CPPTRAJ

Daniel R. Roe

July 11, 2019

https://github.com/Amber-MD/cpptraj

Contents

1 Introduction

Cpptraj[1] (the successor to *ptraj*) is the main program in Amber for processing coordinate trajectories and data files. *Cpptraj* has a wide range of functionality, and makes use of OpenMP/MPI to speed up many calculations, including processing ensembles of trajectories and/or conducting multiple analyses in parallel with MPI.[2]

Here are several notable features of *cpptraj*:

1. Trajectories with different topologies can be processed in the same run.
2. Several actions/analyses in *cpptraj* are OpenMP parallelized; see section 2.7.2 for more details.
3. Trajectory and ensemble reads can be MPI parallelized.
4. Almost any file read or written by *cpptraj* can be compressed (with the exception of the NetCDF trajectory format). So for example gzipped/bzipped topology files can be read, and data files can be written out as gzip/bzip2 files. Compression is detected automatically when reading, and is determined by the filename extension (.gz and .bz2 respectively) on writing.
5. The format of output data files can be specified by extension. For example, data files can be written in xmgrace format if the filename given has a '.agr' extension. A trajectory can be written in DCD format if the '.dcd' extension is used.
6. Multiple output trajectories can be specified, and can be written during action processing (as opposed to only after) via the ***outtraj*** command. In addition, output files can be directed to write only specific frames from the input trajectories.
7. Multiple reference structures can be specified. Specific frames from trajectories may be used as a reference structure.
8. The ***rmsd*** action allows specification of a separate mask for the reference structure. In addition, per-residue RMSD can be calculated easily.
9. Actions that modify coordinates and topology such as the ***strip***/***closest*** actions can often write an accompanying fully-functional stripped topology file.

10. Users usually are able to fine-tune the output format of data files declared in actions using the "**out**" keyword (for example, the precision of the numbers can be changed). In addition, users can control which data sets are written to which files (e.g. if two actions specify the same data file with the 'out' keyword, data from both actions will be written to that data file).

11. Users can manipulate data sets using mathematical expressions (with some limitations), see 5.2 on page 22 for details.

12. There is some support for creating internal loops over e.g. mask expressions and setting internal variables (see ***for***, ***set***, and ***show*** commands).

See the README.md file in the *cpptraj* home directory for information on how to build, authors, and so on.

1.1 Manual Syntax Format

The syntax presented in this manual uses the following conventions:

<> Denotes a variable.

[] Denotes something is optional.

{|} Denotes several choices separated by the '|' character; one of the choices must be specified.

... Denotes the preceding option can be repeated.

Everything else is as printed.

2 Running Cpptraj

Cpptraj can be run in either "interactive mode" or in "batch mode".

2.1 Command Line Syntax

```
cpptraj [-p <Top0>] [-i <Input0>] [-y <trajin>] [-x <trajout>]
        [-ya <args>] [-xa <args>] [<file>]
        [-c <reference>] [-d <datain>] [-w <dataout>] [-o <output>]
        [-h | --help] [-V | --version] [--defines] [-debug <#>]
        [--interactive] [--log <logfile>] [-tl]
        [-ms <mask>] [-mr <mask>] [--mask <mask>] [--resmask <mask>]
* denotes a flag may be specified multiple times.
```

-p <**Top0**>* Load <Top0> as a topology file.

-i <**Input0**>* Read input from <Input0>.

-y <**trajin**>* Read from trajectory file <trajin>; same as input 'trajin <trajin>'.

-x <**trajout**>* Write trajectory file <trajout>; same as input 'trajout <trajout>'.

-ya <**args**>* Input trajectory file arguments.

-xa <**args**>* Output trajectory file arguments.

<**file**>* A topology, input trajectory, or file containing cpptraj input.

-c <**reference**>* Read <reference> as reference coordinates; same as input 'reference <reference>'.

-d <**datain**>* Read data in from file <datain> ('readdata <datain>').

-w <**dataout**> Write data from <datain> as file <dataout> ('writedata <dataout>).

-o <**output**> Write CPPTRAJ STDOUT output to file <output>.

-h | –**help** Print command line help and exit.

-V | –**version** Print version and exit.

–**defines** Print compiler defines and exit.

-debug <#> Set global debug level to <#>; same as input 'debug <#>'.

–**interactive** Force interactive mode.

–**log** <**logfile**> Record commands to <logfile> (interactive mode only). Default is 'cpptraj.log'.

-tl Print length of trajectories specified with '-y' to STDOUT. The total number of frames is written out as 'Frames: <X>'

-ms <**mask**> Print selected atom numbers to STDOUT. Selected atoms are written out as 'Selected= 1 2 3 ...'

-mr <mask> : Print selected residue numbers to STDOUT. Selected residues are written out as 'Selected= 1 2 3 ...'

–**mask** <**mask**> Print detailed atom selection to STDOUT.

–**resmask** <mask> : Print detailed residue selection to STDOUT.

Note that unlike *ptraj*, in *cpptraj* it is not required that a topology file be specified on the command line as long as one is specified in the input file with the 'parm' keyword. Multiple topology/input files can be specified by use of multiple '-p' and '-i' flags. All topology and coordinate flags will be processed before any input flags.

2.2 Commands

Input to *cpptraj* is in the form of commands, which can be categorized in to 2 types: immediate and queued. Immediate commands are executed as soon as they are encountered. Queued commands are initialized when they are encountered, but are not executed until a Run is executed via a ***run*** or ***go*** command. Actions, Analyses, and Trajectory commands (except ***reference***) are queued commands; however, they can also be run immediately via commands such as ***crdaction***, ***runanalysis***, ***loadcrd***, etc. See 7 on page 30 for more details.

Commands fall into seven categories:

General (Immediate) These commands are executed immediately when entered.

System (Immediate) These are unix system commands (e.g. 'ls', 'pwd', etc).

Coords (Immediate) These commands are used to manipulate COORDS data sets; see 7 on page 30 for more details.

Trajectory (Queued) These commands prepare cpptraj for reading or writing trajectories during a Run.

Topology (Immediate) These commands are used to read, write, and modify topology information.

Action (Queued) These commands specify actions that will be performed on coordinate frames read in from trajectories during a Run.

Analysis (Queued) These commands specify analyses that will be performed on data that has been either generated from a Run or read in from an external source.

Control (Immediate) These commands set up control blocks that can be used to e.g. loop over a set of commands.

In addition to normal commands, *cpptraj* now has the ability to perform certain basic math operations, even on data sets. See 5.2 on page 22 for more details.

Commands in *cpptraj* can be read in from an input file or from the interactive command prompt. A '#' anywhere on a line denotes a comment; anything after '#' will be ignored no matter where it occurs. A '\' allows the continuation of one line to another. For example, the input:

```
# Sample input
trajin mdcrd # This is a trajectory
rms first out rmsd.dat \
    :1-10
```

Translates to:

```
trajin mdcrd
rms first out rmsd.dat :1-10
```

2.3 Getting Help

If in interactive mode, 'help <command>' can be used to get the associated keywords as well as an abbreviated description of the command. Most commands have a corresponding test which also serves as an example of how to use the command. See $AMBERHOME/AmberTools/test/cpptraj/README for more details.

2.4 Batch mode

In "batch" mode, cpptraj is executed from the command line with one or more input files containing commands to be processed or STDIN. The syntax of <input file> is similar to that of *ptraj*. Keywords specifying different commands are given one per line. Lines beginning with '#' are ignored as comments. Lines can also be continued through use of the '\' character. This is the only allowed mode for cpptraj.MPI.

2.5 Interactive mode

In "interactive mode" users can enter commands in a UNIX-like shell. Interactive mode is useful for running short and simple analyses or for trying out new kinds of analyses. If *cpptraj* is run with '–interactive', no arguments, or no specified input file:

```
cpptraj
cpptraj --interactive
cpptraj <parm file>
cpptraj -p <parm file>
```

this brings up the interactive interface. This interface supports command history (via the up and down arrows) and tab completion for commands and file names. If no log file name has been given (with '–log <logfile>'), all commands used in interactive mode will be logged to a file named 'cpptraj.log', which can subsequently be used as input if desired. When starting cpptraj, command histories will be read from any existing logs.

2.6 Trajectory Processing "Run"

Like *ptraj*, a trajectory processing "Run" is one of the main ways to run *cpptraj*. First the Run is set up via commands read in from an input file or the interactive prompt. Trajectories are then read in one frame at a time (or in the case of ensemble processing all frames from a given step are read). Actions are performed on the coordinates stored in the frame, after which any output coordinates are written. At the end of the run, any data sets generated are written, and any queued Analyses are performed.

2.6.1 Actions and multiple topologies

Since *cpptraj* supports multiple topology files, during a Run actions are set up every time the topology changes in order to recalculate things like what atoms are in a mask etc. Actions that are not valid for the current topology are skipped for that topology. So for example given two topology files with 100 residues, if the first topology file processed includes a ligand named MOL and the second one does not, the action:

```
distance :80 :MOL out D_80-to-MOL.dat
```

will be valid for the first topology but not for the second, so it will be skipped as long as the second topology is active.

2.7 Parallelization

Cpptraj has many levels of parallelization that can be enabled via the '-mpi', '-openmp', and/or '-cuda' configure flags for MPI, OpenMP, and CUDA parallelization respectively. At the highest level, trajectory and ensemble reads are parallelized with MPI. In addition, certain time consuming actions have been parallelized with OpenMP and/or CUDA.

Note that any combination of the '-openmp', '-cuda', and '-mpi' flags may be used to generate a hybrid MPI/OpenMP/CUDA binary; however this may require additional runtime setup (e.g. setting OMP_NUM_THREADS for OpenMP) to work properly and not oversubscribe cores.

2.7.1 MPI Trajectory Parallelization

Cpptraj has two levels of MPI parallelization for reading input trajectories. The first is for '***trajin***' trajectory input, where the trajectory read is divided as evenly as possible among all input frames (across-trajectory parallelism). For example, if given two trajectories of 1000 frames each and 4 MPI processes, process 0 reads frames 1-500 of trajectory 1, process 1 reads frames 501-1000 of trajectory 1, process 2 reads frames 1-500 of trajectory 2, and process 3 reads frames 501-1000 of trajectory 2. Most Actions will work with across-trajectory parallelization with the exception of the following:

'clusterdihedral', 'contacts', 'createreservoir', 'gist', 'lipidorder', 'pairwise', 'stfcdiffusion', 'unwrap', and 'xtalsymm'. Note that 'diffusion' will only work with across-trajectory parallelism if no imaging is to be performed.

The second is for ***'ensemble'*** trajectory input, where the reading/processing/writing of each member of the ensemble is divided up among MPI processes. The number of MPI processes must be a multiple of the ensemble size. If the number of processes is greater than the ensemble size then the processing of each ensemble member will be divided among MPI processes (i.e. across-trajectory parallelism will be used). For example, given an ensemble of 4 trajectories and 8 processes, processes 0 and 1 are assigned to the first ensemble trajectory, processes 2 and 3 are assigned to the second ensemble trajectory, and so on. When using ensemble

mode in parallel it is recommended that the ***ensemblesize*** command be used prior to any ensemble command as this will make set up far more efficient.

In order to use the MPI version, Amber/*cpptraj* should be configured with the '-mpi' flag. You can tell if *cpptraj* has been compiled with MPI as it will print 'MPI' in the title, and/or by calling 'cpptraj —defines' and looking for '-DMPI'.

2.7.2 OpenMP Parallelization

Some of the more time-consuming actions/analyses in *cpptraj* have been parallelized with OpenMP to take advantage of machines with multiple cores. In order to use OpenMP parallelization Amber/*cpptraj* should be configured with the '-openmp' flag. You can easily tell if *cpptraj* has been compiled with OpenMP as it will print 'OpenMP' in the title, and/or by calling 'cpptraj —defines' and looking for '-D_OPENMP'. The following actions/analyses have been OpenMP parallelized:

```
2drms/rms2d
atomiccorr
checkstructure
closest
cluster (pair-wise distance calculation and sieved frame restore only)
dssp/secstruct
energy
gist (non-bonded calculation)
hbond
kde
lipidscd
mask (distance-based masks only)
matrix (coordinate covariance matrices only)
minimage
radial
replicatecell
rmsavgcorr
spam
surf
velocityautocorr
volmap
watershell
wavelet
```

By default OpenMP *cpptraj* will use all available cores. The number of OpenMP threads can be controlled by setting the OMP_NUM_THREADS environment variable.

2.7.3 CUDA Parallelization

Some time-consuming actions in cpptraj have been parallelized with CUDA to take advantage of machines with NVIDIA GPUs. In order to use CUDA parallelization Amber/cpptraj should be configured with the '-cuda' flag. You can easily tell if cpptraj has been compiled with CUDA as it will print 'CUDA' and details on the current graphics device in the title, and/or by calling 'cpptraj —defines' and looking for '-DCUDA'. The following actions have been CUDA parallelized:

```
closest
watershell
```

3 General Concepts

3.1 Units

Cpptraj uses the AKMA system of units. The execption is time, which is typically expressed in ps (except where noted).

Variable	Unit
Length	Angstrom
Energy	kcal/mol
Mass	AMU
Charge	electron
Time	ps (typically)
Force	kcal/mol*Angstrom

3.2 Atom Mask Selection Syntax

The mask syntax is similar to *ptraj*. Note that the characters ':', '@', and '*' are reserved for masks and should not be used in output file or data set names. All masks are case-sensitive. Either names or numbers can be used. Masks can contain ranges (denoted with '-') and comma separated lists. The logical operands '&' (and), '|' (or), and '!' (not) are also supported.

The syntax for elementary selections is the following:

@{atom numlist} e.g. '@12,17', '@54-85', '@12,54-85,90'

@{atom namelist} e.g. '@CA', '@CA,C,O,N,H'

@%{atom type name} e.g. '@%CT'

@/{atom_element_name} e.g. '@/N'

:{residue numlist} e.g. ':1-10', ':1,3,5', ':1-3,5,7-9'

:{residue namelist} e.g. ':LYS', ':ARG,ALA,GLY'

:/{chain id} e.g. ':/B', ':/A,D'. Requires chain ID information be present in the topology.

:;{pdb residue number} e.g. ':;2-4,8'. Requires a PDB loaded as topology, or Amber topology with embedded PDB information (see **??** on page ??).

^{molecule numlist} e.g. '^1-10', ':23,84,111'

<mask><distance op><distance> Selection by distance, see below.

Several wildcard characters are supported:

'*' Zero or more characters.

'=' Same as '*'

'?' One character.

The wildcards can also be used with numbers or other mask characters, e.g. ':?0' means ":10,20,30,40,50,60,70,80,90", ':*' means all residues and '@*' means all atoms.

Compound expressions of the following type are allowed:

```
:{residue numlist | namelist}@{atom namelist | numlist}
```

and are processed as:

```
:{residue numlist | namelist} & @{atom namelist | numlist}
```

e.g. ':1-10@CA' is equivalent to ":1-10 & @CA".

More examples:

:ALA,TRP All alanine and tryptophan residues.

:5,10@CA CA carbon in residues 5 and 10.

:*&!@H= All non-hydrogen atoms (equivalent to "!@H=").

@CA,C,O,N,H All backbone atoms.

!@CA,C,O,N,H All non-backbone atoms (=sidechains for proteins only).

:1-500@O&!(:WAT|:LYS,ARG) All backbone oxygens in residues 1-500 but not in water, lysine or arginine residues.

^1-2:ASP All residues named 'ASP' in the first two molecules.

:/A,D@CA All atoms named 'CA' in chains A and D.

Distance-based Masks

There are two very important things to keep in mind when using distance based masks:

1. Distance-based masks that update each frame are currently only supported by the ***mask*** action.
2. Selection by distance for everything but the ***mask*** action requires defining a reference frame with ***reference***; distances are then calculated using the specified reference frame only. This reference frame can be changed using the ***activeref*** command.

