.5.2 Moving restraints: umbrella sampling

The centers of the harmonic restraints can also be changed in discrete stages: in this cases a one-dimensional umbrella sampling simulation is performed. The sampling windows in simulation are calculated in sequence.

The colvars trajectory file may then be used both to evaluate the correlation times between consecutive windows, and to calculate the frequency distribution of the colvar of interest in each window. Furthermore, frequency distributions on a predefined grid can be automatically obtained by using the `histogram` bias (see 7.10).

To activate an umbrella sampling simulation, the same keywords as in the previous section can be used, with the addition of the following:

- `targetNumStages` ⟨Number of stages for steering⟩
  Context: `harmonic`
  Acceptable values: non-negative integer
  Default value: `0`
  Description: If non-zero, sets the number of stages in which the restraint centers or force constant are changed to their target values. If zero, the change is continuous. Each stage lasts `targetNumSteps` MD steps. To sample both ends of the transformation, the simulation should be run for `targetNumSteps` × (`targetNumStages` + 1).

7.5.3 Changing force constant

The force constant of the harmonic restraint may also be changed to equilibrate [29].

- `targetForceConstant` ⟨Change the force constant towards this value⟩
  Context: `harmonic`
  Acceptable values: positive decimal
  Description: When defined, the current `forceConstant` will be moved towards this value during the simulation. Time evolution of the force constant is dictated by the `targetForceExponent` parameter (see below). By default, the force constant is changed smoothly over a total of `targetNumSteps` steps. This is useful to introduce or remove restraints in a progressive manner. If `targetNumStages` is set to a nonzero value, the change is performed in discrete stages, lasting `targetNumSteps` steps *each*. This second mode may be used to compute the conformational free energy change associated with the restraint, within the FEP or TI formalisms. For convenience, the code provides an estimate of the free energy derivative for use in TI. A more complete free energy calculation (particularly with regard to convergence analysis), while not handled by the Colvars module, can be performed by post-processing the colvars trajectory, if `colvarsTrajFrequency` is set to a suitably small value. It should be noted, however, that restraint free energy calculations may be handled more efficiently by an indirect route, through the determination of a PMF for the restrained coordinate.[29]

- `targetForceExponent` ⟨Exponent in the time-dependence of the force constant⟩
  Context: `harmonic`
  Acceptable values: decimal equal to or greater than 1.0
  Default value: `1.0`
  Description: Sets the exponent, α, in the function used to vary the force constant as a function of time. The force is varied according to a coupling parameter λ, raised to the power α: $k_\lambda = k_0 + \lambda^\alpha (k_1 - k_0)$, where k_0, k_λ, and k_1 are the initial, current, and final values of the force constant. The parameter λ evolves linearly from 0 to 1, either smoothly, or in `targetNumStages` equally spaced discrete stages, or according to an arbitrary schedule set with `lambdaSchedule`. When the initial value of the force constant is zero, an exponent greater than 1.0 distributes the effects of introducing the restraint more smoothly over time than a linear dependence, and ensures that there is no singularity in

the derivative of the restraint free energy with respect to lambda. A value of 4 has been found to give good results in some tests.

- `targetEquilSteps` ⟨Number of steps discarded from TI estimate⟩
  Context: `harmonic`
  Acceptable values: positive integer
  Description: Defines the number of steps within each stage that are considered equilibration and discarded from the restraint free energy derivative estimate reported reported in the output.

- `lambdaSchedule` ⟨Schedule of lambda-points for changing force constant⟩
  Context: `harmonic`
  Acceptable values: list of real numbers between 0 and 1
  Description: If specified together with targetForceConstant, sets the sequence of discrete λ values that will be used for different stages.

7.6 Computing the work of a changing restraint

If the restraint centers or force constant are changed continuosly (`targetNumStages` undefined) it is possible to record the net work performed by the changing restraint:

- `outputAccumulatedWork` ⟨Write the accumulated work of the changing restraint to the Colvars trajectory file⟩
  Context: `harmonic`
  Acceptable values: boolean
  Default value: `off`
  Description: If `targetCenters` or `targetForceConstant` are defined and this option is enabled, the accumulated work from the beginning of the simulation will be written to the trajectory file (`colvarsTrajFrequency` must be non-zero). When the simulation is continued from a state file, the previously accumulated work is included in the integral. This option allows to conveniently extract the estimated PMF of a steered MD calculation (when `targetCenters` is used), or of other simulation protocols.

7.7 Harmonic wall restraints

The `harmonicWalls` {...} bias is closely related to the harmonic bias (see 7.5), with the following two differences: *(i)* instead of a center a *lower wall* and/or an *upper wall* are defined, outside of which the bias implements a half-harmonic potential;

$$V(\xi) = \begin{cases} \frac{1}{2}k\left(\frac{\xi-\xi_{\text{upper}}}{w_\xi}\right)^2 & \text{if } \xi > \xi_{\text{upper}} \\ 0 & \text{if } \xi_{\text{lower}} \leq \xi \geq \xi_{\text{upper}} \\ \frac{1}{2}k\left(\frac{\xi-\xi_{\text{lower}}}{w_\xi}\right)^2 & \text{if } \xi < \xi_{\text{lower}} \end{cases} \tag{37}$$

where ξ_{lower} and ξ_{upper} are the lower and upper wall thresholds, respectively; *(ii)* because an interval between two walls is defined, only scalar variables can be used (but any number of variables can be defined, and the wall bias is intrinsically multi-dimensional).

