PF2 Manual

20120824

Contents:

1. Installation

1.1. Installation of GROMACS-PF2

The current version of GROMACS-PF2 is based on GROMACS 4.5.3. As PF2 is entirely contained in the **mdrun** executable, there is no need to perform a full GROMACS installation:

```
./configure --program-suffix _pf2 <same flags as for normal
GROMACS>
make mdrun
make install-mdrun
```

This will generate an executable called **mdrun_pf2** which contains all the functionality described in this document. To compile a double precision version:

```
./configure --enable-double --program-suffix _pf2_d <same flags as
for normal GROMACS>
make mdrun
make install-mdrun
```

which will generate an executable called **mdrun_pf2_d**. In the rest of this document only **mdrun_pf2** (default name for a single precision version) is mentioned, but everything applies to the double precision version as well.

1.2. Installation of VMD visualization plugins

The installation is done by unpacking the archive into the VMD plugins directory; normally this is plugins/noarch/tcl. This operation will create 2 directories called pf_loaduser1.0 and pf_draw1.0 which contain the corresponding Tcl files. To test the installation, start VMD then at the console or in the Tk console type:

```
package require pf_loaduser
package require pf_draw
```

2. Principles

2.1. Residue-based operation

For residue-based operation, the following steps are done in order:

1. building of an atom to residue correspondence table
2. if an atom pair is part of those selected for pairwise forces calculations, the residue numbers of the 2 atoms are obtained from the correspondence table
3. if the 2 residue numbers are equal (2 different atoms from the same residue), nothing is done because it makes no sense to calculate the interaction of a residue with itself and the code goes back to step 2 with a different atom pair
4. the atomic vector force is added to the vector force corresponding to the residue pair

So in the end, each residue pair will have a vector force calculated as:

$$\vec{F}_{ri,rj} = \sum_{i \in ri,\, j \in rj} \vec{F}_{ij}$$

where *i* is an atom of residue *ri* and *j* is an atom of residue *rj*, with *ri* and *rj* being different.
One calculated, the residue-based pairwise forces can be written to output files just like the atom-based ones – as vector or scalar values – or can be further used to calculate a per-residue sum/average/minimum/maximum value.

2.2. Residue (re)numbering

For GROMACS, residues are of secondary importance during the actual MD run – they are useful only to write out a structure file (f.e. PDB). The correspondence between numbering of atoms and residues is made based on the information in the structure file where residues numbering might not start from 1 and might not be contiguous – to deal with all these cases, by default GROMACS doesn't make any attempt to renumber residues and writes back whatever residue number was assigned to an atom in the input structure file.
The PF2 code uses these residue numbers to identify which atoms belong to which residues and to output residue-based data. Dealing with residue numbers not starting from 1 or not being contiguous is easy – just fill in the missing numbers starting from 1 and have the output contain zeros for them. But residue numbers can also appear multiple times, f.e. if the structure file contains several chains (say a protein, a ligand and solvent) and in each of them residues start being numbered at 1 or when there are many residues (more than 9999 which is the maximum number in a PDB file) and their number wraps around. To get around this, GROMACS can do residue renumbering which numbers residues with increasing numbers starting from 1. The PF2 code can detect residue number collisions and, by default, automatically switches to renumbering residues; this behavior can be controlled through a .pfi file option.
Renumbering residues influences both the way the residue data is stored in memory and the way it is written in the output files. The information whether renumbering has taken place in the PF2 code (be it automatically or enforced through the .pfi file option) has to be passed on to the VMD plugin; failure to do so will result in wrong representations !

2.3. Pairwise forces decomposition for 3- and 4-body potentials

For potentials involving only two atoms (bonds, Coulomb, Lennard-Jones), the pairwise force is the same as the atomic force, directly calculated from the potential. For potentials involving more than two atoms (angles, dihedral angles – proper and improper, as well as cross bond-bond and cross bond-angle), the atomic forces need to be decomposed to reflect the interactions between each two atoms. The vector sum of pairwise forces which result from this decomposition should equal the atomic forces. The decomposition implemented in the PF2 code is detailed in Appendix A.