The syntax for selection by distance is a <**mask**> expression followed by a <**distance operator**> followed by a <**distance**> (which is in Angstroms). The <**distance operator**> consists of 2 characters: '<' (within) or '>' (without) followed by either '^' (molecules), '**:**' (residues), or '**@**' (atoms). For example, '<**:**3.0' means "residues within 3.0 Angstroms" etc. For residue- and molecule-based distance selection, if any atom in that residue/molecule matches the given distance criterion, the entire residue/molecule is selected.

In plain language, the entire distance mask can be read as "Select <**distance operator**> <**distance**> of <**mask**>". So for example, the mask expression:

```
:11-17<@2.4
```

Means "Select atoms within 2.4 Å distance of atoms selected by ':11-17' (residues numbered 11 through 17)".

To strip everything outside 3.0 Å (i.e. without 3.0 Å) from residue 4 using specified reference coordinates:

```
reference mol.rst7
trajin mol.rst7
strip !(:4<:3.0)
```

3.3 Ranges

For several commands some arguments are ranges (e.g. 'trajout onlyframes <range>', 'nastruct resrange <range>', 'rmsd perres range <range>'); **THESE ARE NOT ATOM MASKS**. They are simple number ranges using '-' to specify a range and ',' to separate different ranges. For example 1-2,4-6,9 specifies 1 to 2, 4 to 6, and 9, i.e. '1 2 4 5 6 9'.

3.4 Parameter/Reference Tagging

Parameter and reference files may be 'tagged' (i.e. given a nickname); these tags can then be used in place of the file name itself. A tag in *cpptraj* is recognized by being bounded by brackets ('[' and ']'). This can be particularly useful when reading in many parameter or reference files. For example, when reading in multiple reference structures:

```
trajin Test1.crd
reference 1LE1.NoWater.Xray.rst7 [xray]
reference Test1.crd lastframe [last]
reference Test2.crd 225 [open]
rms Xray ref [xray] :2-12@CA out rmsd.dat
rms Last ref [last] :2-12@CA out rmsd.dat
rms Open ref [open] :2-12@CA out rmsd.dat
```

This defines three reference structures and gives them tags [xray], [last], and [open]. These reference structures can then be referred to by their tags instead of their filenames by any action that uses reference structures (in this case the RMSD action).

Similarly, this can be useful when reading in multiple parameter files:

```
parm tz2.ff99sb.tip3p.truncoct.parm7 [tz2-water]
parm tz2.ff99sb.mbondi2.parm7 [tz2-nowater]
trajin tz2.run1.explicit.nc parm [tz2-water]
reference tz2.dry.rst7 parm [tz2-nowater] [tz2]
rms ref [tz2] !(:WAT) out rmsd.dat
```

This defines two parm files and gives them tags [tz2-water] and [tz2-nowater], then reads in a trajectory associated with one, and a reference structure associated with the other. Note that in the 'reference' command there are two tags; the first goes along with the 'parm' keyword and specifies what parameter file the reference should use, the second is the tag given to the reference itself (as in the previous example) and is referred to in the subsequent RMSD action.

4 Variables and Control Structures

As of version 18, CPPTRAJ has limited support for "script" variables and 'for' loops. Script variables are referred to by a dollar sign ('$') prefix and are replaced when they are processed. Note that to use script variables in CPPTRAJ input that is also inside e.g. a BASH script, they can be escaped with the '\' character, e.g.

```
#!/bin/bash
TOP=MyTop.parm7
cpptraj <<EOF
set topname=$TOP
parm \$topname
EOF
```

Note that regular CPPTRAJ 1D Data Sets that contain a single value can be used as script variables (if the Data Set contains more than 1 value only the first value will be used).

Command	Description
for	Create a 'for' loop.
set	Set or update a script variable.
show	Show all current script variables and their values.

4.1 for

```
for { {atoms|residues|molecules|molfirstres|mollastres}
    <var> inmask <mask> [parm <name> | parmindex <#> | <#>] ... |
      <var> in <list> |
  <var>=<start>;[<var><end OP><end>;]<var><increment OP>[<value>] ... }
END KEYWORD: 'done'
  Available 'end OP'       : '<' '>'
  Available 'increment OP' : '++', '--', '+=', '-='
```

atoms|**residues**|**molecules**|**molfirstres**|**mollastres** <**var**> **inmask** <**mask**>
Loop over atoms/residues/molecules/first residue in molecules/last residue in molecules selected by the given mask expression, set as script variable <var>.

parm <**name**> | **parmindex** <#> <#> Select topology that <mask> should be based on (default first topology).

<**var**> **in** <**list**> Loop over a comma-separated list of strings. File name wildcards can be used.

<**var**>=<**start**>;[<**var**><**end OP**><**end**>;]<**var**><**increment OP**>[<**value**>]
Loop over integer script variable <var> starting from <start>, optionally ending at <end>, increment by <value>.

Create a for loop using one or more mask expressions or integers. Loops can be nested, but currently inner loops cannot refer to output loop variables. Integer loops may be used without an end condition, but in that case at least one descriptor in the loop should have an end condition or refer to a mask. Loops are ended by the **done** keyword. Note that unlike C-style for loops the variable is not incremented on the final execution, so the final value of <**var**> will be <**end**> (or the last selection in the mask).

For example:

```
for atoms A0 inmask :1-3@CA i=1;i++
  distance d$i :TCS $A0 out $i.dat
done
```

This loops over all atoms in the mask expression ':1-3@CA' (all atoms named CA in residues 1 to 3) and creates a variable named 'i' that starts from 1 and is incremented by 1 each iteration. Inside the loop, the mask selection is referred to by **$A0** and the integer by **$i**. This is equivalent to doing 3 distance commands like so:

```
distance d1 :TCS :1@CA out 1.dat
distance d2 :TCS :2@CA out 2.dat
distance d3 :TCS :3@CA out 3.dat
```

To loop over files named trajA*.nc and trajB*.nc:

```
for TRAJ in trajA*.nc,trajB*.nc
  trajin $TRAJ 1 last 10
done
```

4.2 set

```
set { <variable> <OP> <value> |
  <variable> <OP> {atoms|residues|molecules} inmask <mask>
    [parm <name> | parmindex <#> | <#>]
  <variable> <OP> trajinframes }
  Available <OP> : '=', '+='
```

<variable> <OP> <value> Set or append a script variable.

<variable> <OP> {atoms|residues|molecules} inmask <mask> Set/append a script variable to/by the total number of atoms/residues/molecules selected by given mask expression.

parm <name> | parmindex <#> | <#> Topology to which mask should correspond (default first).

<variable> <OP> trajinframes Set/append a script variable to/by the total number of frames in trajectories currently loaded by *trajin* commands.

Set (<OP> = '=') or append (<OP> = '+=') a script variable. Script variables are referred to in CPPTRAJ input by using a dollar sign '$' prefix.

For example, the following input will print info for the last 10 atoms in a topology to 'last10.dat':

```
set Natom = atoms inmask *
last10 = $Natom - 10
show
atoms "@$last10 - $Natom" out last10.dat
```

4.3 show

```
show
```

Show all current script variables and their values.

5 Data Sets and Data Files

In *cpptraj*, Actions and Analyses can generate one or more data sets which are available for further processing. For example, the ***distance*** command creates a data set containing distances vs time. The data set can be named by the user simply by specifying a non-keyword string as an additional argument. If no name is given, a default one will be generated based on the action name and data set number. For example:

```
distance d1-2 :1 :2 out d1-2.dat
```

will create a data set named "d1-2". If a name is not specified, e.g.:

```
distance :1 :2 out d1-2.dat
```

the data set will be named "Dis_00000".

Data files are created automatically by most commands, usually via the "**out**" keyword. Data files can also be explicitly created with the ***write***/***writedata*** and ***create*** commands. Data can also be read in from files via the ***readdata*** command. Cpptraj currently recognizes the formats listed in 1, although it cannot write in all formats. In addition, a data set must be valid for the data file format. For example, 3D data (such as a grid) can be written to an OpenDX format file but not a Grace format file.

The default file format is called 'Standard', which simply has data in columns, like *ptraj*, although multiple data sets can be directed to the same output file. The format of a file can be changed either by specifying a recognized keyword (either on the command line itself or later via a 'datafile' command) or by giving the file an extension corresponding to te format, so 'filename.agr' will output in Grace format, and 'filename.gnu' will output in Gnuplot contour, and so on. The xmgrace/gnuplot output is particularly nice for the secstruct sumout and rmsd perresout files. Additional options for data files can be found in 6 on page 23.

Any action using the "out" keyword will allow data sets from separate commands to be written into the same file. For example, the commands:

```
dihedral phi :1@C :2@N :2@CA :2@C out phipsi.dat
dihedral psi :2@N :2@CA :2@C :3@N out phipsi.dat
```

will assign the "phi" and "psi" data sets generated from each action to the standard data output file "phipsi.dat":

```
#Frame   phi   psi
```

5.1 Data Set Selection Syntax

Many analysis commands can be used to analyze multiple data sets. The general format for selecting data sets is:

Format	Filename Extensions	Keyword	Valid Dimensions	
Standard	.dat	dat	1D, 2D, 3D	
Grace	.agr, .xmgr	grace	1D	
Gnuplot	.gnu	gnu	1D, 2D	
Xplor	.xplor, .grid	xplor	3D	
OpenDX	.dx	opendx	3D	
Amber REM log	.log	remlog	-	R
Amber MDOUT	.mdout	mdout	-	Energy infor
Amber Evecs	.evecs	evecs	Modes data set only	
Amber Constant pH output	.cpout	cpout	pH data only	
Vector pseudo-traj	.vectraj	vectraj	Vector data set only.	W
Gromacs XVG	.xvg	xvg	-	R
CCP4	.ccp4	ccp4	3D	
Charmm REPD log	.exch	charmmrepd	-	R
Charmm Output	.charmmout	charmmout	-	Energy infor

Table 1: DataFile formats recognized by *cpptraj*. 'Valid Dimensions' shows what dimensions the format is valid for (e.g. you cannot write a 1D data set with OpenDX format).

```
<name>[<aspect>]:<index>
```

The '*' character can be used as a wild-card for *entire* names (no partial matches).

- **<name>:** The data set name, usually specified in the action (e.g. in 'distance d0 @1 @2' the data set name is "d0").
- **<aspect>:** Optional; this is set for certain data sets internally in order to easily select subsets of data. **The brackets are required**. For example, when using 'hbond series', both solute-solute and solute-solvent hydrogen bond time series may be generated. To select all solute-solute hydrogen bonds one would use the aspect "[solutehb]"; to select solute-solvent hydrogen bonds the aspect "[solventhb]" would be used. Aspects are hard-coded and are listed in the commands that use them.
- **<index>:** Optional; for actions that generate many data sets (such as 'rmsd perres') an index is used. Depending on the action, the index may correspond to atom #s, residue #s, etc. A number range (comma and/or dash separated) may be used.

For example: to select all data sets with aspect "[shear]" named NA_00000:

```
NA_00000[shear]
```

To select all data sets with aspect "[stagger]" with any name, indices 1 and 3:

```
*[stagger]:1,3
```

In ensemble mode, data set selection has additional syntax:

```
<name>[<aspect>]:<index>%<member>
```

Where <member> is the ensemble member number starting from 0.

5.2 Data Set Math

As of version 15, *cpptraj* can perform basic math operations, even on data sets (with some limitations). Currently recognized operations are:

Operation	Symbol
Minus	-
Plus	+
Divide	/
Multiply	*
Power	^
Negate	-
Assign	=

Several functions are also supported:

Function	Form
Square Root	sqrt()
Exponential	exp()
Natural Logarithm	ln()
Absolute Value	abs()
Sine	sin()
Cosine	cos()
Tangent	tan()
Summation	sum()
Average	avg()
Standard Deviation	stdev()
Minimum	min()
Maximum	max()

Numbers can be expressed in scientific notation using "E" notation, e.g. 1E-5 = 0.00001. The parser also recognizes PI as the number pi. Expressions can also be enclosed in parentheses. So for example, the following expression is valid:

```
> 1 - ln(sin(PI/4) * 2)^2
Result: 0.879887
```

Results of numerical calculations like the above can be assigned to a variable (essentially a data set of size 1) for use in subsequent calculations, e.g.

```
> R = 1 - ln(sin(PI/4) * 2)^2
Result stored in 'R'
> R + 1 Result: 1.879887
```

Data sets can be specified in expressions as well. Currently data sets in an expression must be of the same type and only 1D, 2D, and 3D data sets are supported. Functions are applied to each member of the data set. So for example, given two 1D data sets of the same size named D0 and D1, the following expression:

```
> D2 = sqrt( D0 ) + D1
```

would take the square root of each member of D0, add it to the corresponding member of D1, and assign the result to D2. The following table lists which operations are valid for data set types. If a type is not listed it is not supported:

Data Set Type	Supported Ops	Supported Funcs	Notes
1D (integer, double, float)	All	All	
1D (vector)	+, -, *, /, =	None	'*' is dot product
2D (matrices)	+, -, /, *, =	sum, avg, stdev, min, max	
3D (grids)	+, -, /, *, =	sum, avg, stdev, min, max	

6 Data File Options

Data file output can be handled multiple ways in cpptraj. Output data files can be created by Actions/Analyses/Commands, or can be explicitly created with ***writedata*** (8.29 on page 48) or ***create*** (8.4 on page 38) commands. Reading data from files is only done via the ***readdata*** command (8.19 on page 45).

In general, data files which have been declared with an **'out'** keyword will recognize data file write keywords on the same command line. For example, the **'time'** argument can be passed directly to the output from a ***distance*** command:

```
distance d0 :1 :2 out d0.agr time 0.001
```

The data file format can be changed from standard implicitly by using specific filename extensions or keywords. If the extension is not recognized or no keyword is give the default format is 'Standard'. Keywords and extensions for data file formats recognized by *cpptraj* are shown in 1. Note that the use of certain options may be restricted for certain data file formats. These options can also be passed to data files via the ***datafile*** command (8.6 on page 39).

```
[<format keyword>]
[{xlabel|ylabel|zlabel} <label>] [{xmin|ymin|zmin} <min>] [sort]
[{xstep|ystep|zstep} <step>] [time <dt>] [prec <width>[.<precision>]]
[xprec <width>[.<precision>]] [xfmt {double|scientific|general}]
[noensextension]
```

{xlabel | ylabel | zlabel} <**label**> Set the x-axis label for the specified datafile to <label>. For regular data files this is the header for the first column of data. If the data is at least 2-dimensional 'datafile ylabel <label>' will likewise set the y-axis label.

{xmin | ymin | zmin} <**min**> Set the starting X coordinate value to <min>. If the data is at least 2-dimensional 'datafile ymin <min>' will likewise set the starting Y coordinate value.

sort Sort data sets prior to write. Ordering is by name, aspect, then index (all descending).

{xstep | ystep | zstep} <**step**> Multiply each frame number by <step> (x coordinates). If the data is at least 2-dimensional 'datafile ystep <step>' will likewise multiply y coordinates by <step>.

time <**dt**> Equivalent to the *ptraj* argument 'time' that could be specified with many actions. Multiplies frame numbers (x-axis) by <dt>.

prec <**width**>**[.**<**precision**>**]** Change the output format width (and optionally precision) of all sets *subsequently* added to the data file (i.e. does not change the precision of any data sets currently in the file). For example,

```
prec 12.4
prec 10
```

xprec <**width**>**[.**<**precision**>**]** Change output ordinate width and precision.

xfmt {double|scientific|general} Change output ordinate format.

[noensextension] Omit ensemble extension in ensemble processing mode. NOTE: THIS OPTION HAS NOT BEEN FULLY TESTED IN PARALLEL.