Note: this bias replaces the keywords `lowerWall`, `lowerWallConstant`, `upperWall` and `upperWallConstant` defined in the `colvar` context. Those keywords are deprecated.

The `harmonicWalls` bias implements the following options:

- `name`: see definition of `name` in sec. 7 (biasing and analysis methods)
- `colvars`: see definition of `colvars` in sec. 7 (biasing and analysis methods)
- `outputEnergy`: see definition of `outputEnergy` in sec. 7 (biasing and analysis methods)
- `writeTIPMF`: see definition of `writeTIPMF` in sec. 7 (biasing and analysis methods)
- `writeTISamples`: see definition of `writeTISamples` in sec. 7 (biasing and analysis methods)
- `lowerWalls` ⟨Position of the lower wall⟩
  Context: `colvar`
  Acceptable values: Space-separated list of decimals
  Description: Defines the values ξ_{lower} below which a confining restraint on the colvar is applied to each colvar ξ.
- `upperWalls` ⟨Position of the lower wall⟩
  Context: `colvar`
  Acceptable values: Space-separated list of decimals
  Description: Defines the values ξ_{upper} above which a confining restraint on the colvar is applied to each colvar ξ.
- `forceConstant`: see definition of `forceConstant` in sec. 7.5 (Harmonic restraints)
- `lowerWallConstant` ⟨Force constant for the lower wall⟩
  Context: `harmonicWalls`
  Acceptable values: positive decimal
  Default value: `forceConstant`
  Description: When both sets of walls are defined (lower and upper), this keyword allows setting different force constants for them. As with `forceConstant`, the specified constant is divided internally by the square of the specific `width` of each variable (see also the equivalent keyword for the harmonic restraint, `forceConstant`). The force constant reported in the output as "k", and used in the change of force constant scheme, is the geometric mean of `upperWallConstant` and `upperWallConstant`.
- `upperWallConstant`: analogous to `lowerWallConstant`
- `targetForceConstant`: see definition of `targetForceConstant` in sec. 7.5 (harmonic restraints)
- `targetForceConstant` ⟨Change the force constant(s) towards this value⟩
  Context: `harmonicWalls`
  Acceptable values: positive decimal
  Description: This keyword allows changing either one or both of the wall force constants over time. In the case that `lowerWallConstant` and `upperWallConstant` have the same value, the behavior of this keyword is identical to the corresponding keyword in the `harmonic` restraint; otherwise, the change schedule is applied to the geometric mean of the two constant. When only one set of walls is defined (`lowerWall` or `upperWalls`), only the respective force constant is changed. **Note:** if only one of the two force constants is meant to change over time, it is possible to use two instances of `harmonicWalls`, and apply the changing schedule only to one of them.

- `targetNumSteps`: see definition of `targetNumSteps` in sec. 7.5 (harmonic restraints)
- `targetForceExponent`: see definition of `targetForceExponent` in sec. 7.5 (harmonic restraints)
- `targetEquilSteps`: see definition of `targetEquilSteps` in sec. 7.5 (harmonic restraints)
- `targetNumStages`: see definition of `targetNumStages` in sec. 7.5 (harmonic restraints)
- `lambdaSchedule`: see definition of `lambdaSchedule` in sec. 7.5 (harmonic restraints)
- `outputAccumulatedWork`: see definition of `outputAccumulatedWork` in sec. 7.5 (harmonic restraints)
- `bypassExtendedLagrangian` ⟨Apply bias to actual colvars, bypassing extended coordinates⟩
  Context: `harmonicWalls`
  Acceptable values: boolean
  Default value: `on`
  Description: This option behaves as `bypassExtendedLagrangian` for other biases, but **it defaults to on, unlike in the general case**. Thus, by default, the `harmonicWalls` bias applies to the actual colvars, so that the distribution of the colvar between the walls is unaffected by the bias, which then applies a flat-bottom potential *as a function of the colvar value*. This bias will affect the extended coordinate distribution near the walls. If `bypassExtendedLagrangian` is disabled, `harmonicWalls` applies a flat-bottom potential *as a function of the extended coordinate*. Conversely, this bias will then modify the distribution of the actual colvar value near the walls.

Example 1: harmonic walls for one variable with two different force constants.

```
harmonicWalls {
  name mywalls
  colvars dist
  lowerWall 22.0
  upperWall 38.0
  lowerWallConstant 2.0
  upperWallConstant 10.0
}
```

Example 2: harmonic walls for two variables with a single force constant.

```
harmonicWalls {
  name mywalls
  colvars phi psi
  lowerWall -180.0 0.0
  upperWall 0.0 180.0
  forceConstant 5.0
}
```

7.8 Linear restraints

The linear restraint biasing method is used to minimally bias a simulation. There is generally a unique strength of bias for each CV center, which means you must know the bias force constant specifically for the

center of the CV. This force constant may be found by using experiment directed simulation described in section 7.9. Please cite Pitera and Chodera when using [30].