3. Running

3.1. Additional mdrun options

PF2 adds a few command line options to **mdrun**:

- -pfi input.pfi – specifies an input file which controls how the code should run
- -pfn index.ndx – is an index file containing one or more groups of atoms which should be used for pairwise forces calculations
- -pfa output.pfa – output file, containing pairwise forces or punctual stress for atoms; the content and format depend on the choices in the .pfi file
- -pfr output.pfr – output file, containing pairwise forces or punctual stress for residues; the content and format depend on the choices in the .pfi file

mdrun_pf2 should always be run with the -rerun option, requiring that an initial trajectory is produced beforehand; apart from the PF2 specific options described above, the options passed to **mdrun_pf2** should be the same as the ones used when producing the original trajectory, with one exception: a single CPU run should be enforced, as the PF2 code is not (yet) parallel. Please note that a non-MPI version of GROMACS 4.5 runs by default as threaded which still counts as parallel, so use "-nt 1" to enforce a non-parallel run:

```
mdrun_pf2 -nt 1 \
     -rerun traj.trr \
     -pfi input.pfi \
     -pfn index.ndx \
     -pfa output.pfa \
     -pfr output.pfr \
     <other options>
```

3.2. The input file (.pfi)

The input file controls how the PF2 code should run; it also specifies the content and format of the output files (.pfa and .pfr). Like the .mdp file which GROMACS uses as input, the .pfi file is a simple text file containing statements of form "keyword = value", with comments started by a semi-colon (;). All keywords start with pf_. The .pfi file is read directly by **mdrun_pf2**, there is no need for a step equivalent to **grompp**.

The .pfi file keywords are:

`pf_onepair` – describes how the pairwise forces between the same pair of atoms *i* and *j* is handled. Typically, a forcefield can allow either several types of non-bonded interactions or several types of bonded interactions between the same pair of atoms. F.e. two distant atoms could interact through both Coulomb and Lennard-Jones interactions, while two bonded atoms could interact through a bond, several angle and several dihedral angle interactions. There are 2 possible values for this keyword:
`summed` – all pairwise forces between the same pair of atoms are summed; if atoms *i* and *j* interact,

the code stores exactly one value of the pairwise force between them
`detailed` – all pairwise forces of the same type (f.e. angle interactions) between the same pair of atoms are stored separately; if atoms *i* and *j* interact, the code stores at least one, possibly many, pairwise forces between them, corresponding to each type of interaction that exists between them.

Using `detailed` can lead to a significant increase in the amount of memory needed compared with the `summed` option. `detailed` should probably only be used in the rare cases where simultaneous output of pairwise forces separated by potential type should be obtained, like forcefield development/debugging. The default is `summed`. For example, the following set of options will compute a sum of all interactions of the same type (say dihedral angles) between the atoms *i* and *j*:

```
pf_onepair = detailed
pf_type = dihedral
```

`pf_group1` and **`pf_group2`** – name the groups which will be used for pairwise forces calculations; if not present, they default to "Protein". The 2 groups are equivalent, there is no difference in how they are treated internally. The groups names should exist in the index file specified with the -pfn command line option; it's not necessary to have a separate index file for this purpose, different from the one used for the **grompp** step of the original run – only the groups named in pf_group1 and pf_group2 will be loaded, other groups will be ignored. The 2 groups define which atoms are used for pairwise forces calculations: only interactions for which an atom is part of pf_group1 and the other atom is part of pf_group2 are being considered. It's fine to have one or more atoms in both groups at the same time. It's also fine to have only one group defined in the index file and the name of this group specified for both pf_group1 and pf_group2 – this is equivalent to the original PF implementation [1] where only one group was defined. Having 2 groups allows a greater flexibility and efficiency: imagine a case where the interactions between a ligand and a protein are of interest – pf_group1 could be defined to contain the protein and pf_group2 could be defined to contain the ligand and the pairwise interactions will only be computed for interactions between them; using only one group as in the original PF implementation would mean including both the protein and the ligand in this group and obtaining potentially useless interactions protein-protein and ligand-ligand.

pf_group1 and pf_group2 always contain atom indeces, even when requesting only residue-based calculations. Specifying all atoms of a residue in one or both groups remains the responsibility of the user. If only some of the atoms of a residue are specified and residue-based calculations are requested, the results corresponding to that residue will only contain the interactions involving the specified atoms - the code doesn't automatically fill in the missing atoms to obtain full residues. This is done for flexibility: f.e. if only the residue-based backbone interactions are of interest, the index file should contain only backbone atoms and residue-based calculations should be requested.

`pf_type` – specifies which types of interactions are of interest. It can have one of the values:
`bond angle dihedral polar coulomb lj nb14 bonded nonbonded all`
`nb14` is defined as the combination of LJ14 and Coulomb14 interactions – LJ14 and Coulomb14 cannot be obtained separately.
`bonded` is defined as the combination of `bond`, `angle` and `dihedral`.
`nonbonded` is defined as the combination of `coulomb`, `lj` and `nb14`.
`all` is defined as the combination of `bonded`, `polar` and `nonbonded`.
If no pf_type is specified, "pf_type = all" is assumed.