6.1 Standard Data File Options

Write

```
[invert] [noxcol] [groupby <type>] [noheader] [square2d|nosquare2d]
[nosparse|sparse [cut <cutoff>]]
```

invert Normally, data is written out with X-values pertaining to frames (i.e. data over all

trajectories is printed in columns). This command flips that behavior so that X-values pertain to data sets (i.e. data over all trajectories is printed in rows).

groupby <**type**> (1D) group data sets by <type>:

name Group by name.

aspect Group by aspect.

idx Group by index.

ens Group by ensemble number.

dim Group by dimension.

noxcol Prevent printing of indices (i.e. the #Frame column in most datafiles) for the specified datafile. Useful e.g. if one would like a 2D plot such as phi vs psi. For example, given the input:

```
dihedral phi :1@C :2@N :2@CA :2@C out phipsi.dat
dihedral psi :2@N :2@CA :2@C :3@N out phipsi.dat
datafile phipsi.dat noxcol
```

Cpptraj will write a 2 column datafile containing only phi and psi, no frame numbers will be written.

noheader Prevent printing of header line (e.g. '#Frame D1') at the beginning of data file.

square2d Write 2D data as a square matrix, e.g.:

```
<1,1> <2,1> <3,1>
<1,2> <2,2> <3,2>
```

nosquare2d Write 2D data in 3 columns as:

```
<X> <Y> <Value>
```

sparse Only write 3D grid voxels with value > cutoff (default 0).

cut <**cut**> Cutoff for 'sparse'; default 0.

nosparse Write all 3D voxels (default).

Read

```
[ read1d [index <col>] [onlycols <range>] ]
[read2d] [vector] [mat3x3]
[ read3d [dims <nx>,<ny>,<nz>] [origin <ox>,<oy>,<oz>]
  [delta <dx>,<dy>,dz>] [prec {dbl|flt}] [bin {center|corner} ]
```

index <**col**> Use column <col> (starting from 1) as index column (1D data only).

read1d Read data as 1D data sets (default).

index <**col**> Use column <col> (starting from 1) as index column (1D data only).

onlycols <**range**> Only read columns in range.

read2d Read data as 2D square matrix.

vector Read data as vector. If indices are present they will be skipped. Assume first 3 columns after the index volumn are vector X, Y, and Z, and (if present) the next 3 columns contain vector origin X, Y, and Z.

mat3x3 Read data as 3x3 matrix. If indices are present they will be skipped. Assume matrices are in row major order on each line, i.e. M(1,1) M(1,2) ... M(3,2) M(3,3).

read3d Read data as 3D grid. If no dimension data in file must also speify 'dims'.

dims <**dx**>,<**dy**>,<**dz**> Grid dimensions.

origin <**ox**>,<**oy**>,<**oz**> Grid origins (default 0,0,0).

delta <**dx**>,<**dy**>,**dz**> Grid spacings (default 1,1,1).

prec {dbl|flt} Grid precision, double or float (default float).

bin {center|corner} Coords specify bin centers or corners (default corners).

By default, standard data files are assumed to contain 1D data in columns. Data set legends will be read in if the file has a header line (denoted by '#'). Columns labeled '#Frame' are automatically considered the 'index' column and skipped. Data sets are stored as <name>:<idx> where <name> is the given data set name (the file name if not specified) and <idx> corresponds to the column the data was read from starting from 1. *Cpptraj* assumes the data increases monotonically and will automatically attempt to determine the dimensions of the data set(s); a warning will be printed if this is not successful.

6.2 Grace Data File Options

For more information on Grace see http://plasma-gate.weizmann.ac.il/Grace/.

Write

[invert] [xydy]

invert Normally, data is written out with X-values pertaining to frames (i.e. data over all trajectories is printed in columns). This command flips that behavior so that X-values pertain to data sets.

xydy Combine consecutive pairs of sets into XYDY sets.

If a single string data set is specified when writing Grace format, it is assumed they are data point labels.

Read

Cpptraj will read set legends from grace files, and data sets are stored as <name>:<idx> where <name> is the given data set name (the file name if not specified) and <idx> corresponds to the set number the data was read from starting from 0.

6.3 Gnuplot Data File Options

For more information on these options it helps to look at the PM3D options in the Gnuplot manual (see http://www.gnuplot.info/).

Write

```
[nolabels] [usemap] [pm3d] [nopm3d] [title <title>]
[jpeg] [noheader] [{xlabels|ylabels|zlabels} <labellist>]
```

nolabels Do not print axis labels.

usemap pm3d output with 1 extra empty row/col (may improve look).

pm3d Normal pm3d map output.

nopm3d Turn off pm3d

jpeg Plot will write to a JPEG file when used with gnuplot.

title <title> Set plot title (default is file name).

binary Plot will be written in binary format.

noheader Do not format plot; data output only.

palette <arg> Change gnuplot pm3d palette to <arg>:

'rgb' Red, yellow, green, cyan, blue, magenta, red.

'kbvyw' Black, blue, violet, yellow, white.

'bgyr' Blue, green, yellow, red.

'gray' Grayscale.

xlabels|ylabels|zlabels <labellist> Set x, y, or z axis labels with comma-separated list, e.g. 'xlabels X1,X2,X3'.

6.4 Amber REM Log Options

Note that multiple REM logs can be specified in a single ***readdata*** command. See 12.27 on page 200 for more on replica log analysis.

Read

`[nosearch] [dimfile <file>] [crdidx <crd indices>]`

[nosearch] If specified do not automatically search for MREMD dimension logs.

[dimfile <file>] remd.dim file for processing MREMD logs.

[crdidx <crd indices>] Use comma-separated list of indices as the initial coordinate indices (H-REMD only). For example (4 replicas):

```
crdidx 4,2,3,1
```

6.5 Amber MDOUT Options

Note that multiple MDOUT files can be specified in a single ***readdata*** command.

6.6 Evecs File Options

Read

`[ibeg <firstmode>] [iend <lastmode>]`

ibeg <firstmode> Number of the first mode (or principal component) to read from evecs file. Default 1.

iend <lastmode> Number of the last mode (or principal component) to read from evecs file. Default is to read all for newer evecs files (generated by *cpptraj* version > 12), 50 for older evecs files.

6.7 Vector psuedo-traj Options

This can be used to write out a representation of a vector data set which can then be visualized. See 11.84 on page 157 for more on generating vector data sets.

Write

`[trajfmt <format>] [parmout <file>] [noorigin]`

trajfmt <format> Output pseudo-trajectory format. See 10 on page 58 for trajectory format keywords.

parmout <**file**> File to write pseudo-trajectory topology to.

[noorigin] Do not write vector origin coordinates.

6.8 OpenDX file options

Write

[bincenter] [gridwrap] [gridext]

bincenter Center grid points on bin centers instead of corners.

gridwrap Like 'bincenter', but also wrap grid density. Useful when grid encompasses unit cell.

gridext Like 'bincenter', but also print extra layer of empty bins.

6.9 CCP4 file options

Write

[title <title>]

[title <**title**>**]** Set CCP4 output title.

6.10 Charmm REPD log options

Read

[nrep <#>] [crdidx <crd indices>]

nrep <#> Total number of replicas.

crdidx <**crd indices**> Comma-separated list of indices to use as initial coordinate indices.

6.11 Amber Constant pH Out options

Read

cpin <file>

cpin <**file**> Constant pH input (CPIN) file name.

Note that when reading in constant pH data the data set aspect will be set to the residue name and the index will be set to the residue number. When reading in constant pH REMD data the data is unsorted, and sortensembledata should be used to create sorted constant pH data sets (see 8.27 on page 48).

7 Coordinates (COORDS) Data Set Commands

Coordinate I/O tends to be the most time-consuming part of trajectory analysis. In addition, many types of analyses (for example two-dimensional RMSD and cluster analysis) require using coordinate frames multiple times. To simplify this, trajectory coordinates may be saved as a separate data set via the ***loadcrd*** command or ***createcrd*** action. Any action can then be performed on the COORDS data set with the ***crdaction*** command. The ***crdout*** command can be used to write coordinates to an output trajectory (similar to ***trajout***).

Although COORDS data sets store everything internally with single-precision, they can still use a large amount of memory. Because of this there is a specialized type of COORDS data set called a TRAJ data set (trajectory), which functions exactly like a COORDS data set except all data is stored on disk. TRAJ data sets can be created with the ***loadtraj*** command. *TRAJ data sets cannot be modified.*

There are several analyses that can be performed using COORDS data sets, either as part of the normal analysis list or via the ***runanalysis*** command. Note that while these analyses can be run on specified COORDS data sets, if one is not specified a default COORDS data set will be created, made up of frames from ***trajin*** commands.

As an example of where this might be useful is in the calculation of atomic positional fluctuations. Previously this required two steps: one to generate an average structure, then a second to rms-fit to that average structure prior to calculating the fluctuations. This can now be done in one pass with the following input:

```
parm topology.parm7
loadcrd mdcrd.nc
# Generate average structure PDB, @CA only
crdaction mdcrd.nc average avg.pdb @CA
# Load average structure PDB as reference
parm avg.pdb
reference avg.pdb parm avg.pdb
# RMS-fit to average structure PDB
crdaction mdcrd.nc rms reference @CA
# Calculate atomic fluctuations for @CA only
crdaction mdcrd.nc atomicfluct out fluct.dat bfactor @CA
```

The following COORDS data set commands are available:

Command	Description
catcrd	Concatenate two or more COORDS sets.
combinecrd	Combine two or more COORDS sets.
crdaction	Run a single Action on a COORDS set.
crdout	Write a COORDS set to a file.
createcrd	(Action) Create a COORDS set during a Run.
loadcrd	Create or append to a COORDS set from a file.
loadtraj	Create special COORDS set where frames remain on disk.
permutedihedrals	Rotate specified dihedral(s) in given COORDS set by specific interval or to random valu
reference	Load a single trajectory frame as a reference.
rotatedihedral	Rotate specified dihedral to specified value or by given increment.
splitcoords	Split molecules in a COORDS set into a trajectory.

7.1 catcrd

```
catcrd <crd1> <crd2> ... name <name>
```

<crdX> COORDS data sets to concatentate, specify 2 or more.

name <name> New COORDS set name

Concatentate two or more COORDS data sets into a single COORDS data set. The topologies must have the same number of atoms for this to work. If the topologies differ in other ways, the topology of the first COORDS set takes priority.

7.2 combinecrd

```
combinecrd <crd1> <crd2> ... [parmname <topname>] [crdname <crdname>]
```

<crdX> COORDS data sets to combine, specify 2 or more.

[parmname <topname>] Name of combined Topology.

[crdname <crdname>] Name of combined COORDS data set.

Combined two or more COORDS data sets into a single COORDS data set. Note that the resulting topology will most likely **not** be usable for MD simulations. Box information will be retained - the largest box dimensions will be used.

For example, to load two MOL2 files as COORDS data sets, combine them, and write them out as a single MOL2:

```
loadcrd Tyr.mol2 CRD1
loadcrd Pry.mol2 CRD2
combinedcrd CRD1 CRD2 parmname Parm-1-2 crdname CRD-1-2
crdout CRD-1-2 Tyr.Pry.mol2
```

7.3 crdaction

```
crdaction <crd set> <actioncmd> [<action args>] [crdframes <start>,<stop>,<offset>]
```

Perform action <actioncmd> on COORDS data set <crd set>. A subset of frames in the COORDS data set can be specified with 'crdframes'.

For example, to calculate RMSD for a previously created COORDS data set named crd1 using frames 1 to the last, skipping every 10:

```
crdaction crd1 rmsd first @CA out rmsd-ca.agr crdframes 1,last,10
```

7.4 createcrd

This command is actually an Action that can be used to create COORDS data sets during trajectory processing, see 11.19 on page 85.

7.5 crdout

```
crdout <crd set> <filename> [<trajout args>] [crdframes <start>,<stop>,<offset>]
```

Write COORDS data set <crd set> to trajectory named <filename>. A subset of frames in the COORDS data set can be specified with 'crdframes'.

For example, to write frames 1 to 10 from a previously created COORDS data set named "crd1" to separate PDB files:

```
crdout crd1 crd1.pdb multi crdframes 1,10
```

7.6 loadcrd

```
loadcrd <filename> [parm <parm> | parmindex<#>] [<trajin args>] [name <name>]
```

Immediately load trajectory <filename> as a COORDS data set named <name> (default base name of <filename>). If <name> is already present the coordinates will be appended to the existing data set.

For example, to load frames from trajectories named 'traj1.nc' and 'traj2.nc' into a COORDS data set named Crd1:

```
loadcrd traj1.nc name Crd1
loadcrd traj2.nc name Crd2
```

7.7 loadtraj

```
loadtraj name <setname> [<filename>]
```

This command functions in two ways. If <filename> is not provided, all currently loaded input trajectories (from ***trajin*** commands) are added to TRAJ data set named <setname>. **Note that if the input trajectory list is cleared (via 'clear trajin') this will invalidate the TRAJ data set.** In addition, currently all trajectories must have the same number of atoms. Otherwise add trajectory <filename> to TRAJ data set <setname>.

TRAJ data sets cannot be modified.

7.8 permutedihedrals

```
permutedihedrals crdset <COORDS set> resrange <range> [{interval | random}]
              [outtraj <filename> [<outfmt>]] [crdout <output COORDS>] [<dihedral types
  Options for 'random':
[rseed <rseed>] [out <# problems file> [<set name>]]
[ check [cutoff <cutoff>] [rescutoff <rescutoff>] [checkallresidues]
  [backtrack <backtrack>] [increment <increment>] [maxfactor <max_factor>] ]
  Options for 'interval':
<interval deg>
  <dihedral types> =  alpha beta gamma delta epsilon zeta nu1 nu2 h1p c2p chin phi psi
```

crdset <COORDS set> COORDS data set to operate on.

resrange <range> Residue range to search for dihedrals.

interval Rotate found dihedrals by <interval>. This is done in an ordered fashion so that every combination of dihedral rotations is sampled at least once.

random Rotate each found dihedral randomly.

[outtraj <filename>] Trajectory file to write coordinates to.

[<outfmt>] Trajectory file format.

[crdout <output COORDS>] COORDS data set to write coordinates to.

<dihedral type> One or more dihedral types to search for.

Options for 'interval:

<interval deg> Amount to rotate dihedral by each step.

Options for 'random':

[rseed <rseed>] Random number seed.

[out <# problems file>] File to write number of problems (clashes) each frame to.

[<set name>] Number of problems data set name.

[check] Check randomly rotated structure for clashes.

[cutoff <cutoff>] Atom cutoff for checking for clashes (default 0.8 Å).

[rescutoff <cutoff>] Residue cutoff for checking for clashes (defualt 10.0 Å).

[checkallresidues] If specified all residues checked for clashes, otherwise only residues up to the currently rotated dihedral check.

[backtrack <backtrack>] If a clash is encountered at dihedral N and cannot be resolved, go to dihedral N-<backtrack> to try and resolve the clash (default 4).

[increment <increment>] If a clash is encountered, first attempt to rotate dihedral by increment to resolve it; if it cannot be resolved by a full rotation the calculation will backtrack (default 1).

[maxfactor <max_factor>] The maximum number of total attempted rotations will be <max_factor> * <total # of dihedrals> (default 2).

Create a trajectory by rotating specified dihedrals in a structure by regular intervals (**interval**), or create 1 structure by randomly rotating specified dihedrals (**random**). When randomly rotating dihedrals steric clashes will be checked if **check** is specified; in such cases the algorithm will attempt to resolve the clash as best it can. If clashes are not being resolved you can increase the number of rotation attempts *cpptraj* will make by increasing **maxfactor**.