- `name`: see definition of `name` in sec. 7 (biasing and analysis methods)
- `colvars`: see definition of `colvars` in sec. 7 (biasing and analysis methods)
- `outputEnergy`: see definition of `outputEnergy` in sec. 7 (biasing and analysis methods)
- `forceConstant` ⟨Scaled force constant (kcal/mol)⟩
  Context: `linear`
  Acceptable values: positive decimal
  Default value: `1.0`
  Description: This option defines a *scaled* force constant for the linear bias. To ensure consistency for multidimensional restraints, it is divided internally by the specific `width` of each variable (which is 1 by default), so that all variables are effectively dimensionless and of commensurate size. See also the equivalent keyword for the harmonic restraint, `forceConstant`. *The values of the actual force constants k/w_ξ are always printed when the restraint is defined.*
- `centers` ⟨Initial linear restraint centers⟩
  Context: `linear`
  Acceptable values: space-separated list of colvar values
  Description: These are analogous to the `centers` keyword of the harmonic restraint. Although they do not affect dynamics, they are here necessary to ensure a well-defined energy for the linear bias.
- `writeTIPMF`: see definition of `writeTIPMF` in sec. 7 (biasing and analysis methods)
- `writeTISamples`: see definition of `writeTISamples` in sec. 7 (biasing and analysis methods)
- `targetForceConstant`: see definition of `targetForceConstant` in sec. 7.5 (Harmonic restraints)
- `targetNumSteps`: see definition of `targetNumSteps` in sec. 7.5 (Harmonic restraints)
- `targetForceExponent`: see definition of `targetForceExponent` in sec. 7.5 (Harmonic restraints)
- `targetEquilSteps`: see definition of `targetEquilSteps` in sec. 7.5 (Harmonic restraints)
- `targetNumStages`: see definition of `targetNumStages` in sec. 7.5 (Harmonic restraints)
- `lambdaSchedule`: see definition of `lambdaSchedule` in sec. 7.5 (Harmonic restraints)
- `outputAccumulatedWork`: see definition of `outputAccumulatedWork` in sec. 7.5 (Harmonic restraints)

7.9 Adaptive Linear Bias/Experiment Directed Simulation

Experiment directed simulation applies a linear bias with a changing force constant. Please cite White and Voth [31] when using this feature. As opposed to that reference, the force constant here is scaled by the `width` corresponding to the biased colvar. In White and Voth, each force constant is scaled by the colvars set center. The bias converges to a linear bias, after which it will be the minimal possible bias. You may also stop the simulation, take the median of the force constants (ForceConst) found in the colvars trajectory

file, and then apply a linear bias with that constant. All the notes about units described in sections 7.8 and 7.5 apply here as well. **This is not a valid simulation of any particular statistical ensemble and is only an optimization algorithm until the bias has converged**.

- `name`: see definition of `name` in sec. 7 (biasing and analysis methods)
- `colvars`: see definition of `colvars` in sec. 7 (biasing and analysis methods)
- `centers` ⟨Collective variable centers⟩
  Context: `alb`
  Acceptable values: space-separated list of colvar values
  Description: The desired center (equilibrium values) which will be sought during the adaptive linear biasing. The number of values must be the number of requested colvars. Each value is a decimal number if the corresponding colvar returns a scalar, a "`(x, y, z)`" triplet if it returns a unit vector or a vector, and a "`q0, q1, q2, q3)`" quadruplet if it returns a rotational quaternion. If a colvar has periodicities or symmetries, its closest image to the restraint center is considered when calculating the linear potential.
- `updateFrequency` ⟨The duration of updates⟩
  Context: `alb`
  Acceptable values: An integer
  Description: This is, N, the number of simulation steps to use for each update to the bias. This determines how long the system requires to equilibrate after a change in force constant ($N/2$), how long statistics are collected for an iteration ($N/2$), and how quickly energy is added to the system (at most, $A/2N$, where A is the `forceRange`). Until the force constant has converged, the method as described is an optimization procedure and not an integration of a particular statistical ensemble. It is important that each step should be uncorrelated from the last so that iterations are independent. Therefore, N should be at least twice the autocorrelation time of the collective variable. The system should also be able to dissipate energy as fast as $N/2$, which can be done by adjusting thermostat parameters. Practically, N has been tested successfully at significantly shorter than the autocorrelation time of the collective variables being biased and still converge correctly.
- `forceRange` ⟨The expected range of the force constant in units of energy⟩
  Context: `alb`
  Acceptable values: A space-separated list of decimal numbers
  Default value: 3 k_bT
  Description: This is largest magnitude of the force constant which one expects. If this parameter is too low, the simulation will not converge. If it is too high the simulation will waste time exploring values that are too large. A value of 3 k_bT has worked well in the systems presented as a first choice. This parameter is dynamically adjusted over the course of a simulation. The benefit is that a bad guess for the forceRange can be corrected. However, this can lead to large amounts of energy being added over time to the system. To prevent this dynamic update, add `hardForceRange yes` as a parameter
- `rateMax` ⟨The maximum rate of change of force constant⟩
  Context: `alb`
  Acceptable values: A list of space-separated real numbers
  Description: This optional parameter controls how much energy is added to the system from this bias. Tuning this separately from the `updateFrequency` and `forceRange` can allow for large bias changes but with a low `rateMax` prevents large energy changes that can lead to instability in the simulation.