GROMACS contains several functions which calculate potentials and forces for bonds, angles and dihedral angles. Which ones are used depends on the forcefield, choices in the .mdp file (f.e. “morse=yes”) and user modifications to the topology files (f.e. to add extra harmonic potentials). The PF2 code treats them based on the number of particles involved: all bonded interactions involving 2 particles are considered bonds, all bonded interactions involving 3 particles are considered angles and all bonded interactions involving 4 particles are considered dihedral angles. This makes impossible to differentiate between several types of interactions involving the same number of atoms; f.e. in a protein topology using the OPLS-AA forcefield, the dihedral angles are modeled using a Ryckaert-Bellemans form while the improper dihedral angles are modeled using a classical dihedral angle form – both these types of interactions will be selected by `dihedral`.

When requesting residue-based calculations, a pairwise residue-based interaction of a certain type will exist if at least one atom from one residue and one atom from the second residue are involved in an interaction of that type.

`pf_atombased and pf_residuebased` – specify how the pairwise forces are handled and saved. The 2 keywords are independent: one can be set to “no” to specify that there is no interest in storing, handling and saving that type of data; the default is “no”. The following table shows the possible values and their effect; these values also change the format of the output files (.pfa or .pfr) which is documented in the following chapter.

pf_atombased	pf_residuebased	calculation	output data
no	no	Pairwise forces are not stored, handled or saved.	none
scalar	scalar	Pairwise forces are saved as signed scalars.	signed scalar
vector	vector	Pairwise forces are saved as vectors.	vector
compat_bin	compat_bin	Pairwise forces are saved as signed scalars into a binary format compatible to the original PF implementation. Only works with pf_onepair=summed.	signed scalar
compat_ascii	compat_ascii	Pairwise forces are saved as signed scalars into a text format compatible to the original PF implementation. Only works with pf_onepair=summed.	signed scalar
per_atom_sum	per_residue_sum	For each atom/residue, the sum of all its (unsigned) scalar pairwise forces is saved. Only works with pf_onepair=summed.	scalar
per_atom_average	per_residue_average	For each atom/residue, the average of all its (unsigned) scalar pairwise forces is saved. Only works with pf_onepair=summed.	scalar
per_atom_min	per_residue_min	For each atom/residue, the minimum of all its (unsigned) scalar pairwise forces is saved. Only works with pf_onepair=summed.	scalar
per_atom_max	per_residue_max	For each atom/residue, the maximum of all its (unsigned) scalar pairwise forces is saved.	scalar

		Only works with pf_onepair=summed.	

pf_vector2scalar – controls how the transformation between the vector pairwise forces and their scalar representation is made. It can have the following values:
`projection` – the scalar is the length of the projection of the force vector on the vector determined by the position of the 2 atoms or the position of the COM of the 2 residues.
`norm` – the scalar is the norm of the vector and the sign is determined based on the cosine of the angle between the vector force and the vector determined by the positions of the 2 atoms or the position of the COM of the 2 residues:

- if it's positive, the vector force is oriented within -90 to 90 degrees with respect to the atoms/residues - considered to be same direction = attractive = negative
- if it's negative, the vector force is oriented within 90 to 270 degrees with respect to the atoms/residues - considered to be in opposite direction = repulsive = positive
- if it's zero, the vector force is oriented at exactly 90 or 270 degrees with respect to the atoms/residues - the force can be considered neither attractive nor repulsive, so it's set to zero

The default value is `norm`.

pf_residuesrenumber – controls residue renumbering. The possible values are:
`auto` – keep the residue numbering as in the structure file if there are not collisions and do renumbering if collisions are present
`yes` – do residue renumbering independent of the presence of residue number collisions
`no` – don't do residue renumbering. In case a collision is detected, a warning will be written but renumbering will not take place. Using this option is dangerous as residues cannot be uniquely identified ! If residue number collisions are present, pairwise forces of residues with the same residue number will be summed up together – which is probably not desired.
This option is taken into account only if pf_residuebased is set to something else than `no`; the default value is `auto`.

pf_time_averages_period – is the number of input trajectory frames between averaging scalar data. This can only be used together with pf_atombased and/or pf_residuebased set to `scalar`, `compat_bin` or `compat_ascii`. The default value is 1 – no averaging is done. The output will contain one frame for this number of input frames; if the input frames are not a multiple of this number, a last frame is written after averaging over the remainder of input frames. A value of 0 specifies that averaging should be done over all frames present in the input trajectory; the output will then contain a single frame.