For example, to rotate all backbone dihedrals in a protein with coordinates in a file named tz2.rst7 in -120 degree intervals and write the resulting trajectory in Amber format to rotations.mdcrd:

```
reference tz2.rst7 [TZ2]
permutedihedrals crdset [TZ2] interval -120 outtraj rotations.mdcrd phi psi
```

To randomly rotate backbone dihedrals for the same structure and write to file random.mol2 in MOL2 format:

```
reference tz2.rst7 [TZ2]
permutedihedrals crdset [TZ2] random rseed 1 check maxfactor 10 phi psi \
                 outtraj random.mol2 multi
```

7.9 reference

Reference coordinates can now be used and manipulated like COORDS data sets. See 10.3 on page 60 for command syntax.

7.10 rotatedihedral

```
rotatedihedral crdset <COORDS set> [frame <#>] [name <output set name>]
               {value <value> | increment <increment>}
               { <mask1> <mask2> <mask3> <mask4> |
                 res <#> type <dih type> }
<dih type> = alpha beta gamma delta epsilon zeta nu1 nu2 h1p c2p chin phi psi chip omeg
```

crdset <COORDS set> Coordinates data set to work on. If a TRAJ data set is specified, name must also be specified.

[frame <#>] Frame of the COORDS set to work on.

[name <output set name>] Output COORDS set. If not specified the input COORDS set will be modified.

value <value> Set specified dihedral to given value in degrees.

increment <increment> Increment specified dihedral by increment in degrees.

<mask1> <mask2> <mask3> <mask4> Define dihedral by atom masks. Each mask should only select one atom.

res <#> Rotate dihedral specified by type in residue number <#>.

type <dih type> Dihedral type to rotate in specified residue.

Rotate the specified dihedral in given COORDS set to a target value or by given increment. For example, to set the protein chi dihedral in residue 8 to 35 degrees and write out to a mol2 file:

```
parm ../tz2.parm7
loadcrd ../tz2.nc 1 1 name TZ2
rotatedihedral crdset TZ2 value 35 res 8 type chip
crdout TZ2 tz2.rotate.1.mol2
```

7.11 splitcoords

```
splitcoords <crd set> name <output set name>
```

<crd set> COORDS set to split.

name <output set name> `Name of new set to create.`

Split trajectory specified by <crd set> by molecule into a new COORDS set. All molecules in <crd set> must be the same size. For example, if there are 10 molecules and 10 frames in COORDS set "Set0", the following would create a new COORDS set with 100 frames (original molecules 1-10 frame 1, original molecules 1-10 frame 2, etc):

```
splitcoords Set0 name Set0Split
```

8 General Commands

The following general commands are available:

Command	Description
activeref	Select the reference for distance-based masks.
calc	Evaluate the given mathematical expression.
clear	Clear various objects from the cpptraj state.
create	Create (but do not yet write) a data file.
createset	Create a dataset from a simple mathematical expression.
datafile	Used to manipulate data files.
datafilter	Filter data sets based on given criteria.
dataset	Use to manipulate data sets.
debug | prnlev	Set debug level. Higher levels give more info.
ensextension	Enable/disable ensemble number extension for files in ensemble mode.
exit | quit	Quit cpptraj.
go | run	Start a trajectory processing Run.
help	Provide help for commands.
list	List various objects in the cpptraj state.
noexitonerror	Attempt to continue even if errors are encountered.
noprogress	Do not print a progress bar during a Run.
parallelanalysis	(MPI only) Divide current Analyses among MPI processes.
precision	Change the output precision of data sets.
printdata	Print data set to screen.
readdata	Read data sets from files.
readensembledata	Read data files in ensemble mode.
readinput	Read cpptraj input from a file.
removedata	Remove specified data set(s).
rst	Generate Amber-style distance/angle/torsion restraints.
runanalysis	Run an analysis immediately or run all queued analyses.
select	Print the results of an atom mask expression.
selectds	Print the results of a data set selection expression.
silenceactions	Prevent Actions from writing information to STDOUT.
sortensembledata	Sort data sets using replica information (currently constant pH only).
usediskcache	Turn caching of data sets to disk on or off.
write | writedata	Immediately write data to a file or write to all current data files.

8.1 activeref

```
activeref <#>
```

Set which reference structure should be used when setting up distance-based masks for everything but the 'mask' action. Numbering starts from 0, so 'activeref 0' selects the first reference structure read in, 'activeref 1' selects the second, and so on.

8.2 calc

```
calc <expression>
[prec <width>.<precision>] [format {double|general|scientific}]
```

<expression> Mathematical expression to evaluate.
See 5.2 on page 22 for details.

prec <width>.<precision> Set the width and precision of the result.

format {double|general|scientific} Set the format of the result.

Evaluate the given mathematical expression. This version gives more control over the format of the output.

8.3 clear

```
clear [{all | <type>}]
      (<type> = actions,trajin,trajout,ref,parm,analysis,datafile,dataset)
```

Clear list of indicated type, or all lists if 'all' specified. Note that when clearing actions or analyses, associated data sets and data files are not cleared and vice versa.

8.4 create

```
create <filename> <datasetname0> [<datasetname1> ...] [<DataFile Options>]
```

Add specified data sets to the data file named <filename>; if the file does not exist, it will be added to the DataFileList. Data files created in this way are only written at the end of coordinate processing, analyses, or via the ***'writedata'*** command. See 6 on page 23 for more data file format options.

8.5 createset

```
createset <expression> [xmin <min>] xstep <step> nx <nxvals>
```

expression Simple mathematical expression, must contain equals sign, can contain X (e.g. Y=2*X). If not enclosed in quotes must not contain whitespace.

xmin <min> Minimum X value.

xstep <step> X step.

nx <nxvals> Number of X values.

Generate a data set from a simple mathematical expression.

8.6 datafile

```
datafile <filename> <datafile arg>
```

Pass <datafile arg> to data file <filename>. See 6 on page 23 for more details.

8.7 datafilter

```
datafilter <dataset arg> min <min> max <max> [out <file> [name <setname>]]
```

<dataset arg> min <min> max <max> Data set and min/max cutoffs to use; can specify more than once.

[out <file>] Write out to file named <file>.

[name <setname>] Name of filter data set.

Create a data set (optionally named <setname>) containing 1 for data within given **<min>** and **<max>** criteria for each specified data set. There must be at least one **<min>** and **<max>** argument, and can be as many as there are specified data sets. For example, to read in data from two separate files (d1.dat and a1.dat) and generate a filter data set named FILTER having 1 when d1 is between 0.0 and 3.0 and a1 is between 135.0 and 180.0:

```
readdata a1.dat name a1
readdata d1.dat name d1
datafilter d1 min 0.0 max 3.0 a1 min 135.0 max 180.0 out filter.dat name FILTER
```

Note that a similar command that can be used with data generated by Actions during trajectory processing is ***filter*** (see page 96).

8.8 dataset

```
dataset { legend <legend> <set> |
          makexy <Xset> <Yset> [name <name>] |
          vectorcoord {X|Y|Z} <set> [name <name>] |
       cat <set0> <set1> ... [name <name>] [nooffset] |
          make2d <1D set> cols <ncols> rows <nrows> [name <name>] |
          {drop|keep}points {range <range arg> | [start <#>] [stop <#>] [offset <#>]}
                   [name <output set>] <set arg1> ... |
          remove <criterion> <select> <value> [and <value2>] [<set selection>] |
          dim {xdim|ydim|zdim|ndim <#>} [label <label>] [min <min>] [step <step>] |
          outformat {double|scientific|general} <set arg1> [<set arg 2> ...] |
          invert <set arg0> ... name <new name> [legendset <set>] |
       [mode <mode>] [type <type>] <set arg1> [<set arg 2> ...]
        }
  <mode>:  'distance' 'angle' 'torsion' 'pucker' 'rms' 'matrix' 'vector'
  <type>:  'alpha' 'beta' 'gamma' 'delta' 'epsilon' 'zeta' 'nu0' 'nu1' 'nu2' 'nu3'
```

'nu4' 'h1p' 'c2p' 'chin' 'phi' 'psi' 'chip' 'omega' 'chi2' 'chi3' 'chi4'
'chi5' 'pucker' 'noe' 'distance' 'covariance' 'mass-weighted covariance'
'correlation' 'distance covariance' 'IDEA' 'IRED' 'dihedral covariance'
Options for 'type noe':
[bound <lower> bound <upper>] [rexp <expected>] [noe_strong] [noe_medium] [noe_w

[name <name>] New data set name for makexy/vectorcoord/cat/make2d/droppoints/keeppoints.

legend <legend> <set> Set the legend for data set <set> to <legend>.

makexy <Xset> <Yset> Create a new data set (optionally named <name>) with X values from <Xset> and Y values from <Yset>.

vectorcoord {X|Y|Z} <set> Extract X/Y/Z coordinates from vector data set into a new 1D data set.

cat <set0> <set1> ... Concatenate two or more data sets into a new data set (optionally named <name>).

make2d <1D set> cols <ncols> rows <nrows> Convert 1D data set into row-major 2D data set with specified number of rows and columns.

{drop|keep}points <set arg1> ... Drop or keep specified points from data set(s), optionally creating a new data set.

range <range arg> Range of points to drop/keep.

[start <#>] [stop <#>] [offset <#>]
Start/stop/offset values of points to drop/keep.

remove <criterion> <select> <value> [and <value2>] [<set selection>]
Remove data sets from <set selection> according to specified criterion and selection.

<criterion>: 'ifaverage' 'ifsize' 'ifmode' 'iftype'
<select> : 'equal' '==' 'notequal' '!=' 'lessthan' '<' 'greaterthan' '>' 'betwee

dim {xdim|ydim|zdim|ndim <#>} Change specified dimension in set(s).

label <label> Change dimension label to <label>

min <min> Change dimension minimum to <min>.

step <step> Change dimension step to <step>.

invert <set arg0> ... name <new name> [legendset <set>]

<set arg0> ... Specify sets to invert.

name <new name> Inverted output set name.

[legendset <set>] String data set containing legends

[mode <mode>] Set data set(s) mode to <mode>.

[type <type>] Set data set(s) type to 'type', useful for e.g. analysis with *statistics*. Note this can also be done with 'type <type>' for certain commands (*distance*, *dihedral*, *pucker* etc). Note that not every <type> is compatible with a given <mode>.

Options for type noe only:

[bound <lower> bound <upper>] Lower and upper bounds for NOE (in Angstroms); must specify both.

[rexp <expected>] Expected value for NOE (in Angstroms); if not given '(<lower> + <upper>)' / 2.0 is used.

[noe_strong] Set lower and upper bounds to 1.8 and 2.9 Å respectively.

[noe_medium] Set lower and upper bounds to 2.9 and 3.5 Å respectively.

[noe_weak] Set lower and upper bounds to 3.5 and 5.0 Å respectively.

Either set the legend for a single data set, create a new set with X values from one set and Y values from another, concatenate 2 or more sets, make a 2D set from 1D set, remove sets according to a certain criterion, or change the mode/type for one or more data sets.

Setting the mode/type can be useful for cases where the data set is being read in from a file; for example when reading in a dihedral data set the type can be set to 'dihedral' so that various Analysis routines like ***statistics*** know to treat it as periodic. A brief description of possible modes and types follows:

Mode	Type	Description
distance	noe	NOE distance.
angle		Angle.
torsion	alpha	Nucleic acid alpha.
	beta	Nucleic acid beta.
	gamma	Nucleic acid gamma.
	delta	Nucleic acid delta.
	epsilon	Nucleic acid epsilon.
	zeta	Nucleic acid zeta.
	nu1	Nucleic pucker (O4').
	nu2	Nucleic pucker (C4').
	h1p	Nucleic acid H1'.
	c2p	Nucleic acid C2'.
	chin	Nucleic acid chi.
	phi	Protein Phi.
	psi	Protein psi.
	chip	Protein chi.
	omega	Protein omega.
pucker	pucker	Sugar pucker.
rms		RMSD.
matrix	distance	Distance matrix.
	covariance	Cartesian covariance matrix.
	'mass-weighted covariance'	Mass weighted Cartesian covariance matrix.
	correlation	Dynamic cross correlation matrix.
	'distance covariance'	Distance covariance matrix.
	IDEA	IDEA matrix.
	IRED	IRED matrix.
	'dihedral covariance'	Dihedral covariance matrix.
vector	IRED	IRED vector.

The invert mode takes a group of M 1D data sets of size N and create N new "inverted" data sets of size M. This is similar to the invert keyword already available for standard and Grace data writes, but operates directly on data sets. For example, given the following two data sets:

```
D0 D1
 1  4
 2  5
 3  6
```

The new data sets will be laid out like so:

```
N0 N1 N2
 1  2  3
 4  5  6
```

The dataset invert command can be useful if you want to easily view output from multiple analysis commands in a single graph. For example, to view state counts from two different simulations side by side:

```
calcstate name Sim1 state bound1,dist1,0.0,2.0
calcstate name Sim2 state bound1,dist1,0.0,2.0
runanalysis dataset invert Sim*[Count] name Inverted legendset Sim1[Name]
dataset dim xdim label Simulation min 1 step 1 Inverted*
writedata statecount.agr Inverted*
```

8.9 debug | prnlev

```
debug [<type>] <#>
      (<type> = actions,trajin,trajout,ref,parm,analysis,datafile,dataset)
```

Set the level of debug information to print. In general the higher the <#> the more information that is printed. If <type> is specified only set the debug level for a specific area of *cpptraj*.

8.10 ensextension

```
ensextension {on|off}
```

Turn printing of ensemble member number filename extensions on or off. By default ensemble extensions are printed in parallel and not in serial.

NOTE: THE 'ensextension off' OPTION HAS NOT BEEN FULLY TESTED IN PARALLEL AND IS NOT CURRENTLY RECOMMENDED.

8.11 exit | quit

Exit normally.

8.12 go | run

Begin trajectory processing, followed by analysis and datafile write.

8.13 help

```
help [ { All |
     <cmd> |
      <command category> |
      Form[ats] [{read|write}] |
      Form[ats] [{trajin|trajout|readdata|writedata|parm|parmwrite} [<fmt key>]] |
      Mask     } ]
Command Categories: Gen[eral] Sys[tem] Coor[ds] Traj[ectory] Top[ology]
                    Act[ion] Ana[lysis] Con[trol]
```

```
All                : Print all known commands.
<cmd>              : Print help for command <cmd>.
<command category> : Print all commands in specified category.
Form[ats]          : Help for file formats.
Mask               : Help for mask syntax.
```

If 'All' is specifed, list all commands known to *cpptraj*. If given with a command, print help for that command. Otherwise, list all commands of a certain category (General, System, Coords, Trajectory, Topology, Action, Analysis, or Control), help for various file formats, or help with atom mask syntax.

8.14 list

```
list <type>
     (<type> = actions,trajin,trajout,ref,parm,analysis,datafile,dataset)
```

List the currently loaded objects of <type>. If no type is given then list all loaded objects.

8.15 noexitonerror

```
noexitonerror
```

Normally *cpptraj* will exit if actions fail to initialize properly. If ***noexitonerror*** is specified, *cpptraj* will attempt to continue past such errors. This is the default if in interactive mode.

8.16 noprogress

```
noprogress
```

Do not display read progress during trajectory processing.

8.17 parallelanalysis

```
parallelanalysis [sync]
```

MPI only. Divide all currently set up analyses as evenly as possible among available MPI processes and execute. Each analysis will get a single MPI process. If **sync** is specified all data will be synced back to the master thread (for e.g. subsequent analysis). For an example of how to use the parallelanalysis command, see 12.14 on page 185.

8.18 precision

```
precision {<filename> | <dataset arg>} [<width>] [<precision>]
```

Set the precision for all data sets in data file <filename> or data set(s) specified by <dataset arg> to *width.precision*, where width is the column width and precision is the number of digits after the decimal point. Note that the <precision> argument only applies to floating-point data sets.