7.10 Multidimensional histograms

The `histogram` feature is used to record the distribution of a set of collective variables in the form of a N-dimensional histogram. A `histogram` block may define the following parameters:

- `name`: see definition of `name` in sec. 7 (biasing and analysis methods)
- `colvars`: see definition of `colvars` in sec. 7 (biasing and analysis methods)
- `outputFreq` ⟨Frequency (in timesteps) at which the histogram files are refreshed⟩
  Context: `histogram`
  Acceptable values: positive integer
  Default value: `colvarsRestartFrequency`
  Description: The histogram data are written to files at the given time interval. A value of 0 disables the creation of these files (**note:** all data to continue a simulation are still included in the state file).
- `outputFile` ⟨Write the histogram to a file⟩
  Context: `histogram`
  Acceptable values: UNIX filename
  Default value: *outputName*.<name>.dat
  Description: Name of the file containing histogram data (multicolumn format), which is written every `outputFreq` steps. For the special case of 2 variables, Gnuplot may be used to visualize this file.
- `outputFileDX` ⟨Write the histogram to a file⟩
  Context: `histogram`
  Acceptable values: UNIX filename
  Default value: *outputName*.<name>.dat
  Description: Name of the file containing histogram data (OpenDX format), which is written every `outputFreq` steps. For the special case of 3 variables, VMD may be used to visualize this file.
- `gatherVectorColvars` ⟨Treat vector variables as multiple observations of a scalar variable?⟩
  Context: `histogram`
  Acceptable values: UNIX filename
  Default value: `off`
  Description: When this is set to on, the components of a multi-dimensional colvar (e.g. one based on `cartesian`, `distancePairs`, or a vector of scalar numbers given by `scriptedFunction`) are treated as multiple observations of a scalar variable. This results in the histogram being accumulated multiple times for each simulation step or iteration of `cv update`). When multiple vector variables are included in `histogram`, these must have the same length because their components are accumulated together. For example, if ξ, λ and τ are three variables of dimensions 5, 5 and 1, respectively, for each iteration 5 triplets (ξ_i, λ_i, τ) $(i = 1, \ldots 5)$ are accumulated into a 3-dimensional histogram.
- `weights` ⟨Treat vector variables as multiple observations of a scalar variable?⟩
  Context: `histogram`
  Acceptable values: list of space-separated decimals
  Default value: all weights equal to 1
  Description: When `gatherVectorColvars` is on, the components of each multi-dimensional colvar

are accumulated with a different weight. For example, if x and y are two distinct `cartesian` variables defined on the same group of atoms, the corresponding 2D histogram can be weighted on a per-atom basis: to compute an electron density map, it is possible to use `weights [$sel get atomicnumber]` in the definition of `histogram`.

As with any other biasing and analysis method, when a histogram is applied to an extended-system colvar (5.18), it accesses the value of the extended coordinate rather than that of the actual colvar. This can be overridden by enabling the `bypassExtendedLagrangian` option. A *joint histogram* of the actual colvar and the extended coordinate may be collected by specifying the colvar name twice in a row in the `colvars` parameter (e.g. `colvars myColvar myColvar`): the first instance will be understood as the actual colvar, and the second, as the extended coordinate.

- `bypassExtendedLagrangian`: see definition of `bypassExtendedLagrangian` in sec. 7 (biasing and analysis methods)

7.10.1 Grid definition for multidimensional histograms

Like the ABF and metadynamics biases, `histogram` uses the parameters `lowerBoundary`, `upperBoundary`, and `width` to define its grid. These values can be overridden if a configuration block `histogramGrid { ... }` is provided inside the configuration of `histogram`. The options supported inside this configuration block are:

- `lowerBoundaries` ⟨Lower boundaries of the grid⟩
  Context: `histogramGrid`
  Acceptable values: list of space-separated decimals
  Description: This option defines the lower boundaries of the grid, overriding any values defined by the `lowerBoundary` keyword of each colvar. Note that when `gatherVectorColvars` is `on`, each vector variable is automatically treated as a scalar, and a single value should be provided for it.

- `upperBoundaries`: analogous to `lowerBoundaries`

- `widths`: analogous to `lowerBoundaries`

7.11 Probability distribution-restraints

The `histogramRestraint` bias implements a continuous potential of many variables (or of a single high-dimensional variable) aimed at reproducing a one-dimensional statistical distribution that is provided by the user. The M variables $(\xi_1, \ldots, \xi_M)$ are interpreted as multiple observations of a random variable ξ with unknown probability distribution. The potential is minimized when the histogram $h(\xi)$, estimated as a sum of Gaussian functions centered at $(\xi_1, \ldots, \xi_M)$, is equal to the reference histogram $h_0(\xi)$:

$$V(\xi_1, \ldots, \xi_M) = \frac{1}{2} k \int (h(\xi) - h_0(\xi))^2 \, \mathrm{d}\xi \tag{38}$$

$$h(\xi) = \frac{1}{M\sqrt{2\pi\sigma^2}} \sum_{i=1}^{M} \exp\left(-\frac{(\xi - \xi_i)^2}{2\sigma^2} \right) \tag{39}$$

When used in combination with a `distancePairs` multi-dimensional variable, this bias implements the refinement algorithm against ESR/DEER experiments published by Shen *et al* [32].