3.3. Input file examples

An example .pfi file for only residue-based output in text compatibility format, taking into consideration all interaction types and with time averaging over all input frames:

```
; pf_onepair could be detailed or summed
pf_onepair = summed

; group1 and group2 as defined in the -pfn file; if not defined,
defaults to 'Protein'
pf_group1 = Protein
```

```
pf_group2 = Ligand

; no interest in atom-based information
pf_atombased = no

; output residue-based information in compatibility format
pf_residuebased = compat_ascii

; interactions type could be one of more of:
; bond angle dihedral polar coulomb lj nb14 bonded nonbonded all
pf_type = all

; scalar summation over all steps
pf_time_averages_period = 0
```

Another example .pfi file, for only atom-based output containing a per-atom sum of all bonded interactions:

```
; pf_onepair could be detailed or summed
pf_onepair = summed

; group1 and group2 as defined in the -pfn file; if not defined,
defaults to 'Protein'
pf_group1 = Protein
pf_group2 = Ligand

; sum of scalar pairwise forces per atom
pf_atombased = per_atom_sum

; no interest in residue-based information
pf_residuebased = no

; interactions type could be one of more of:
; bond angle dihedral polar coulomb lj nb14 bonded nonbonded all
pf_type = bonded
```

Yet another example .pfi file, for atom-based output containing per-atom maximum scalar pairwise interactions and residue-based output containing scalar pairwise forces:

```
; pf_onepair could be detailed or summed
pf_onepair = summed

; group1 and group2 as defined in the -pfn file; if not defined,
defaults to 'Protein'
pf_group1 = Protein
pf_group2 = Ligand

; maximum scalar pairwise force per atom
pf_atombased = per_atom_max

; scalar pairwise forces between residues
```

```
pf_residuebased = scalar

; interactions type could be one of more of:
; bond angle dihedral polar coulomb lj nb14 bonded nonbonded all
pf_type = all
```

And finally an example .pfi file for only residue-based output in text compatibility format, taking into consideration all interaction types and performing residue renumbering, with time averaging over all input frames:

```
; pf_onepair could be detailed or summed
pf_onepair = summed

; group1 and group2 as defined in the -pfn file; if not defined,
defaults to 'Protein'
pf_group1 = Protein
pf_group2 = Ligand

; no interest in atom-based information
pf_atombased = no

; output residue-based information in compatibility format
pf_residuebased = compat_ascii

; ask for residues to be renumbered - starting from 1 and
continuous
pf_residuesrenumber = yes

; interactions type could be one of more of:
; bond angle dihedral polar coulomb lj nb14 bonded nonbonded all
pf_type = all

; scalar summation over all steps
pf_time_averages_period = 0
```

4. Output file formats

The file formats are independent on whether they contain atom-based or residue-based data (.pfa or .pfr file respectively).

The compatibility formats (pf_atombased/pf_residuebased set to `compat_bin` or `compat_ascii`) are supposed to be as close as possible to the ones obtained from the original PF implementation. The following mapping between PF2 and PF interaction types is used:

PF2	PF
bond	iBond
angle	iAngle
dihedral	iDihedral
polar	iPolar
lj	iLJ
coulomb	iCoul
nb14	iDihedral

However, the compatibility formats contain further data: the original PF implementation made the approximation that for an angle or dihedral angle only the extreme atoms interact; PF2 correctly treats interactions between all atoms in these cases, leading to more interactions present in the output file. These formats are supposed to be used only to facilitate reading the data into the current R FDA tools.

The output formats containing pairwise forces (pf_atombased/pf_residuebased set to `scalar` or `vector`) are composed of a set of frames, each frame containing a variable set of interactions. A frame starts with a line containing:

```
frame <frame_number>
```

where <frame_number> is an integer starting from 0, separated by a whitespace from the word “frame”. This line is followed by zero or more lines of form:

```
i j scalar_force interaction_type
```

for pf_atombased/pf_residuebased set to `scalar` or

```
i j force_x force_y force_z interaction_type
```

for pf_atombased/pf_residuebased set to `vector`. i and j are integer numbers which represent the atom indeces (0-based); scalar_force is a floating point number which represents the magnitude of the pairwise force vector; force_x, force_y and force_z are floating point numbers which represent the X, Y and Z components of the pairwise force vector; interaction_type is an integer number which represents the type of interaction – for detailed interactions, this is a constant defined in include/pf_interactions.h file and might change depending on the PF2 version (the existing

constants will normally be kept, new ones will be added with higher values); for summed interactions, this is a sum of the constants corresponding to all interactions present between atoms i and j. All floating point numbers are output using the C printf format specifier “%e”. All values on one line are separated by whitespaces.