For example, if one wanted to set the precision of the output of an Rmsd calculation to 8.3, the input could be:

```
trajin ../run0.nc
rms first :10-260 out prec.dat
precision prec.dat 8 3
```

and the output would look like:

```
#Frame RMSD_00000
1 0.000
2 0.630
```

8.19 readdata

`readdata <filename>  [name <dsname>] [as <fmt>] [separate] [<format options>]`

name <**dsname**> `Name for read-in data set(s).  Default is <filename>.`

as <**fmt**> `Force <filename> to be read as a specific format using given format keyword.`

separate `Read each file specified into separate data sets indexed from 0.`

Read data from file <filename> and store as data sets. For more information on formats currently recognized by cpptraj see 1 on page 21. For format-specific options see 6. For example, given the file calc.dat:

```
#Frame  R0   D1
1      1.7 2.22
```

The command '`readdata calc.dat`' would read data into two data sets, calc.dat:2 (legend set to "R0") and calc.dat:3 (legend set to "D1").

8.20 readensembledata

`readensembledata <filename> [filenames <additional files>] [<readdata args>]`

<**filename**> `Lowest replica file name.`

filenames <**additional files**> `Specified additional members of the ensemble.  If not specified ensemble members will be search for using numerical extensions.`

<**readdata args**> Additional data file arguments.

Read data sets as an ensemble, i.e. each file is a different member of an ensemble. This command is MPI-aware.

If one filename is given, it is assumed it is the "lowest" member of an ensemble with a numerical extension, e.g. 'file.001' and the remaining files are searched for automatically. Otherwise all other members of the ensemble can be specified with **'filenames'** and a comma-separated list e.g. 'file.001 filenames file.002,file.003,file.004. For additional 'readdata' arguments that can be passed in see 6 on page 23.

For example, to read in data files named cpout.001 to cpout.006 automatically:

```
readensembledata cpout.001 cpin cpin name PH
```

Or specified:

```
readensembledata cpout.001 \
                 filenames cpout.002,cpout.003,cpout.004,cpout.005,cpout.006 \
                 cpin cpin name PH
```

8.21 readinput

```
readinput <filename>
```

Read *cpptraj* commands from file <filename>.

8.22 removedata

```
removedata <arg>
```

Remove data set corresponding to <arg>.

8.23 rst

```
rst <mask1> <mask2> [<mask3>] [<mask4>]
    r1 <r1> r2 <r2> r3 <r3> r4 <r4> rk2 <rk2> rk3 <rk3>
    {[parm <parmfile / tag> | parmindex <#>]}
    [{ref <refname> | refindex <#> | reference} [offset <off>] [width <width>]]
    [out <outfile>]
```

<**mask1**> (Required) First atom mask.

<**mask2**> (Required) Second atom mask. If only two masks assume distance restraint.

[<**mask3**>] (Optional) Third atom mask. If 3 atom masks assume angle restraint.

[<mask4>] (Optional) Fourth atom mask. If 4 atom masks assume dihedral restraint.

rX <rX> Value of RX (X=1-4, default 0.0)

rk2 <rk2> Value of RK2 (force constant to be applied when R is R1 <= R < R2)

rk3 <rk3> Value of RK3 (force constant to be applied when R is R3 <= R < R4)

[parm <parmfile / tag> | parmindex <#>] Topology to be used for atom masks.

{ref <refname> | refindex <#> | reference} Use distance/angle/dihedral in reference structure to determine values for r1, r2, r3, and r4. The value of r2 is set to <r2> + <off>, r3 = r2, r1 = r2 - <width>, r4 = r3 + <width>.

[offset <off>] (Reference only) Value to offset distance/angle/torsion in reference by (default 0.0).

[width <width>] (Reference only) Width between r1 and r2, r3 and r4 (default 0.5).

[out <outfile>] Write restraints to outfile. If not specified, write to STDOUT.

Generate Amber-style distance restraints for use with nmropt=1. This is particularly useful for generating distance restraints based off of reference coordinates. For example to generate a distance restraint between two C5' atoms using the current distance between them in a reference structure, offsetting the distance by 1.0 Ang.:

```
parm 30bp-longbox-tip3p-na.parm7
reference 30bp-longbox.rst7
rst :1@C5' :31@C5' reference offset 1.0 rk2 10.0 rk3 10.0 out output
```

8.24 runanalysis

```
runanalysis [<analysiscmd> [<analysis args>]]
```

Run given analysis command immediately and write any data generated. If no command is given run any analysis currently set up. NOTE: When 'runanalysis' is specified alone, data is not automatically written; to write data generated with 'runanalysis' use the 'writedata' command (this allows multiple analysis runs between output if desired).

8.25 select

```
select <mask>
```

Prints the number of selected atoms corresponding to the given mask, as well as the atom numbers with format:

```
Selected= <#atom1> <#atom2> ...
```

This does not affect the state in any way, but is intended for use in scripts etc. for testing the results of a mask expression.

8.26 selectds

```
selectds <dataset arg>
```

Show the results of a data set selection. Data set selection has the format:

```
<name>[<aspect>]:<index>
```

Either the [<aspect>] or the <index> arguments may be omitted. A '*' can be used in place of <name> or [<aspect>] as a wildcard. The <index> argument can be a single number or a range separated by '-' and ','.

This command does not affect the state in any way, but is particularly useful in interactive mode for determining the results of a dataset argument.

8.27 sortensembledata

```
sortensembledata <dset arg0> [<dset arg1> ...]
```

<dset arg0> [<dset arg1> ...] `Data set(s) to sort.`

Sort unsorted data sets. Currently only works for constant pH REMD data.

8.28 usediskcache

```
usediskcache {on|off}
```

If on, CPPTRAJ will attempt to cache data sets to disk if possible. This currently only works for integer data sets (e.g. ***hbond* series** data sets, etc).

8.29 write | writedata

```
write [<filename> <datasetname0> [<datasetname1> ...]] [<DataFile Options>]
```

With no arguments, write all files currently in the data file list. Otherwise, write specified data set(s) to <filename>. This is like the 'create' command except a data file is not added to the data file list; it is written immediately. See 6 on page 23 for more data file format options.

8.30 System Commands

These commands call the equivalent external system commands.

gnuplot <args> Call `gnuplot` (if it is installed on your system) with the given arguments.

head <args> Call `head`, which lists the first few lines of a file.

less <args> Call `less`, which can be used to view the contents of a file.

ls <args> List the contents of a directory.

pwd <args> Print the current working directory.

xmgrace <args> Call `xmgrace` (if it is installed on your system) with the given arguments.

9 Topology File Commands

These commands control the reading and writing of topology files. Cpptraj supports the following topology file formats:

Format	Keyword	Extension	Notes
Amber Topology	amber	.parm7	Only fully-supported format for write.
PDB	pdb	.pdb	Read Only
Mol2	mol2	.mol2	Read Only
CIF	cif	.cif	Read Only
Charmm PSF	psf	.psf	Limited Write
Gromacs Topology	gromacs	.top	Read only
SDF	sdf	.sdf	Read Only
Tinker ARC	arc	.arc	Read Only

For most commands that require a topology one can be specified via two keywords:

parm [<name>] Select topology corresponding to given file name, tag, or name.

parmindex [<#>] Select topology by order in which it was loaded, starting from 0.

The following topology related commands are available:

Command	Description
angleinfo, angles, printangles	Print angle info for selected atoms.
atominfo, atoms, printatoms	Print details for selected atoms.
bondinfo, bonds, printbonds	Print bond info for selected atoms.
change	Change specified parts of a topology.
charge	Print total charge for selected atoms.
comparetop	Compare two topologies and report differences.
dihedralinfo, dihedrals, printdihedrals	Print dihedral info for selected atoms.
mass	Print total mass for selected atoms.
molinfo	Print molecule info for selected atoms.
parm	Load a topology file.
parmbox	Modify box info for a loaded topology.
parminfo	Print details for selected topology.
parmstrip	Remove selected atoms from topology.
parmwrite	Write selected topology to file.
resinfo	Print residue info for selected atoms.
scaledihedralk	Scale selected dihedral force constants.
solvent	Change which molecules are considered solvent.

9.1 angleinfo | angles | printangles

```
angleinfo [parm <name> | parmindex <#> | <#>] [<mask>]
```

Print angle information of atoms in <mask> for selected topology (first loaded topology by default) with format:

```
# Angle Kthet degrees atom names (numbers)
```

Where `Angle` is the internal angle index, `Kthet` is the angle force constant, `degrees` is the angle equilibrium value, `atom names` shows the atoms involved in the angle with format `:<residue num>@<atom name>`, and `(numbers)` shows the atom indices involved in a comma-separated list. Atom types will be shown in the last column.

9.2 atominfo | atoms | printatoms

```
atominfo [parm <name> | parmindex <#> | <#>] [<mask>]
```

Print information on atoms in <mask> for selected topology (first loaded topology by default) with format:

```
#Atom Name #Res Name #Mol Type Charge Mass GBradius El [rVDW] [eVDW]
```

where `#Atom` is the internal atom index, the first `Name` column is the atom name, `#Res` is the atom's residue number, the second `Name` column is residue name,

`#Mol` is the atom's molecule number, `Type` is the atom's type (certain topologies only), `Charge` is the atom charge (in units of electron charge), `Mass` is the atom's mass (in amu), `GBradius` is the generalized Born radius of the atom (Amber topologies only), and `El` is the 2 character element string. The final two columns are only shown if the topology contains non-bonded parameters: `rVDW` is the atom's Lennard-Jones radius and `eVDW` is the atom's Lennard-Jones epsilon.

9.3 bondinfo | bonds | printbonds

```
bondinfo [parm <name> | parmindex <#> | <#>] [<mask>]
```

Print bond information for atoms in <mask> for selected topology (first loaded topology by default) with format:

```
# Bond Kb Req atom names (numbers)
```

where `Bond` is the internal bond index, `Kb` is the bond force constant, `Req` is the bond equilibrium value (in Angstroms), `atom names` shows both atom names with format `:<residue num>@<atom name>`, and `(numbers)` shows both atom numbers in a comma-separated list. Atom types will be shown in the last column.

9.4 change

```
change [parm <name> | parmindex <#> | <#> |
        crdset <COORDS set> ]
{ resname from <mask> to <value> |
      chainid of <mask> to <value> |
  atomname from <mask> to <value> |
      addbond <mask1> <mask2> [req <length> <rk> <force constant>] }
```

parm <name> | parmindex <#> | <#> | crdset <COORDS set>
Topology to change.

resname from <mask> to <value> Change residue names for residues in <mask> to the given <value>.

chainid of <mask> to <value> Change the chain ID of residues in <mask> to given <value>.

atomname from <mask> to <value> Change atom names for atoms in <mask> to the given <value>.

addbond <mask1> <mask2> Add bond between atom specified by <mask1> and atom specified by <mask2>.

[req <length>] The equilibrium bond length in Angstroms.

[rk <force constant>] The bond force constant in kcal/mol*Angstrom.

Change specified parts of the specified topology. For example, to change atoms named 'HN' to 'H' in topology 0:

```
change parmindex 0 atomname from @HN to H
```

9.5 charge

```
charge [parm <name> | parmindex <#> | <#>] <mask>
```

Print the total charge of atoms in <mask> (in units of electron charge) for selected topology (first loaded topology by default).

9.6 comparetop

```
comparetop {parm <name> | parmindex <#>} {parm <name> | parmindex <#>} [out <file>]
           [atype] [lj] [bnd] [ang] [dih] [atoms]
```

parm <**name**> | **parmindex** <#> Topologies to compare.

out <**file**> Print results to file instead of screen.

[atype] Only report atom type differences.

[lj] Only report differences in Lennard-Jones parameters.

[bnd] Only report differences in bond parameters.

[ang] Only report differences in angle parameters.

[dih] Only report differences in dihedral parmeters.

[atoms] Only report differences in atom properties.

Compare and report differences in atoms/parameters between two topologies. Differences are reported in standard 'diff' format, with '<' prefix indicating the parameter is from the first topology and '>' prefix indicating the parameter is from the second topology.

9.7 dihedralinfo | dihedrals | printdihedrals

```
dihedralinfo [parm <name> | parmindex <#> | <#>] [<mask>] [and]
```

Print dihedral information of atoms in <mask> for selected topology (first loaded topology by default) with format:

```
#Dihedral pk phase pn atoms
```

where `#Dihedral` is the internal dihedral index, `pk` is the dihedral force constant, `phase` is the dihedral phase, `pn` is the dihedral periodicity, and `atoms` shows the names of the atoms involved in the angle with format `:<residue num>@<atom name>`, followed by the atom indices involved in a comma-separated list. In

addition if the dihedral is an end dihedral, improper dihedral, or both it will be prefaced with an `E`, `I`, or `B` respectively. Atom types will be shown in the last column.

If the **and** keyword is specified then printed dihedrals must contain all atoms specified in the <mask>. For example, to only get dihedrals containing the atom types N3, CT, and OH:

```
dihedrals @%N3,CT,OH and
```

9.8 mass

```
[<parmindex>] [parm <name> | parmindex <#> | <#>] <mask>
```

Print the total mass of atoms in <mask> (in amu) for selected topology (first loaded topology by default).

9.9 molinfo

```
molinfo [parm <name> | parmindex <#> | <#>] [<mask>]
```

Print molecule information for atoms in <mask> for selected topology (first loaded topology by default) with format:

```
#Mol  Natom  #Res Name [SOLVENT]
```

where `#Mol` is the molecule number, `Natom` is the number of atoms in the molecule, and `#Res` and `Name` are the residue number and residue name of the first residue in the molecule respectively. `SOLVENT` will be printed if the molecule is currently considered a solvent molecule.

9.10 parm

```
parm <filename> [{[TAG] | name <setname>}]
     [{ nobondsearch |
        [bondsearch <offset>] [searchtype {grid|pairlist}]
      }]
```

<filename> Parameter file to read in; format is auto-detected.

'[TAG]' Optional tag (bounded in brackets) which can be referred to in place of the topology file name in order to simplify references to it (see 3.4 on page 16 for examples of how to use tags).

[name <setname>] Optional name that can be used to refer to the topology in place of the file name.

[nobondsearch] Optional; if specified do not search for bonds via geometry if Topology does not include bond information. May cause some Actions to fail.

[bondsearch <offset>] Optional; when searching for bonds via geometry search (default for Topologies without bond information) add <offset> to distances (default 0.2 Å). Increase this if your system includes unusually long bonds.

[searchtype {grid|pairlist}] Change search algorithm from the default search between residues algorithm:

grid Uses a grid when searching for bonds between residues. This can find bonds between residues that are not sequential (e.g. disulfide bonds).

pairlist Uses a pair list to search for bonds between atoms. This can potentially find bonds across periodic boundaries, but is the more experimental of the two.

Read in parameter file. The file format will be auto-detected. Current formats recognized by cpptraj are listed on page 49. If the file does not contain bond information, cpptraj will attempt to assign bonds based on a simple distance search of atoms within and between residues. The distance cutoff for determining bonds between atoms depends on the elements of the two atoms in question, augmented by <offset>. Molecule information is then determined from bond information.

9.10.1 *PDB format:*

[pqr] [readbox] [noconect]

[pqr]: Read charge and radius information from the occupancy and B-factor columns.

[readbox]: Read unit cell information from CRYST1 record if present.

[noconect] Do not read in CONECT records from PDB file.

IMPORTANT NOTES FOR PDB FILES

In some older format PDB files (before version 3), certain atoms may contain the asterisk ('*') character in their name (e.g. 'C1*' in a nucleic acid backbone). Since in *cpptraj* the asterisk is a reserved character for atom masks all asterisks in PDB atom names are replaced with single quote (') to avoid issues with the mask parser. So in a structure with an atom named C1*, to select it use the mask "@C1'".