This bias behaves similarly to the `histogram` bias with the `gatherVectorColvars` option, with the important difference that *all* variables are gathered, resulting in a one-dimensional histogram. Future versions will include support for multi-dimensional histograms.

The list of options is as follows:

- `name`: see definition of `name` in sec. 7 (biasing and analysis methods)
- `colvars`: see definition of `colvars` in sec. 7 (biasing and analysis methods)
- `outputEnergy`: see definition of `outputEnergy` in sec. 7 (biasing and analysis methods)
- `lowerBoundary` ⟨Lower boundary of the colvar grid⟩
  Context: `histogramRestraint`
  Acceptable values: decimal
  Description: Defines the lowest end of the interval where the reference distribution $h_0(\xi)$ is defined. Exactly one value must be provided, because only one-dimensional histograms are supported by the current version.
- `upperBoundary`: analogous to `lowerBoundary`
- `width` ⟨Width of the colvar grid⟩
  Context: `histogramRestraint`
  Acceptable values: positive decimal
  Description: Defines the spacing of the grid where the reference distribution $h_0(\xi)$ is defined.
- `gaussianSigma` ⟨Standard deviation of the approximating Gaussian⟩
  Context: `histogramRestraint`
  Acceptable values: positive decimal
  Default value: $2 \times$ `width`
  Description: Defines the parameter σ in eq. 39.
- `forceConstant` ⟨Force constant (kcal/mol)⟩
  Context: `histogramRestraint`
  Acceptable values: positive decimal
  Default value: `1.0`
  Description: Defines the parameter k in eq. 38.
- `refHistogram` ⟨Reference histogram $h_0(\xi)$⟩
  Context: `histogramRestraint`
  Acceptable values: space-separated list of M positive decimals
  Description: Provides the values of $h_0(\xi)$ consecutively. The mid-point convention is used, i.e. the first point that should be included is for ξ = `lowerBoundary+width/2`. If the integral of $h_0(\xi)$ is not normalized to 1, $h_0(\xi)$ is rescaled automatically before use.
- `refHistogramFile` ⟨Reference histogram $h_0(\xi)$⟩
  Context: `histogramRestraint`
  Acceptable values: UNIX file name
  Description: Provides the values of $h_0(\xi)$ as contents of the corresponding file (mutually exclusive with `refHistogram`). The format is that of a text file, with each line containing the space-separated values of ξ and $h_0(\xi)$. The same numerical conventions as `refHistogram` are used.

- `writeHistogram` ⟨Periodically write the instantaneous histogram $h(\xi)$⟩
  Context: `metadynamics`
  Acceptable values: boolean
  Default value: `off`
  Description: If `on`, the histogram $h(\xi)$ is written every `colvarsRestartFrequency` steps to a file with the name *outputName*.<`name`>`.hist.dat` This is useful to diagnose the convergence of $h(\xi)$ against $h_0(\xi)$.

7.12 Defining scripted biases

Rather than using the biasing methods described above, it is possible to apply biases provided at run time as a Tcl script. This option, also available in NAMD, can be useful to test a new algorithm to be used in a MD simulation.

- `scriptedColvarForces` ⟨Enable custom, scripted forces on colvars ⟩
  Context: global
  Acceptable values: boolean
  Default value: `off`
  Description: If this flag is enabled, a Tcl procedure named `calc_colvar_forces` accepting one parameter should be defined by the user. It is executed at each timestep, with the current step number as parameter, between the calculation of colvars and the application of bias forces. This procedure may use the `cv` command to access the values of colvars (e.g. `cv colvar xi value`), apply forces on them (`cv colvar xi addforce $F`) or add energy to the simulation system (`cv addenergy $E`), effectively defining custom collective variable biases.

7.13 Performance of scripted biases

If concurrent computation over multiple threads is available (this is indicated by the message "SMP parallelism is available." printed at initialization time), it is useful to take advantage of the scripting interface to combine many components, all computed in parallel, into a single variable.

The default SMP schedule is the following:

1. distribute the computation of all components across available threads;
2. on a single thread, collect the results of multi-component variables using polynomial combinations (see 5.14), or scripted functions (see 5.15);
3. distribute the computation of all biases across available threads;
4. compute on a single thread any scripted biases implemented via the keyword `scriptedColvarForces`.
5. communicate on a single thread forces to VMD.

The following options allow to fine-tune this schedule:

- `scriptingAfterBiases` ⟨Scripted colvar forces need updated biases?⟩
  Context: global
  Acceptable values: boolean
  Default value: `on`
  Description: This flag specifies that the `calc_colvar_forces` procedure (last step in the list above) is executed only after all biases have been updated (next-to-last step) For example, this allows using the energy of a restraint bias, or the force applied on a colvar, to calculate additional scripted forces, such as boundary constraints. When this flag is set to `off`, it is assumed that only the values of the variables (but not the energy of the biases or applied forces) will be used by `calc_colvar_forces`: this can be used to schedule the calculation of scripted forces and biases concurrently to increase performance.

8 Scripting interface (Tcl): list of commands

This section lists all the commands used in VMD to control the behavior of the Colvars module from within a run script.