The output formats containing per-atom/per-residue data are composed of one line per frame with values on the line separated by whitespaces. All values are floating point numbers which represent the sum/average/minimum/maximum scalar pairwise interaction; they are written using the C printf format specifier “%e”. The first value on the line corresponds to the first atom/residue, the second value on the line corresponds to the second atom/residue, etc.; on each line there are as many per-atom values as the number of atoms and as many per-residue values as the number of residues. If an atom/residue does not have any pairwise interactions (f.e. because it is located further than the cutoff from other atoms, because it doesn't have any interactions of the type specified with pf_type or because it was included neither in group1 nor in group2), the corresponding value is 0.0.

For per-residue data, residue numbering depends on whether residue renumbering has taken place. If the residue renumbering has not taken place, the residue numbers start from 0 and continue until the highest number found in the input structure file. F.e. if the input structure contains 4 residues numbered 5, 6, 8 and 9, the output will contain 10 values: the first 5 values and the 8^{th} value will always be zero. If the residue renumbering has taken place, the residue numbers start from 0 and continue until the total number of residues minus 1.

5. Visualization with VMD

Tcl scripts are used to read and represent in VMD scalar pairwise interactions; there is no representation for vector pairwise interactions.

5.1. Usage of pf_loaduser

Per-atom data is loaded into the “user” field that each atom has assigned in VMD; this field can be used to set the color of the atom. An animation can be produced by updating the value in this field for each frame, showing the variations in per-atom data as they vary in time.

To load and use the pf_loaduser plugin, follow the syntax:

```
package require pf_loaduser
pf_loaduser <filename.pfa> <residues_renumbered> <ignore_zeros>
<color_scale> <color_min> <color_max>
```

The first line will instruct VMD to load the pf_loaduser plugin which will make available the pf_loaduser function which is called on the 2nd line. The pf_loaduser function takes several arguments:

- filename.pfa is a string representing the name of the file containing per-atom data; it assumes that there are as many lines with data in the file as there are currently loaded frames. There is no default, a file name should always be provided.
- residues_renumbered is a string containing “true” or “false”. It should correspond to whether a residue renumbering has been done (automatically or enforced) in the GROMACS PF2 code. The default is “false”.
- ignore_zeros is an integer; if set to 0, zeros will be taken into consideration when calculating the minimum values, if set to something else, zeros will not be taken into consideration when calculating minimum values. The default is 0. This is useful for the case when analyzing only a subset of atoms in which case the atoms not analyzed will have a zero value. If color_min is given, ignore_zeros will not have any effect.
- color_scale is a string representing the name of the VMD color scale. The default is “BGR”.
- color_min and color_max are floating point numbers representing the minimum and maximum values used for color mapping, useful when doing a uniform mapping for several trajectories; if they are not specified or specified as “-1”, the minimum and maximum values found in the file will be used.

At the end of loading a per-atom/per-residue data file, the Tcl script will print the minimum and maximum values used for color mapping.

Example for using the pf_loaduser plugin: load a PDB file and a trajectory then at the console or in the Tk console type:

```
package require pf_loaduser
animate delete beg 0 end 0 skip 0 0
pf_loaduser "peratom-sum.pfa"
```

The second line tells VMD to remove the first frame – this is needed as the PDB will act as the first frame and the .pfa file contains only as many frames as there are in the trajectory. The third line loads the user data from the “peratom-sum.pfa” file and lets VMD automatically map the colors using the minimum and maximum value from this file.

As the data is assigned to the per-atom “user” field, VMD takes care of updating the colors when drawing each frame. For residue-based data, all atoms in a residue receive the same per-atom “user” data.

5.2. Usage of pf_draw

Scalar pairwise forces are represented as a set of cylinders drawn for each pair between the coordinates of the 2 atoms. The scalar values are mapped to either thickness or color of cylinders.