Sometimes PDB files can contain alternate coordinates for the same atom in a residue, e.g.:

```
ATOM    806  CA ACYS A 105       6.460 -34.012 -21.801  0.49 32.23
```

```
ATOM    807  CB ACYS A 105       6.054 -33.502 -20.415  0.49 35.28
ATOM    808  CA BCYS A 105       6.468 -34.015 -21.815  0.51 32.42
ATOM    809  CB BCYS A 105       6.025 -33.499 -20.452  0.51 35.38
```

If this is the case *cpptraj* will print a warning about duplicate atom names but will take no other action. Both residues are considered 'CYS' and the mask ':CYS@CA' would select both atom 806 and 809. Residue insertion codes are read in but also not used by the mask parser.

9.11 parmbox

```
parmbox [parm <name> | parmindex <#> | <#>] [nobox] [truncoct]
     [x <xval>] [y <yval>] [z <zval>] [alpha <a>] [beta <b>] [gamma <g>]
```

[parm <name> | parmindex <#> | <#>] Name/tag or index of topology to modify. Default is first loaded topology.

[nobox] Remove box information.

[truncoct] Set truncated octahedon angles with lengths equal to <xval>.

[x <xval>] Box X length.

[y <yval>] Box Y length.

[z <zval>] Box Z length.

[alpha <a>] Box alpha angle.

[beta <b>] Box beta angle.

[gamma <g>] Box gamma angle.

Modify the box information for specified topology. Overwrites any box information if present with specified values; any that are not specified will remain unchanged. Note that unlike the ***box*** action this command affect box information immediately. This can be useful for e.g. removing box information from a parm when stripping solvent:

```
parm mol.water.parm7
parmstrip :WAT
parmbox nobox
parmwrite out strip.mol.nobox.parm7
```

9.12 parminfo

```
parminfo [parm <name> | parmindex <#> | <#>] [<mask>]
```

Print a summary of information contained in the specified topology (first loaded topology by default) .

9.13 parmstrip

```
parmstrip <mask> [parm <name> | parmindex <#> | <#>]
```

Strip atoms in <**mask**> from specified topology (by default the first topology loaded). Note that unlike the ***strip*** Action, this permanently modifies the topology for as long as *cpptraj* is running, so this should not be used if the topology is being used to read or write a trajectory via ***trajin***/***trajout***. This command can be used to quickly created stripped Amber topology files. For example, to strip all residues name WAT from a topology and write a new topology:

```
parm mol.water.parm7
parmstrip :WAT
parmwrite out strip.mol.parm7
```

9.14 parmwrite

```
parmwrite out <filename> [{parm <name> | parmindex <#> | <#> | crdset <setname>}]
          [<fmt>] [nochamber]
```

<**filename**> File to write to.

[parm <**name**> | **parmindex** <#> | <#>**]** Topology to write out.

[crdset <**setname**>**]** Write topology from specified COORDS data set.

[<**fmt**>**]** Format keyword. If not specified the file name extension will be used. Default is Amber Topology.

[nochamber] (Amber topology only) Remove any CHAMBER information from the topology.

Write out specified topology (first topology loaded by default) to <**filename**> with format <**fmt**> (Amber topology if not specified). Note that the Amber topology format is the only fully supported format for topology writes.

9.15 resinfo

```
resinfo [parm <name> | parmindex <#> | <#>] [<mask>] [short]
```

Print residue information for atoms in <mask> for selected topology (first loaded topology by default) with format:

```
#Res  Name First  Last Natom #Orig #Mol
```

where `#Res` is the residue number, `Name` is the residue name, `First` and `Last` are the first and last atom numbers of the residue, `Natom` is the total number of atoms in the residue, `#Orig` is the original residue number (in PDB files), and #Mol is the molecule number.

If short is specified then residues will be printed out in a condensed format. Each residue name will be shortened to 1 character, and residues are printed out in groups of 10, 5 groups to a line, with each line beginning with a residue number, e.g.

```
> resinfo short 4
1        MGFLAGKKIL ITGLLSNKSI AYGIAKAMHR EGAELAFTYV GQFKDRVEKL
51       CAEFNPAAVL PCDVISDQEI KDLFVELGKV WDGLDAIVHS IAFAPRDQLE
```

If the 1 character name for a residue is unknown it will be shown as the first letter of the residue name in lower-case.

9.16 scaledihedralk

```
scaledihedralk [parm <name> | parmindex <#>] <scale factor> [<mask> [useall]]
```

Scale dihedral force constants for dihderals selected by <mask> for specified topology. If **useall** is specified all atoms in <mask> must be present to select a dihedral, otherwise any atom in <mask> will selected a dihedral.

9.17 solvent

```
solvent [parm <name> | parmindex <#> | <#>] { <mask> | none }
```

Set solvent for selected topology (first loaded topology by default) based on <**mask**>, or set nothing as solvent if **none** is specified.

10 Trajectory File Commands

These commands control the reading and writing of trajectory files. There are three trajectory types in *cpptraj*: input, output, and reference. In *cpptraj*, trajectories are always associated with a topology file. If a topology file is not specified, a trajectory file will be associated with the first topology file loaded by default (this is true for both input and output trajectories.

Cpptraj currently understands the following trajectory file formats:

Format	Keyword(s)	Extension	Notes
Amber Trajectory	crd	.crd	Default format if keywords/extensions no
Amber NetCDF	cdf, netcdf	.nc	No compression.
Amber Restart	restart	.rst7, .rst	
Amber NetCDF Restart	ncrestart, restartnc	.ncrst	
Charmm "DCD" Trajectory	dcd, charmm	.dcd	
Charmm COORdinates	cor	.cor	
Charmm Restart	charmmres	.res	Read Only
PDB	pdb	.pdb	
Mol2	mol2	.mol2	
Scripps Binpos	binpos	.binpos	
Gromacs TRR	trr	.trr	
Gromacs GRO	gro	.gro	Read Only
Gromacs XTC	xtc	.xtc	
CIF	cif	.cif	Read Only
Tinker ARC	arc	.arc	Read Only
SQM Input	sqm	.sqm	Write Only
SDF	sdf	.sdf	Read Only
XYZ	xyz	.xyz	
LMOD Conflib	conflib	.conflib	Read Only, Detection by extens

The following trajectory-related commands are available:

Command	Description
ensemble	Set up a trajectory ensemble for reading during a run.
ensemblesize	(MPI only) specify number of members expected in subsequent ***ensemble*** commands.
reference	Read in a reference structure.
trajin	Set up a trajectory for reading during a Run.
trajout	Set up an output trajectory or ensemble for writing during a Run.

10.1 ensemble

```
ensemble <file0> {[<start>] [<stop> | last] [offset]} | lastframe
         [parm <parmfile / tag> | parmindex <#>]
         [trajnames <file1>,<file2>,...,<fileN>
         [remlog <remlogfile> [nstlim <nstlim> ntwx <ntwx>]]
```

<file0> Lowest replica filename.

[<start>] Frame to begin reading ensemble at (default 1).

[<stop> | last] Frame to stop reading ensemble at; if not specified or 'last' specified, end of trajectories.

[<offset>] Offset for reading in trajectory frames (default 1).

[lastframe] Select only the final frame of the trajectories.

[parm <parmfile>] Topology filename/tag to associate with trajectories (default first topology).

[parmindex <#>] Index of Topology to associate with trajectories (default 0, first topology).

[trajnames <file1>,...,<fileN>] Do not automatically search for additional replica trajectories; use comma-separated list of trajectory names.

[remlog <remlogfile>] For H-REMD trajectories only, use specified REMD log file to sort trajectories by coordinate index (instead of by Hamiltonian).

[nstlim <nstlim> ntwx <ntwx>] If trajectory and REMD log were not written at the same rate, these are the values for nstlim (steps between each exchange) and ntwx (steps between trajectory write) used in the REMD simulation.

Read in and process trajectories as an ensemble. Similar to '***trajin*** remdtraj', except instead of processing one frame at a target temperature, process all frames. This means that action and trajout commands apply to the entire ensemble; note however that not all actions currently function in ***'ensemble'*** mode. For example, to read in a replica ensemble, convert it to temperature trajectories, and calculate a distance at each temperature:

```
parm ala2.99sb.mbondi2.parm7
ensemble rem.crd.000 trajnames rem.crd.001,rem.crd.002,rem.crd.003
trajout temp.crd
distance d1 out d1.ensemble.dat @1 @21
```

This will output 4 temperature trajectories named 'temp.crd.X', where X ranges from 0 to 3 with 0 corresponding to the lowest temperature, and 'd1.ensemble.dat' containing 4 columns, each corresponding to a temperature. If run with MPI, data will be written to separate files named 'd1.ensemble.dat.X', similar to the output trajectories.

Note that in parallel (i.e. MPI) users should specify the ***ensemblesize*** command prior to ***ensemble*** in order to improve set up efficiency.

10.2 ensemblesize

```
ensemblesize <#>
```

This command is MPI only. It is used to set the expected number of members in any subsequent ***ensemble*** command, which dramatically improves set up efficiency.

10.3 reference

```
reference <name> [<frame#>] [<mask>] ([tag]) [lastframe] [crdset]
          [parm <parmfile / tag> | parmindex <#>]
```

<name> File name (or COORDS set name if 'crdset' specified) to read in as reference; any trajectory recognized by 'trajin' can be used.

[<frame#>] Frame number to use (default 1).

[<mask>] Only load atoms corresponding to <mask> from reference.

([tag]) Tag to give this reference file, e.g. "[MyRef]"; BRACKETS MUST BE INCLUDED.

[lastframe] Use last frame of reference.

[crdset] Use for COORDS data set named <name> instead of file.

[parm <parmfile/tag>] Topology filename/tag to associate with reference (default first topology).

[parmindex <#>] Index of Topology to associate with reference (default 0, first topology).

Use specified trajectory as reference coordinates. For trajectories with multiple frames, the first frame is used if a specific frame is not specified. An optional tag can be given (bounded in brackets) which can then be used in place of the name (see 3.4 on page 16 for examples of how to use tags). If desired, an atom mask can be used to read in only specified atoms from a reference.

Reference coordinates are now considered COORDS data sets and can be used anywhere a COORDS data set could, which allows reference structures to be manipulated once they are loaded. For example, a reference structure could be centered on the origin like so:

```
reference tz2.rst7 [MyRef]
crdaction [MyRef] center origin
```

Note that the 'average' keyword has been deprecated for reference. If desired, an averaged reference COORDS data set can be created from a trajectory using the 'average' command like so:

```
parm myparm.parm7
trajin mytraj.nc
rms first :1-12
average crdset RefAvg
run
rms ToAvg reference :1-12 out ToAvg.dat
```

10.4 trajin

```
trajin <filename> {[<start> [<stop> | last] [<offset>]]} | lastframe
       [parm <parmfile / tag> | parmindex <#>]
       [mdvel <velocities>] [mdfrc <forces>]
       [as <format keyword>] [ <Format Options> ]
       [ remdtraj {remdtrajtemp <Temperature> | remdtrajidx <idx1,idx2,...>
                   | remdtrajvalues <value1,value2,...>}
                  [trajnames <file1>,<file2>,...,<fileN>] ]
```

<filename> Trajectory file to read in.

[<start>] Frame to begin reading at (default 1). If a negative value is given it means "<start> frames before <stop>".

[<stop> | last] Frame to stop reading at; if not specified or 'last' specified, end of trajectory.

[<offset>] Offset for reading in trajectory frames (default 1).

[lastframe] Select only the final frame of the trajectory.

[parm <parmfile/tag>] Topology filename/tag to associate with trajectory (default first topology).

[parmindex <#>] Index of Topology to associate with trajectory (default 0, first topology).

[mdvel <velocities>] Use velocities from specified file.

[mdfrc <forces>] Use forces from specified file.

[as <format keyword>] Force file to be read as specified format; overrides file autodetection.

[<Format Options>] See below.

[remdtraj] Read <filename> as the first replica in a group of replica trajectories.

remdtrajtemp <Temperature> | remdtrajidx <idx1,idx2,...> Use frames at <Temperature> (for temperature replica trajectories) or index <idx1,idx2,...> (for Hamiltonian replica trajectories); For Multidimensional REMD simulations, multiple values are comma-separated.

remdtrajvalues <value1,value2,...> Use frames at <value1,value2,...> (for Multidimensional REMD trajectories). Each value may correspond to either temperature, pH, Redox Potential or Hamiltonian index. The values need to be

entered in the same order as the dimensions in the Multidimensional REMD simulation. For example, for T,pH-REMD value1 would correspond to a temperature and value2 to a pH. In the command, the values are comma-separated.

[trajnames <file1>,...,<fileN>] Do not automatically search for additional replica trajectories; use comma-separated list of trajectory names.

Read in trajectory specified by filename. See page 58 for currently recognized trajectory file formats. If just the <start> argument is given, all frames from <start> to the last frame of the trajectory will be read. To read in a trajectory with offsets where the last frame # is not known, specify the **last** keyword instead of a <stop> argument, e.g.

```
trajin Test1.crd 10 last 2
```

This will process Test1.crd from frame 10 to the last frame, skipping by 2 frames. To explicitly select only the last frame, specify the **lastframe** keyword:

```
trajin Test1.crd lastframe
```

Here is an example of loading in multiple trajectories which have difference topology files:

```
parm top0.parm7
parm top1.parm7
parm top2.parm7 [top2]
parm top3.parm7
trajin Test0.crd
trajin Test1.crd parm top1.parm7
trajin Test2.crd parm [top2]
trajin Test3.crd parmindex 3
```

Test0.crd is associated with top0.parm7; since no parm was specified it defaulted to the first parm read in. Test1.crd was associated with top1.parm7 by filename, Test2.crd was associated with top2.parm7 by its tag, and finally Test3.crd was associated with top3.parm7 by its index (based on the order it was read in).

Replica Trajectory Processing

If the **remdtraj** keyword is specified the trajectory is treated as belonging to the lowest # replica of a group of REMD trajectories. The remaining replicas can be either automatically detected by following a naming convention of <REMDFILENAME>.X, where X is the replica number, or explicitly specified in a comma-separated list following the **trajnames** keyword. All trajectories will be processed at the same time, but only frames with a temperature matching the one specified by **remdtrajtemp** or **remdtrajidx** will be processed.

For example, to process replica trajectories rem.001, rem.002, rem.003, and rem.004, grabbing only the frames at temperature 300.0 (assuming that this is a temperature in the ensemble):

```
trajin rem.001 remdtraj remdtrajtemp 300
```

or

```
trajin rem.001 remdtraj remdtrajtemp 300 trajnames rem.002,rem.003,rem.004
```

Note that the **remdout** keyword is deprecated. For this functionality see the **ensemble** keyword.

10.4.1 *Options for Amber NetCDF, Amber NC Restart, Amber Restart:*

```
[usevelascoords] [usefrcascoords]
```

usevelascoords Read in velocities in place of coordinates if present.

usefrcascoords Read in forces in place of coordinates if present.

10.4.2 *Options for CHARMM DCD:*

```
[{ucell | shape}]
```

ucell Force reading of box information as unit cell (for e.g. NAMD DCD trajectories).

shape Force reading of box information as shape matrix.

10.5 trajout

```
trajout <filename> [<format>] [append] [nobox] [novelocity]
        [notemperature] [notime] [noforce] [noreplicadim]
        [parm <parmfile> | parmindex <#>] [onlyframes <range>] [title <title>]
        [onlymembers <memberlist>]
        [start <start>] [stop <stop>] [offset <offset>]
        [ <Format Options> ]
```

<filename> Trajectory file to write to.

[<format>] Keyword specifying output format (see Table on page 58). If not specified format will be determined from extension, otherwise default to Amber trajectory.

[append] If <filename> exists, frames will be appended to <filename>.

[nobox] Do not write box coordinates to trajectory.

[novelocity] Do not write velocities to trajectory.

[notemperature] Do not write temperature to trajectory.

[notime] Do not write time to trajectory.

[noreplicadim] Do not write replica dimensions to trajectory.

[parm <parmfile>] Topology filename/tag to associate with trajectory (default first topology).

[parmindex <#>] Index of Topology to associate with trajectory (default 0, first topology).