8.1 Commands to manage the Colvars module

- cv addenergy E
  Add an energy to the MD engine (no effect in VMD)
  Parameters
  E : float - Amount of energy to add

- cv config conf
  Read configuration from the given string
  Parameters
  conf : string - Configuration string

- cv configfile conf_file
  Read configuration from a file
  Parameters
  conf_file : string - Path to configuration file

- cv delete
  Delete this Colvars module instance (VMD only)

- cv frame [frame]
  Get or set current frame number (VMD only)
  Parameters
  frame : integer - Frame number (optional)

- cv getconfig
  Get the module's configuration string read so far

- cv getenergy
  Get the current Colvars energy

- cv help [command]
  Get the help string of the Colvars scripting interface
  Parameters
  command : string - Get the help string of this specific command (optional)

- cv list [param]
  Return a list of all variables or biases
  Parameters
  param : string - "colvars" or "biases"; default is "colvars" (optional)

- cv listcommands
  Get the list of script functions, prefixed with "cv_", "colvar_" or "bias_"

- cv load prefix
  Load data from a state file into all matching colvars and biases
  Parameters
  prefix : string - Path to existing state file or input prefix

- cv molid [molid]
  Get or set the molecule ID on which Colvars is defined (VMD only)
  Parameters
  molid : integer - Molecule ID; -1 means undefined (optional)

- cv printframe
  Return the values that would be written to colvars.traj

- cv printframelabels
  Return the labels that would be written to colvars.traj

- cv reset
  Delete all internal configuration

- cv resetindexgroups
  Clear the index groups loaded so far, allowing to replace them

- cv save prefix
  Change the prefix of all output files and save them
  Parameters
  prefix : string - Output prefix with trailing ".colvars.state" gets removed)

- cv units [units]
  Get or set the current Colvars unit system
  Parameters
  units : string - The new unit system (optional)

- cv update
  Recalculate colvars and biases

- cv version
  Get the Colvars Module version number

8.2 Commands to manage individual collective variables

- cv colvar name addforce force
  Apply the given force onto this colvar and return the same
  Parameters
  force : float or array - Applied force; must match colvar dimensionality

- cv colvar name cvcflags flags
  Enable or disable individual components by setting their active flags
  Parameters
  flags : integer array - Zero/nonzero value disables/enables the CVC

- cv colvar name delete
  Delete this colvar, along with all biases that depend on it

- cv colvar name get feature
  Get the value of the given feature for this colvar
  Parameters
  feature : string – Name of the feature

- cv colvar name getappliedforce
  Return the total of the forces applied to this colvar

- cv colvar name getatomgroups
  Return the atom indices used by this colvar as a list of lists

- cv colvar name getatomids
  Return the list of atom indices used by this colvar

- cv colvar name getconfig
  Return the configuration string of this colvar

- cv colvar name getgradients
  Return the atomic gradients of this colvar

- cv colvar name gettotalforce
  Return the sum of internal and external forces to this colvar

- cv colvar name help [command]
  Get a help summary or the help string of one colvar subcommand
  Parameters
  command : string – Get the help string of this specific command (optional)

- cv colvar name modifycvcs confs
  Modify configuration of individual components by passing string arguments
  Parameters
  confs : sequence of strings – New configurations; empty strings are skipped

- cv colvar name run_ave
  Get the current running average of the value of this colvar

- cv colvar name set feature value
  Set the given feature of this colvar to a new value
  Parameters
  feature : string – Name of the feature
  value : string – String representation of the new feature value

- cv colvar name state
  Print a string representation of the feature state of this colvar

- cv colvar name type
  Get the type description of this colvar

- cv colvar name update
  Recompute this colvar and return its up-to-date value

- cv colvar name value
  Get the current value of this colvar

- cv colvar name width
  Get the width of this colvar

8.3 Commands to manage individual biases

- cv bias name bin
  Get the current grid bin index (1D ABF only for now)

- cv bias name bincount [index]
  Get the number of samples at the given grid bin (1D ABF only for now)
  Parameters
  index : integer – Grid index; defaults to current bin (optional)

- cv bias name binnum
  Get the total number of grid points of this bias (1D ABF only for now)

- cv bias name delete
  Delete this bias

- cv bias name energy
  Get the current energy of this bias

- cv bias name get feature
  Get the value of the given feature for this bias
  Parameters
  feature : string – Name of the feature

- cv bias name getconfig
  Return the configuration string of this bias

- cv bias name help [command]
  Get a help summary or the help string of one bias subcommand
  Parameters
  command : string – Get the help string of this specific command (optional)

- cv bias name load prefix
  Load data into this bias
  Parameters
  prefix : string – Read from a file with this name or prefix

- cv bias name save prefix
  Save data from this bias into a file with the given prefix
  Parameters
  prefix : string – Prefix for the state file of this bias

- cv bias name set feature value
  Set the given feature of this bias to a new value
  Parameters
  feature : string – Name of the feature
  value : string – String representation of the new feature value

- cv bias name share
  Share bias information with other replicas (multiple-walker scheme)

- cv bias name state
  Print a string representation of the feature state of this bias

- cv bias name update
  Recompute this bias and return its up-to-date energy

9 Syntax changes from older versions

The following is a list of syntax changes in Colvars since its first release. Many of the older keywords are still recognized by the current code, thanks to specific compatibility code. *This is not a list of new features:* its primary purpose is to make you aware of those improvements that affect the use of old configuration files with new versions of the code.
Note: if you are using any of the NAMD and VMD tutorials:
https://www.ks.uiuc.edu/Training/Tutorials/
please be aware that *several of these tutorials are not actively maintained*: for those cases, this list will help you reconcile any inconsistencies.