To load and use the pf_draw plugin, follow the syntax:

```
package require pf_draw
pf_init <max_cylinder_width> <what_to_scale>
pf_read <filename.pfa> <residues_renumbered>
pf_draw
```

The first line will instruct VMD to load the pf_draw plugin which will make available functions shown on the 2nd to 4th line.
The second line initializes the plugin: max_cylinder_width is a floating point number specifying the maximum cylinder width; what_to_scale is either “cyl_w” - scale width or “cyl_col” - scale color.
The third line reads scalar pairwise force data (as output with pf_atombased set to `scalar`) from the specified file; residues_renumbered is a string containing “true” or “false” and should correspond to whether a residue renumbering has been done (automatically or enforced) in the GROMACS PF2 code - the default is “false”.
The fourth line draws the cylinders for the currently active structure (a PDB file or the last frame of a trajectory); any cylinders drawn previously are removed before drawing new ones.

When scaling the cylinder width, the maximum cylinder width is taken from the pf_init command line. For each pair of atoms, the value used as cylinder width is the absolute value of the pairwise force. To show the sign of the pairwise forces, the cylinders are colored red for positive pairwise forces and blue for negative ones (zero forces can't be visualized as the cylinder width would be zero).

When scaling the cylinder color, the cylinder width is taken from the pf_init command line. For each pair of atoms, the value used as cylinder color is the value of the pairwise force scaled such that the minimum pairwise force corresponds to the lowest color, the maximum pairwise force corresponds to the maximum color and a zero pairwise force corresponds to the midpoint of the color scale.

At the end of reading a scalar pairwise forces file, the Tcl script will print the difference between the maximum and minimum force in the set; this difference is used for color scaling.

Example for using the pf_loaduser plugin: load a PDB file and a trajectory then at the console or in the Tk console type:

```
package require pf_draw
animate delete beg 0 end 0 skip 0 0
pf_init 0.2 "cyl_w"
pf_read "summed_scalar.pfa"
trace variable vmd_frame(0) w pf_draw
```

The second line tells VMD to remove the first frame – this is needed as the PDB will act as the first frame and the .pfa file contains only as many frames as there are in the trajectory. The third line will initialize the drawing to scale by cylinder width, with a maximum cylinder width of 0.2 – this value needs experimentation. The fifth line looks a bit strange – it tells VMD to call the function pf_draw for each drawing of a frame; this is needed because VMD doesn't know what to do with the pairwise forces values. If this line would be replaced by a simple pf_draw, cylinders would be drawn for the currently loaded PDB file or last frame of the trajectory but they would not be updated when VMD displays a different frame.

If there are many pairwise forces to be represented, drawing the cylinders from a Tcl function can become slow and the display can become cluttered with so many geometrical shapes. It's recommended to filter the pairwise forces, so that only a few are represented at a time.

6. References

1. Stacklies *et al.*: **Implementation of force distribution analysis for molecular dynamics simulations.** *BMC Bioinformatics* 2011 **12**:101

7. Appendix A

Decomposition of pairwise forces for 3- and 4-body potentials.

Notation

$\vec{F}_i$ – force on atom *i* as vector

F_i – magnitude of $\vec{F}_i$

$\vec{U}_i$ – unit vector in the direction of $\vec{F}_i$

such that $\vec{F}_i = F_i \cdot \vec{U}_i$

Forces decomposition for 3-body potentials

For an angle formed by atoms *i*, *j*, *k* (bonds *i*-*j* and *j*-*k*), the sum of the atomic forces is zero:

$$\vec{F}_i + \vec{F}_j + \vec{F}_k = 0 \qquad \text{(S1)}$$

The atomic forces $\vec{F}_i$ and $\vec{F}_k$ can be decomposed into a component in the direction of $\vec{F}_j$ and a perpendicular component. The perpendicular components cancel each other out because of Eq. S1. The components in the direction of $\vec{F}_j$ can be written as:

$$\vec{F}_{ij} = -\vec{F}_{ji} = F_i \cdot \cos(\vec{F}_i, -\vec{F}_j) \cdot \vec{U}_j \qquad \text{(S2)}$$

$$\vec{F}_{kj} = -\vec{F}_{jk} = F_k \cdot \cos(\vec{F}_k, -\vec{F}_j) \cdot \vec{U}_j \qquad \text{(S3)}$$

The pairwise force between atoms *i* and *k* can be obtained by a vector difference:

$$\vec{F}_{ki} = \vec{F}_i - \vec{F}_{ji} = -\vec{F}_{ik} = -(\vec{F}_k - \vec{F}_{jk}) \qquad \text{(S4)}$$