[onlyframes <range>] Write only the specified input frames to <filename>.

[title <title>] Output trajectory title.

[onlymembers <memberlist>] Ensemble processing only; only write from specified members (starting from 0).

[start <start>] Begin output at frame <start> (1 by default).

[stop <stop>] End output at frame <stop> (last frame by default).

[offset <offset>] Skip <offset> frames between each output (1 by default).

During a run, write frames to trajectory specified by filename in specified file format (Amber trajectory if none specified) after all Action processing has occurred. To write out trajectories within the Action queue see the outtraj Action (11.54 on page 133). See page 58 for currently recognized output trajectory formats and their associated keyword(s). Note that now the file type can be determined from the output extension if not specified by a keyword. Multiple output trajectories of any format can be specified.

Frames will be written to the output trajectory when the parameter file being processed matches the parameter file the output trajectory was set up with. So given the input:

```
parm top0.parm7
parm top1.parm7 [top1]
trajin input0.crd
trajin input1.crd parm [top1]
trajout output.crd parm [top1]
```

only frames read in from input1.crd (which is associated with top1.parm7) will be written to output.crd. The trajectory input0.crd is associated with top0.parm7; since no output trajectory is associated with top0.parm7 no frames will be written when processing top0.parm7/input0.crd.

If **onlyframes** is specified, only input frames matching the specified range will be written out. For example, given the input:

```
trajin input.crd 1 10
trajout output.crd onlyframes 2,5-7
```

only frames 2, 5, 6, and 7 from input.crd will be written to output.crd.

10.5.1 *Options for pdb format:*

```
[dumpq | parse | vdw] [pdbres] [pdbatom]
[pdbv3] [teradvance] [terbyres | pdbter | noter]
[model | multi] [chainid <ID>] [sg <group>]
[include_ep] [conect] [keepext] [usecol21]
[bfacdata <set>] [occdata <set>] [bfacbyres] [occbyres]
[bfacscale] [occscale] [bfacmax <max>] [occmax <max>]
```

dumpq PQR format; write charges (in units of e-) and GB radii to occupancy and B-factor columns respectively.

parse PQR format; write charges and PARSE radii to occupancy/B-factor columns.

vdw PQR format; write charges and vdW radii to occupancy/B-factor columns.

pdbres: Use PDB V3 residue names.

pdbatom: Use PDB V3 atom names.

pdbv3: Use PDB V3 residue/atom names.

teradvance: Increment record (atom) number for TER records (not done by default).

terbyres: Print TER cards based on residue sequence instead of molecules.

pdbter: Print TER cards according to original PDB TER (if available).

noter: Do not write TER cards.

model (Default) Frames will be written to a single PDB file separated by MODEL/ENDMDL keywords.

multi Each frame will be written to a separate file with the frame # appended to <filename>.

chainid <ID> Write PDB file with chain ID <ID>.

sg <group> Space group for CRYST1 record; only used if box coordinates written.

include_ep Include extra points.

conect Write CONECT records for all bonds.

keepext Keep filename extension; write '<name>.<num>.<ext>' instead (implies 'multi').

usecol21 Use column 21 for 4-letter residue names.

bfacdata <set> Use data in <set> for B-factor column.

occdata <set> Use data in <set> for occupancy column.

bfacbyres If specified assume X values in B-factor data set are residue numbers.

occbyres If specified assume X values in occupancy data set are residue numbers.

bfacscale If specified scale values in B-factor column between 0 and <bfacmax>.

occscale If specified scale values in occupancy column between 0 and <occmax>.

bfacmax <max> Max value for bfacscale.

occmax <max> Max value for occscale.

10.5.2 *Options for Amber ASCII format:*

[remdtraj] [highprecision] [mdvel|mdfrc]

remdtraj Write REMD header to trajectory that includes temperature: 'REMD <Replica> <Step> <Total_Steps> <Temperature>'. Since *cpptraj* has no concept of replica number, 0 is printed for <Replica>. <Step> and <Total_Steps> are set to the current frame #.

highprecision: (EXPERT USE ONLY) Write with 8.6 precision instead of 8.3. Note that since the width does not change, the precision of large coords may be lower than 6.

mdvel Write velocities instead of coordinates.

mdfrc Write forces instead of coordinates.

10.5.3 *Options for Amber NetCDF format:*

[remdtraj] [mdvel] [mdfrc] [mdcrd]

remdtraj Write replica temperature to trajectory.

mdvel Write only velocity information in trajectory.

mdfrc Write only force information in trajectory.

mdcrd Write coordinates to trajectory (only required with mdvel/mdfrc).

10.5.4 *Options for Amber Restart/NetCDF Restart format:*

`[remdtraj] [novelocity] [notime] [time0 <initial time>] [dt <timestep>] [keepext]`

remdtraj Write replica temperature to restart. Note that this will automatically include time in the restart file (see the time0 keyword).

time0 <initial time> Time for first frame (default 1.0).

dt <timestep> Time step between frames (default 1.0). Time is calculated as t=(time0+frame)*dt.

keepext Keep filename extension; write '<name>.<num>.<ext>' instead.

10.5.5 *Options for CHARMM COORdinates:*

`[keepext] [ext] [segid <segid>] [segmask <mask> <segid> ...]`

keepext Keep filename extension; write '<name>.<num>.<ext>'

ext Use 'extended' format (default when > 99999 atoms).

segid <segid> Use <segid> as segment ID for all atoms.

segmask <mask> <segid> Use <segid> as segment ID for atoms selected by <mask>. Can be specified more than once.

10.5.6 *Options for CHARMM DCD:*

`[x64] [ucell] [veltraj]`

x64 Use 8 byte block size (default 4 bytes).

ucell Write older (v21) format trajectory that stores unit cell params instead of shape matrix.

veltraj Write velocity trajectory instead of coordinates.

10.5.7 *Options for GROMACS TRX/XTC format:*

`[dt <time step>]`

dt Time step tp multiply set numbers by (default 1.0). Ignored if time already present.

10.5.8 *Options for mol2 format:*

`[single | multi] [sybyltype] [sybylatom <file>] [sybylbond <file>] [keepext]`

single (Default) Frames will be written to a single Mol2 file separated by MOLECULE keywords.

multi Each frame will be written to a separate file with the frame # appended to <filename>.

sybyltype Convert Amber atom types (if present) to SYBYL types. Requires $AMBERHOME is set.

sybylatom File containing Amber to SYBYL atom type correspondance (optional).

sybylbond File containing Amber to SYBYL bond type correspondance (optional).

keepext Keep filename extension; write '<name>.<num>.<ext>' instead (implies 'multi').

10.5.9 Options for SQM input format:

`[charge <c>]`

charge <**c**> Set total integer charge. If not specified it will be calculated from atomic charges.

10.5.10 Options for XYZ format:

`ftype {atomxyz|xyz} titletype {none|single|perframe} width <#> prec <#>`

ftype {atomxyz|xyz} Choose either 'ATOM X Y Z' (default) or 'X Y Z' output format.

titletype {none|single|perframe} No title, one title (default), or title before every frame.

width <#> Output format width.

prec <#> Output format precision.

11 Action Commands

Actions in *cpptraj* operate on frames read in by the ***trajin*** or ***ensemble*** commands one at a time and extract derived data, modify the coordinates/topology in some way, or both. Most Actions in *cpptraj* function exactly the way they do in *ptraj* and are backwards-compatible. Some Action commands in *cpptraj* have extra functionality compared to *ptraj* (such as the per-residue RMSD function

of the ***rmsd*** Action, or the ability to write out stripped topologies for visualization in the ***strip*** Action), while other Actions produce slightly different output (like the ***hbond***/***secstruct*** Actions).

Unlike some other command types, when an Action command is issued it is by default added to the Action queue and is not executed until trajectory processing is started (e.g. by a ***run*** or ***go*** command). However, Actions can be executed immediately on COORDS data sets via the ***crdaction*** command (7.3 on page 32).

When a frame is modified by an Action, it is modified for every Action that follows them during trajectory processing. For example, given a solvated system with water residues named WAT and the following Action commands:

```
rmsd R1 first :WAT out water-rmsd.dat
strip :WAT
rmsd R2 first :WAT out water-rmsd-2.dat
```

the first ***rms*** command will be valid, but the second ***rms*** command will not since all residues named WAT are removed from the state by the ***strip*** command.

Note that for commands which can use a reference mask as well as a target mask (e.g. ***rms***, ***drmsd***, ***symmrmsd***, etc.) there must be a 1 to 1 correspondence between the atoms in each mask, i.e. the *number of atoms and the ordering of selected atoms must be the same.*

The following Actions are available. If an Action may modify coordinate/topology information for subsequent Actions it is denoted with an X in the `Mod` column.

Command	Description	Mod
align	Align structure to a reference.	X
angle	Calculate the angle between three points.	
areapermol	Calculate area per molecule for molecules in a specified plane.	
atomiccorr	Calculate average correlation between motions of specified atoms.	
atomicfluct, rmsf	Calculate root mean square fluctuation of specified atoms/residues.	
atommap	Attempt to create a map between atoms in molecules with different atom ordering.	X
autoimage	Automatically re-image coordinates.	X
average	Calculate average structure.	
bounds	Calculate the min/max coordinates for specified atoms. Can be used to create grid data sets.	
box	Set or overwrite box information for frames.	
center	Center specified coordinates to box center or onto reference structure.	X
check, checkoverlap, checkstructure	Check for bad atomic overlaps or bond lengths. Can be used to skip corrupted frames.	
checkchirality	Report chirality around alpha carbons in amino acids (L, D).	
closest, closestwaters	Retain only the specified number of solvent molecules closest to specified solute.	X

clusterdihedral	Assign frames into clusters based on binning of backbone dihedral angles in amino acids.	
contacts	Older version of ***nativecontacts***, retained for backwards compatibility.	
createcrd	Create a COORDS data set from input frames.	
createreservoir	Create a structure reservoir for use with reservoir REMD simulations.	
density	Calculate density along a coordinate.	
diffusion	Calculate translational diffusion of molecules.	
dihedral	Calculate the dihedral angle using four points.	
dihrms dihedralrms	Calculate the RMSD of dihedrals to dihedrals in a reference structure.	
dipole	Bin dipoles of solvent molecules in 3D grid. Not well tested, may be obsolete.	
distance	Calculate the distance between two points.	
drms, drmsd	Calculate the RMSD of distance pairs within selected atoms.	
dssp, secstruct	Calculate secondary structure content using the DSSP algorithm	
energy	Calculate simple bond, angle, dihedral, and non-bonded energy terms (no PME).	
esander	Calculate energies using via SANDER; requires compilation with the SANDER API.	
filter	Filter frames for subsequent Actions using data sets and user defined criteria.	
fixatomorder	Fix atom ordering so that all atoms in molecules are sequential.	X
fiximagedbonds	Fix bonds which have been split across periodic boundaries by imaging.	
gist	Perform grid inhomogenous solvation theory.	
grid	Bin selected atoms on a 3D grid.	
hbond	Calculate hydrogen bonds using geometric criteria.	
image	Re-image coordinates. The ***autoimage*** command typically provides better results.	X
jcoupling	Calculate J-coupling values from specified dihedral angles.	
lessplit	Split/average frames from LES trajectories.	
lie	Calculate linear interaction energy between user-specified ligand and surroundings.	
lipidorder	Calculate order parameters for lipids in planar membranes.	
lipidscd	Calculate lipid order parameters SCD (|<P2>|) for lipid chains. Automatically identifies lipids.	
makestructure	Modify structure by applying dihedral values to specified residues.	X
mask	Print the results of selection by specified atom mask. Good for distnace-based masks.	
matrix	Calculate a matrix of the specified type from input coordinates.	
minimage	Calculate minimum non-self imaged distance between atoms in specified masks.	
molsurf	Calculate Connolly surface area of specified atoms. Cannot do partial surface areas.	

multidihedral	Calculate multiple dihedral angles of specified/given types.	
multivector	Calculate multiple vectors between specified atoms.	
nastruct	Perform nucelic acid structure analysis.	
nativecontacts	Calculate native contacts within a region or between two regions using a given reference. Can also be used to get min/max distances between groups of atoms.	
outtraj	Write frames to a trajectory file within a list of Actions.	
pairdist	Calculate pair distribution function.	
pairwise	Calculate pair-wise non-bonded energies.	
principal	Calculate and optionally align system along principal axes.	X
projection	Project coordinates along given eigenvectors.	
pucker	Calculate ring pucker using five or six points.	
radgyr, rog	Calculate radius of gyration (and optionally tensor) for specified atoms.	
radial, rdf	Calculate radial distribution function.	
randomizeions	Swap specified ions with randomly selected solvent molecules.	X
replicatecell	Replicate unit cell in specified (or all) directions for specfied atoms and write to trajectory.	
rms, rmsd	Perform best fit of coordinates to reference and calculate coordinate RMSD. Fitting can be disabled.	X
rotate	Rotate the system around X/Y/Z axes, a specified axis, or via given rotation matrices.	X
runavg, runningaverage	Calculate the running average of coordinates over specified window size.	X
scale	Scale coordinates in X/Y/Z directions by specified factors.	X
setvelocity	Set velocities for specified atoms using Maxwellian distribution based on given temperature.	
spam	SPAM method for estimating relative free energies of waters in hydration shell around proteins.	X
stfcdiffusion	Alternative translational diffusion calculation which can calculate diffusion in specified regions.	
strip	Remove specified atoms from the system.	
surf	Calculate the LCPO surface area of specified atoms. Can do partial surface areas.	
symmrmsd	Calculate symmetry-corrected RMSD.	X
temperature	Calculate system temperature using velocities of specified atoms.	
time	Add/remove/modify time information in frames.	X
trans, translate	Translate specified atoms by specified amounts in X/Y/Z directions.	X
unstrip	Undo all previous ***strip*** Action commands.	
unwrap	Reverse of ***image***; unwrap selected atoms so they have continuous trajectories.	X
vector	Calculate various types of vector quantities.	
velocityautocorr	Calculate velocity autocorrelation function.	
volmap	Create volumetric map for specified coordinates; similar to ***grid*** but takes	

	into account atomic radii. Similar to VMD ***volmap***.	
volume	Calculate unit cell volume.	
watershell	Calculate the number of waters in the first and second solvation shells based on distance critera.	
xtalsymm	Re-image coordinates based on crystal space group symmetry operations and asymmetric unit volume.	X

11.1 align

`align <mask> [<refmask>] [move <mask>] [mass]`
`[ first | reference | ref <name> | refindex <#> | previous |`
`reftraj <name> [parm <name> | parmindex <#>] ]`

<**mask**> `Target atoms to fit.`

[<**refmask**>] `Reference atoms to fit (default is target mask).`

[**move** <**mask**>] `Atoms to move when aligning (default is target mask).`

[**mass**] `Mass-weight the fit.`

`Reference keywords:`

first `Use the first trajectory frame processed as reference.`

reference `Use the first previously read in reference structure (refindex 0).`

ref <**name**> `Use previously read in reference structure specified by filename/tag.`

refindex <#> `Use previously read in reference structure specified by <#> (based on order read in).`

previous `Use frame prior to current frame as reference.`

reftraj <**name**> `Use frames from COORDS set <name> or read in from trajectory file <name> as references. Each frame from <name> is used in turn, so that frame 1 is compared to frame 1 from <name>, frame 2 is compared to frame 2 from <name> and so on. If <trajname> runs out of frames before processing is complete, the last frame of <trajname> continues to be used as the reference.`

parm <**parmname**> | **parmindex** <#> `If reftraj specifies a trajectory file, associate it with specified topology; if not specified the first topology is used.`

Align structure using specified <mask> onto reference. If 'move' is specified, only move atoms in the move mask.

11.2 angle

```
angle [<dataset name>] <mask1> <mask2> <mask3> [out <filename>] [mass]
```

[<dataset name>] Output data set name.