- **Colvars version 2016-06-09 or later (VMD version 1.9.3 or later).**
  The legacy keyword `refPositionsGroup` has been renamed `fittingGroup` for clarity (the legacy version is still supported).

- **Colvars version 2016-08-10 or later (VMD version 1.9.3 or later).**
  "System forces" have been replaced by "total forces" (see for example `outputTotalForce`). See the following page for more information:
  https://colvars.github.io/README-totalforce.html

- **Colvars version 2017-01-09 or later (VMD version 1.9.4 or later).**
  A new type of restraint, `harmonicWalls` (see 7.7), replaces and improves upon the legacy keywords `lowerWall` and `upperWall`: these are still supported as short-hands.

- **Colvars version 2018-11-15 or later (VMD version 1.9.4 or later).**
  The global `analysis` keyword has been discontinued: specific analysis tasks are controlled directly by the keywords `corrFunc` and `runAve`, which continue to remain `off` by default.

- **Colvars version 2020-02-25 or later (VMD version 1.9.4 or later).**
  The parameter `hillWidth`, expressing the Gaussian width 2σ in relative units (number of grid points), does not have a default value any more. A new alternative parameter `gaussianSigmas` allows setting the σ parameters explicitly for each variable if needed.
  Furthermore, to facilitate the use of other analysis tools such as for example `sum_hills`:
  https://www.plumed.org/doc-v2.6/user-doc/html/sum_hills.html
  the format of the file written by `writeHillsTrajectory` has also been changed to use σ instead of 2σ. This change does not affect how the biasing potential is written in the state file, or the simulated trajectory.

- **Colvars version 2020-02-25 or later (VMD version 1.9.4 or later).**
  The legacy keywords `lowerWall` and `upperWall` of a `colvar` definition block do not have default values any longer, and need to be set explicitly, preferably as part of the `harmonicWalls` restraint. When using an ABF bias, it is recommended to set the two walls equal to `lowerBoundary` and `upperBoundary`, respectively. When using a metadynamics bias, it is recommended to set the two walls strictly *within* `lowerBoundary` and `upperBoundary`; see 7.4.1 for details.

Up-to-date documentation can always be accessed at:
https://colvars.github.io/colvars-refman-vmd/colvars-refman-vmd.html

10 Compilation notes

The Colvars module is typically built using the recipes of each supported software package: for this reason, no installation instructions are needed, and the vast majority of the features described in this manual are supported in the most common builds of each package. This section lists the few cases where the choice of compilation settings affects features in the Colvars module.

- Scripting commands using the Tcl language (https://www.tcl.tk) are supported in VMD and NAMD. All precompiled builds of each code include Tcl, and it is highly recommended to enable Tcl support in any custom build, using precompiled Tcl libraries from the UIUC website.
- The Lepton library (https://simtk.org/projects/lepton) used to implement the `customFunction` feature is currently included only in NAMD (always on) and in LAMMPS (on by default).
- Some features require compilation using the C++11 language standard. Although it is becoming commonplace, this standard is not yet available on all scientific computing systems. Deailed information can be found at:
  https://colvars.github.io/README-c++11.html

Index

References

[1] G. Fiorin, M. L. Klein, and J. Hénin. Using collective variables to drive molecular dynamics simulations. *Mol. Phys.*, 111(22-23):3345–3362, 2013. 4, 72, 74, 78

[2] M. Iannuzzi, A. Laio, and M. Parrinello. Efficient exploration of reactive potential energy surfaces using car-parrinello molecular dynamics. *Phys. Rev. Lett.*, 90(23):238302, 2003. 25, 52

[3] E A Coutsias, C Seok, and K A Dill. Using quaternions to calculate RMSD. *J. Comput. Chem.*, 25(15):1849–1857, 2004. 28, 32, 59

[4] Yuguang Mu, Phuong H. Nguyen, and Gerhard Stock. Energy landscape of a small peptide revealed by dihedral angle principal component analysis. *Proteins*, 58(1):45–52, 2005. 36

[5] Alexandros Altis, Phuong H. Nguyen, Rainer Hegger, and Gerhard Stock. Dihedral angle principal component analysis of molecular dynamics simulations. *J. Chem. Phys.*, 126(24):244111, 2007. 36

[6] Nicholas M Glykos. Carma: a molecular dynamics analysis program. *J. Comput. Chem.*, 27(14):1765–1768, 2006. 36, 37

[7] G. D. Leines and B. Ensing. Path finding on high-dimensional free energy landscapes. *Phys. Rev. Lett.*, 109:020601, 2012. 38

[8] Davide Branduardi, Francesco Luigi Gervasio, and Michele Parrinello. From a to b in free energy space. *J Chem Phys*, 126(5):054103, 2007. 38, 42, 44, 47

[9] F. Comitani L. Hovan and F. L. Gervasio. Defining an optimal metric for the path collective variables. *J. Chem. Theory Comput.*, 15:25–32, 2019. 42