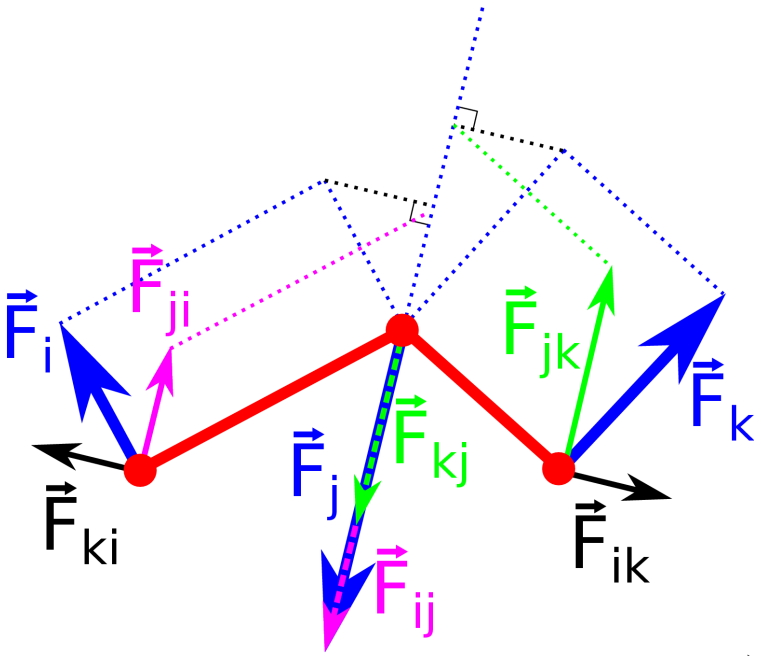


Figure A1: Force decomposition for an angle potential. $\vec{F}_i$ *and* $\vec{F}_k$ *are decomposed into a component in the direction opposite to* $\vec{F}_j$ *and a component in the perpendicular direction.*

Forces decomposition for 4-body potentials

For a dihedral angle formed by atoms *i*, *j*, *k*, *l* (bonds *i*-*j*, *j*-*k* and *k*-*l*), the sum of atomic forces is zero:

$$\vec{F}_i+\vec{F}_j+\vec{F}_k+\vec{F}_l=0 \tag{S5}$$

which can also be written as:

$$\vec{F}_i+\vec{F}_l=-(\vec{F}_j+\vec{F}_k) \tag{S6}$$

Atoms j and k can be considered to form a single body on which a combined force:

$$\vec{F}_{j+k}=\vec{F}_j+\vec{F}_k \tag{S7}$$

acts as shown in Fig. S2a. Similar to the forces decomposition for an angle, $\vec{F}_i$ and $\vec{F}_l$ can be decomposed in a component in the direction of $\vec{F}_{j+k}$ and a component in the perpendicular direction. Due to Eq. S6, the components in the perpendicular direction cancel each other out. The components in the direction of $\vec{F}_{j+k}$ can be written as:

$$\vec{F}_{i(j+k)}=F_i\cdot\cos(-\vec{F}_i,\vec{F}_{j+k})\cdot\vec{U}_{j+k} \tag{S8}$$

$$\vec{F}_{l(j+k)}=F_l\cdot\cos(-\vec{F}_l,\vec{F}_{j+k})\cdot\vec{U}_{j+k} \tag{S9}$$

The pairwise forces acting on atoms j and k can be written in a generic way as:

$$\vec{F}_j=\vec{F}_{ij}+\vec{F}_{kj}+\vec{F}_{lj} \tag{S10}$$

$$\vec{F}_k=\vec{F}_{ik}+\vec{F}_{jk}+\vec{F}_{lk} \tag{S11}$$

and, because *j*+*k* is considered a single body, *j* and *k* do not move with respect to each other, so the pairwise force between them is zero:

$$\vec{F}_{kj}=-\vec{F}_{jk}=0 \tag{S12}$$

such that:

$$\vec{F}_{j+k}=\vec{F}_{ij}+\vec{F}_{lj}+\vec{F}_{ik}+\vec{F}_{lk} \tag{S13}$$

A decomposition of $\vec{F}_{i(j+k)}$ and $\vec{F}_{l(j+k)}$ can be made in the direction of $\vec{F}_j$ and $\vec{F}_k$, shown in Fig. S2b, such that:

$$\vec{F}_{i(j+k)}=\vec{F}_{ij}+\vec{F}_{ik} \tag{S14}$$

$$\vec{F}_{l(j+k)}=\vec{F}_{lj}+\vec{F}_{lk} \tag{S15}$$

Defining α as the angle between $\vec{F}_{j+k}$ and $\vec{F}_j$ and β as the angle between $\vec{F}_{j+k}$ and $\vec{F}_k$, these vectors can be written as:

$$\vec{F}_{ij}=-\vec{F}_{ji}=\frac{F_{i(j+k)}\cdot\sin(\beta)}{\sin(\alpha)\cos(\beta)+\sin(\beta)\cos(\alpha)}\cdot\vec{U}_j \tag{S16}$$