[10] Marco Jacopo Ferrarotti, Sandro Bottaro, Andrea Pérez-Villa, and Giovanni Bussi. Accurate multiple time step in biased molecular simulations. *Journal of chemical theory and computation*, 11:139–146, 2015. 53

[11] Reza Salari, Thomas Joseph, Ruchi Lohia, Jérôme Hénin, and Grace Brannigan. A streamlined, general approach for computing ligand binding free energies and its application to GPCR-bound cholesterol. *Journal of Chemical Theory and Computation*, 14(12):6560–6573, 2018. 60

[12] Eric Darve, David Rodríguez-Gómez, and Andrew Pohorille. Adaptive biasing force method for scalar and vector free energy calculations. *J. Chem. Phys.*, 128(14):144120, 2008. 64

[13] J. Hénin and C. Chipot. Overcoming free energy barriers using unconstrained molecular dynamics simulations. *J. Chem. Phys.*, 121:2904–2914, 2004. 64

[14] J. Hénin, G. Fiorin, C. Chipot, and M. L. Klein. Exploring multidimensional free energy landscapes using time-dependent biases on collective variables. *J. Chem. Theory Comput.*, 6(1):35–47, 2010. 64

[15] A. Carter, E, G. Ciccotti, J. T. Hynes, and R. Kapral. Constrained reaction coordinate dynamics for the simulation of rare events. *Chem. Phys. Lett.*, 156:472–477, 1989. 65

[16] M. J. Ruiz-Montero, D. Frenkel, and J. J. Brey. Efficient schemes to compute diffusive barrier crossing rates. *Mol. Phys.*, 90:925–941, 1997. 65

[17] W. K. den Otter. Thermodynamic integration of the free energy along a reaction coordinate in cartesian coordinates. *J. Chem. Phys.*, 112:7283–7292, 2000. 65

[18] Giovanni Ciccotti, Raymond Kapral, and Eric Vanden-Eijnden. Blue moon sampling, vectorial reaction coordinates, and unbiased constrained dynamics. *ChemPhysChem*, 6(9):1809–1814, 2005. 65

[19] Adrien Lesage, Tony Lelièvre, Gabriel Stoltz, and Jérôme Hénin. Smoothed biasing forces yield unbiased free energies with the extended-system adaptive biasing force method. *J. Phys. Chem. B*, 121(15):3676–3685, 2017. 70

[20] A. Laio and M. Parrinello. Escaping free-energy minima. *Proc. Natl. Acad. Sci. USA*, 99(20):12562–12566, 2002. 71

[21] Helmut Grubmüller. Predicting slow structural transitions in macromolecular systems: Conformational flooding. *Phys. Rev. E*, 52(3):2893–2906, Sep 1995. 71

[22] T. Huber, A. E. Torda, and W.F. van Gunsteren. Local elevation - A method for improving the searching properties of molecular-dynamics simulation. *Journal of Computer-Aided Molecular Design*, 8(6):695–708, DEC 1994. 71

[23] G. Bussi, A. Laio, and M. Parrinello. Equilibrium free energies from nonequilibrium metadynamics. *Phys. Rev. Lett.*, 96(9):090601, 2006. 72

[24] Fabrizio Marinelli, Fabio Pietrucci, Alessandro Laio, and Stefano Piana. A kinetic model of trp-cage folding from multiple biased molecular dynamics simulations. *PLOS Computational Biology*, 5(8):1–18, 2009. 72

[25] Yanier Crespo, Fabrizio Marinelli, Fabio Pietrucci, and Alessandro Laio. Metadynamics convergence law in a multidimensional system. *Phys. Rev. E*, 81:055701, May 2010. 72

[26] Fabrizio Marinelli and José D. Faraldo-Gómez. Ensemble-biased metadynamics: A molecular simulation method to sample experimental distributions. *Biophysical Journal*, 108(12):2779 – 2782, 2015. 76

[27] Alessandro Barducci, Giovanni Bussi, and Michele Parrinello. Well-tempered metadynamics: A smoothly converging and tunable free-energy method. *Phys. Rev. Lett.*, 100:020603, 2008. 78

[28] P. Raiteri, A. Laio, F. L. Gervasio, C. Micheletti, and M. Parrinello. Efficient reconstruction of complex free energy landscapes by multiple walkers metadynamics. *J. Phys. Chem. B*, 110(8):3533–9, 2006. 78

[29] Yuqing Deng and Benoît Roux. Computations of standard binding free energies with molecular dynamics simulations. *J. Phys. Chem. B*, 113(8):2234–2246, 2009. 83

[30] Jed W. Pitera and John D. Chodera. On the use of experimental observations to bias simulated ensembles. *J. Chem. Theory Comput.*, 8:3445–3451, 2012. 87

[31] A. D. White and G. A. Voth. Efficient and minimal method to bias molecular simulations with experimental data. *J. Chem. Theory Comput.*, ASAP, 2014. 87

[32] Rong Shen, Wei Han, Giacomo Fiorin, Shahidul M Islam, Klaus Schulten, and Benoît Roux. Structural refinement of proteins by restrained molecular dynamics simulations with non-interacting molecular fragments. *PLoS Comput. Biol.*, 11(10):e1004368, 2015. 91