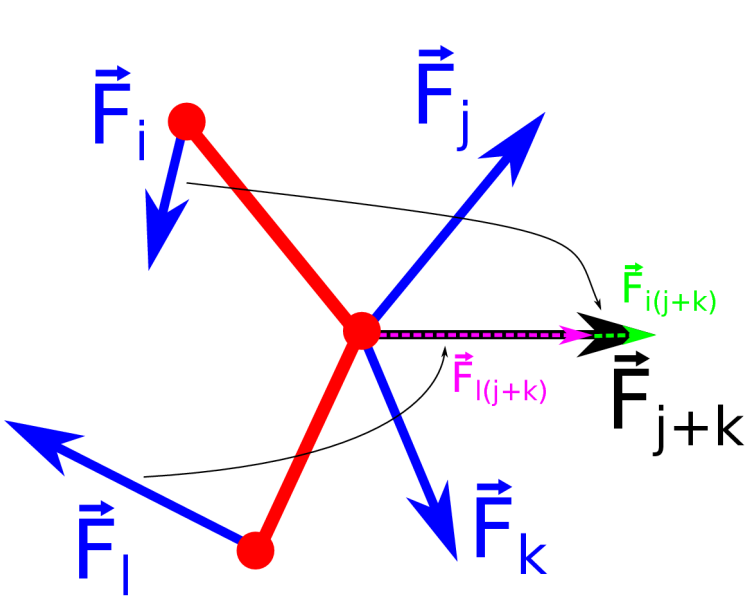
$$\vec{F}_{lj}=-\vec{F}_{jl}=\frac{F_{l(j+k)}\cdot\sin(\beta)}{\sin(\alpha)\cos(\beta)+\sin(\beta)\cos(\alpha)}\cdot\vec{U}_j \tag{S17}$$

$$\vec{F}_{ik}=-\vec{F}_{ki}=\frac{F_{i(j+k)}\cdot\sin(\alpha)}{\sin(\alpha)\cos(\beta)+\sin(\beta)\cos(\alpha)}\cdot\vec{U}_k \tag{S18}$$

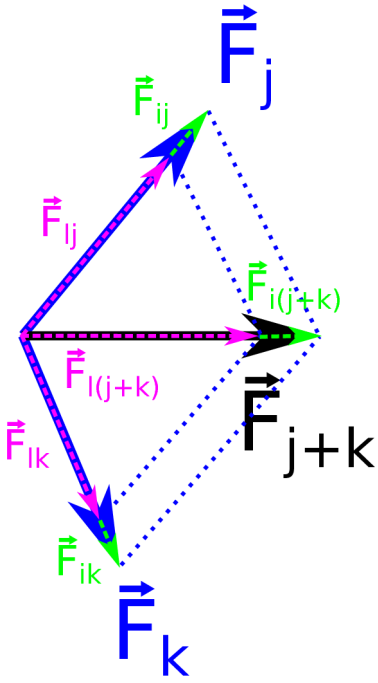
$$\vec{F}_{lk}=-\vec{F}_{kl}=\frac{F_{l(j+k)}\cdot\sin(\alpha)}{\sin(\alpha)\cos(\beta)+\sin(\beta)\cos(\alpha)}\cdot\vec{U}_k \tag{S19}$$

Finally, the pairwise force between atoms *i* and *l* can be obtained by a vector difference:

$$\vec{F}_{li}=\vec{F}_i-\vec{F}_{ji}-\vec{F}_{ki}=-\vec{F}_{il}=-(\vec{F}_l-\vec{F}_{jl}-\vec{F}_{kl}) \tag{S20}$$


$\vec{F}_i$
$\vec{F}_j$
$\vec{F}_{i(j+k)}$
$\vec{F}_{l(j+k)}$
$\vec{F}_{j+k}$
$\vec{F}_l$
$\vec{F}_k$

a


$\vec{F}_j$
$\vec{F}_{ij}$
$\vec{F}_{lj}$
$\vec{F}_{i(j+k)}$
$\vec{F}_{l(j+k)}$
$\vec{F}_{j+k}$
$\vec{F}_{lk}$
$\vec{F}_{ik}$
$\vec{F}_k$

b