<maskX> Three atom masks selecting atom(s) to calculate angle for.

[out <filename>] Output file name.

[mass] Use center of mass of atoms in <maskX> instead of geometric center.

Calculate angle (in degrees) between atoms in <mask1>, <mask2>, and <mask3>. For example, to calculate the angle between the first three atoms in the system:

```
angle A123 @1 @2 @3 out A123.agr
```

11.3 areapermol

```
areapermol [<name>] {[<mask1>] [nlayers <#>] | nmols <#>} [out <filename>] [{xy | xz |
```

[<name>] Data set name.

[<mask1>] Atom mask for selecting molecules. If any atom in a molecule is selected the whole molecule is selected.

[nlayers <#>] Number of layers of molecules. Total number of molecules used will be # molecules divided by # layers.

[nmols <#>] If <mask1> is not specified, the number of molecules to use when calculating area per molecule.

[out <filename>] Output file name.

[{xy|xz|yz}] Cross-section of box to calculate area of. Default is X-Y.

Calculate area per molecule as Area / # molecules. The area is determined from the specified cross-section of the box (X-Y by default). Currently the calculation is only guaranteed to work properly with orthorhombic unit cells. For example, to get the area per molecule of residues named "OL" which are arranged in 2 layers:

```
areapermol OL_area :OL nlayers 2 out apm.dat
```

11.4 atomiccorr

```
atomiccorr [<mask>] out <filename> [cut <cutoff>] [min <min spacing>]
           [byatom | byres]
```

<mask> Atoms to calculate motion vectors for.

out <filename> File to write results to.

cut <cutoff> Only print correlations with absolute value greater than <cutoff>.

min <min spacing> Only calculate correlations for motion vectors spaced <min spacing> apart.

byatom Default; calculate atomic motion vectors.

byres Calculate motion vectors for entire residues (selected atoms in residues only).

Calculate average correlations between the motion of atoms in <mask>. For each frame, a motion vector is calculated for each selected atom from its previous position to its current position. For each pair of motion vectors *Va* and *Vb*, the average correlation between those vectors is calculated as the average of the dot product of those vectors over all N frames.

$$AvgCorr(a,b) = \frac{\sum V_a(i) \cdot V_b(i)}{N}$$

The value of AvgCorr can range from 1.0 (correlated) to 0.0 (no correlation) to -1.0 (anti-correlated). For example, to calculate the correlation of motion vectors between residues 1 to 13, writing to a Gnuplot-readable formatted file:

```
atomiccorr :1-13 out acorr.gnu byres
```

11.5 atomicfluct | rmsf

```
atomicfluct [out <filename>] [<mask>] [byres | byatom | bymask] [bfactor]
            [calcadp [adpout <file>]]
            [start <start>] [stop <stop>] [offset <offset>]
```

out <filename> Write data to file named <filename>

[<mask>] Calculate fluctuations for atoms in <mask> (all if not specified).

byres Output the average (mass-weighted) fluctuation by residue.

bymask Output the average (mass-weighted) fluctuation for all atoms in <mask>.

byatom (default) Output the fluctuation by atom.

[bfactor] Calculate atomic positional fluctuations squared and weight by $\frac{8}{3}\pi^2$; this is similar but not necessarily equivalent to the calculation of crystallographic B-factors.

[calcadp [adpout <file>]] Calculate anisotropic displacement parameters and optionally output them to <file>.

[<start>] Frame to begin calculation at (default 1).

[<stop>] Frame to end calculation at (default last).

[<offset>] Frames to skip between calculations (default 1).

Compute the atomic positional fluctuations (also referred to as root-mean-square fluctuations, RMSF) for atoms specified in the <**mask**>. Note that RMS fitting is not done implicitly. If you want fluctuations without rotations or translations (for example to the average structure), perform an RMS fit to the average structure (best) or the first structure (see ***rmsd***) prior to this calculation. The units are (Å) for RMSF or $Å^2 \times \frac{8}{3}\pi^2$ if **bfactor** is specified.

If **byres** or **bymask** are specified, the mass-weighted average of atomic fluctuations of each atom for either each residue or the entire mask will be calculated respectively:

$$\langle Fluct \rangle = \frac{\sum AtomFluct_i * Mass_i}{\sum Mass_i}$$

If **calcadp** is specified, anisotropic displacement factors for atoms will be calculated and written to the file specified by **adpout** (or STDOUT if not specified) using PDB ANISOU record format. Note that **calcadp** automatically implies **bfactor**.

With *cpptraj* it is possible to perform coordinate averaging, the fit to average coordinates, and the atomic fluctuation calculation in a single execution like so:

```
parm myparm.parm7
trajin mytrajectory.crd
rms first
average crdset MyAvg
run
rms ref MyAvg
atomicfluct out fluct.agr
```

To write the mass-weighted B-factors for the protein backbone atoms C, CA, and N, averaged by residue use the command:

```
atomicfluct out back.agr @C,CA,N byres bfactor
```

To write the RMSF or atomic positional fluctuations of the same atoms, use the command:

```
atomicfluct out backbone-atoms.agr @C,CA,N
```

11.6 atommap

```
atommap <target> <reference> [mapout <filename>] [maponly]
        [rmsfit [ rmsout <rmsout> ]]
```

<target> Reference structure whose atoms will be remapped.

<reference> Reference structure that <target> should be mapped to.

mapout <filename> Write atom map to <filename> with format:
TargetAtomNumber TargetAtomName ReferenceAtomNumber ReferenceAtomName
Target atoms that cannot be mapped to a reference atom are denoted "--".

maponly Write atom map but do not reorder atoms.

rmsfit Any input frames using the same topology as <target> will be RMS fit to <reference> using whatever atoms could be mapped.

rmsout <rmsout> If rmsfit specified, write resulting RMSDs to <rmsout>.

Attempt to map the atoms of <target> to those of <reference> based on structural similarity. This is useful e.g. when there are two files containing the same structure but with different atom names or atom ordering. Both <target> and <reference> need to have been read in with a previous ***reference*** command. The state will then be modified so that any trajectory read in with the same parameter file as <target> will have its atoms mapped (i.e. reordered) to match those of <reference>. If the number of atoms that can be mapped in <target> are less than those in <reference>, the reference structure specified by <reference> will be modified to include only mapped atoms; this is useful if for example the reference structure is protonated with respect to the target. The **rmsfit** keyword is useful in cases where the atom mapping will not be complete (e.g. two ligands with the same scaffold but different substituents).

For example, say you have the same ligand structure in two files, Ref.mol2 and Lig.mol2, but the atom ordering in each file is different. To map the atoms in Lig.mol2 onto those of Ref.mol2 so that Lig.mol2 has the same ordering as Ref.mol2:

```
parm Lig.mol2
reference Lig.mol2
parm Ref.mol2
reference Ref.mol2 parmindex 1
atommap Lig.mol2 Ref.mol2 mapout atommap.dat
trajin Lig.mol2
trajout Lig.reordered.mol2 mol2
```

11.7 autoimage

```
autoimage [<mask> | anchor <mask> [fixed <mask>] [mobile <mask>]]
          [origin] [firstatom] [familiar | triclinic]
```

[<mask> | anchor <mask>] Atoms to image around; this is the region that will be centered. Default is the entire first molecule.

[fixed <mask>] Molecules that should remain 'fixed' to the anchor region; default is all non-ion/non-solvent molecules.

[mobile <mask>] Molecules that can be freely imaged; default is all ion/solvent molecules.

[origin] Center anchor region at the origin; if not specified, center at box center.

[firstatom] Image based on molecule first atom; default is to image by molecule center of mass.

[familiar] Image to familiar truncated-octahedral shape; this is on by default if the original cell is truncated octahedron.

[triclinic] Force general triclinic imaging.

Automatically center and image (by molecule) a trajectory with periodic boundaries. For most cases just specifying ***'autoimage'*** alone is sufficient. The atoms of the **'anchor'** region (default the entire first molecule) will be centered; all **'fixed'** molecules will be imaged only if imaging brings them closer to the **'anchor'** molecule (default for **'fixed'** molecules is all non-solvent non-ion molecules). All other molecules (referred to as **'mobile'**) will be imaged freely.

The autoimage command works for the majority of systems; however, for very densely packed systems the default anchor (entire first molecule) may not be appropriate. In these cases, it is recommended to choose as the anchor a small region which should lie near the center of your system. For example, in a protein dimer system one could choose a single residue that is near the center of the interface between the two monomers.

11.8 average

```
average {crdset <set name> | <filename>} [<mask>]
        [start <start>] [stop <stop>] [offset <offset>]
        [Trajout Args]
```

<filename> If specified, write averaged coordinates to <filename> (not compatible with crdset).

crdset <**set name**> If specified, save averaged coordinates to COORDS set <set name> (not compatible with <filename>).

[<**mask**>**]** Average coordinates in <mask> (all atoms if not specified).

[<**start**>**]** Frame to begin calculation at (default 1).

[<**stop**>**]** Frame to end calculation at (default last).

[<**offset**>**]** Frames to skip between calculations (default 1).

[Trajout args] Output trajectory format argument(s) (default Amber Trajectory).

Calculate the average of input coordinates and write out to file named <**filename**> or save to COORDS set named <**set name**> in any trajectory format *cpptraj* recognizes (Amber Trajectory if not specified). If the number of atoms in <**mask**> are less than the total number of atoms, the topology will be stripped to match <**mask**>.

Note that since coordinates are being averaged over many frames, resulting structures may appear distorted. For example, if one averages the coordinates of a freely rotating methyl group the average position of the hydrogen atoms will be close to the center of rotation. Also note that typically one will want to remove global rotational and translation movement prior to this command by using e.g. the ***rms*** (11.64 on page 142) command.

Any arguments that are valid for the ***trajout*** command (10.5 on page 63) can be passed to this command in order to control the format of the output coordinates. For example, to write out a PDB file containing the averaged coordinates over all frames:

```
average test.pdb pdb
```

To write out a mol2 file containing only the averaged coordinates of residues 1 to 10 for frames 1 to 100:

```
average test.mol2 mol2 start 1 stop 100 :1-10
```

To create an average structure of atoms named CA and then use it as a reference for an rms command in a subsequent run:

```
trajin Input.nc
average crdset MyAvg @CA
run
rms ref MyAvg @CA out RmsToAvg.dat
run
```

11.9 avgcoord

This command is deprecated. Use 'vector center' (optionally with keyword 'magnitude') instead.

11.10 bounds

```
bounds [<mask>] [out <filename>]
       [dx <dx> [dy <dy>] [dz <dz>] name <gridname> [offset <bin offset>]]
```

[<mask>] Mask of atoms to determine bounds of.

[out <filename>] File to write bounds to (default STDOUT if not specified).

[dx <dx> [dy <dy>] [dz <dz>]] Triggers creation of a grid data set from bounds. Spacings of generated grid in the X, Y and Z directions. If only dx is specified <dx> will be used for <dy> and <dz> as well.

[name <gridname>] Name of generated data sets.

[offset <bin offset>] Number of bins to add/subtract in each direction to generated grid.

DataSets Generated

<gridname> The 3D grid (only if 'dx' etc specified).

<gridname>[xmin] The minimum x coordinate encountered.

<gridname>[xmax] The maximum x coordinate encountered.

<gridname>[ymin] The minimum y coordinate encountered.

<gridname>[ymax] The maximum y coordinate encountered.

<gridname>[zmin] The minimum z coordinate encountered.

<gridname>[zmax] The maximum z coordinate encountered.

Calculate the boundaries (i.e. the max/min X/Y/Z coordinates) of atoms in **<mask>** and write to **<filename>** (STDOUT if not specified). Useful for determining dimensions for the ***grid*** command, and can be used to generate a grid data set that can be used by ***grid*** (see 11.36 on page 106).

11.11 box

```
box [x <xval>] [y <yval>] [z <zval>] [alpha <a>] [beta <b>] [gamma <g>]
    [nobox] [truncoct]
```

[x <xval>] [y <yval>] [z <zval>] Change box length(s) to specified value(s).

[alpha <a>] [beta <b>] [gamma <g>] Change box angle(s) to specified value(s).

[nobox] Remove any existing box information.

[truncoct] Set box angles to truncated octahedron.

Modify box information during trajectory processing. Note that this will permanently modify the box information for topology files during trajectory processing as well. It is possible to modify any number of the box parameters (e.g. only the Z length can be modified if desired while leaving all other parameters intact).

11.12 center

```
center [<mask>] [origin] [mass]
       [ reference | ref <name> | refindex <#> [<refmask>]]
```

[<mask>] Center based on atoms in mask; default is all atoms.

[origin] Center to origin (0, 0, 0); default is center to box center (X/2, Y/2, Z/2).

[mass] Use center of mass instead of geometric center.

[reference | ref <name> | refindex <#> [<refmask]] Center using coordinates in specified reference structure selected by <refmask> (<mask> if not specified.

Move all atoms so that the center of the atoms in **<mask>** is centered at the specified location: box center (default), coordinate origin, or reference coordinates.

For example, to move all coordinates so that the center of mass of residue 1 is at the center of the box:

```
center :1 mass
```

11.13 check | checkoverlap | checkstructure

```
check [<mask>] [around <mask2>] [reportfile <report>] [noimage] [skipbadframes]
      [offset <offset>] [cut <cut>] [nobondcheck] [silent]
```

[<mask>] Check structure of atoms in <mask> (all if not specified).

[around <mask2>] If specified, only check for problems between atoms in <mask> and atoms in <mask2>.

[reportfile <report>] Write any problems found to <report> (STDOUT if not specified).

[noimage] Do not image distances.

[skipbadframes] If errors are encountered for a frame, subsequent actions/trajectory output will be skipped.

[offset <offset>] Report bond lengths greater than the equilibrium value plus <offset> (default 1.0 Å)

[cut <cut>] Report atoms closer than <cut> (default 0.8 Å).

[nobondcheck] Check overlaps only.

[silent] Do not print information for bad frames - useful in conjunction with the skipbadframes option.

Check the structure and report problems related to atomic overlap/unusual bond length. Problems are reported when any two atoms in the mask are closer than <**cut**>. If bonds are being checked then bond lengths greater than their equilibrium value + <**offset**> are reported as well. This command can also be used to skip corrupted frames in a trajectory during processing. For example, if this message is encountered:

```
Warning: Frame 10 coords 1 & 2 overlap at origin; may be corrupt.
```

One could use ***check*** so that e.g. a subsequent ***distance*** command is not processed for bad frames:

```
check @1,2 skipbadframes silent
distance d1 :1 :10
```

Usually frame corruption can be detected using only a few atoms, but this may not catch all types of corruption. The more atoms that are used the better the corruption detection will be, but the slower it will be to process the command. Typically a good procedure to follow when corruption is suspected is to run ***check*** using all important atoms (e.g. all solute heavy atoms) with the **skipbadframes** keyword followed by a ***trajout*** command to write all non-corrupt frames, for example:

```
trajin corrupted.crd
check :1-13 skipbadframes silent
trajout fixed.corrupted.nc
```

11.14 checkchirality

`checkchirality [<name>] [<mask>] [out <filename>]`

[<name>] Data set name.

[<mask>] Atoms to check.

[out <filename>] File to write results to.

DataSet Aspects:

[L] Number of frames 'L' for each residue.

[D] Number of frames 'D' for each residue.

Check the chirality around the alpha carbon in amino acid residues selected by <mask>. Note that cpptraj expects atom names to correspond to the PDB V3 standard: N, CA, C, CB. For each residue, the number of frames in which the amino acid is 'L' or 'D' will be recorded. For example, to check the chirality of all amino acids in a system and write to a file named chiral.dat with data set name DPDP:

```
checkchirality DPDP out chiral.dat
```

Output will have format similar to:

```
#Res      DPDP[L]  DPDP[D]
   2.000      100        0
```

So in this example residue 2 was 'L' for 100 frames and 'D' for 0 frames.

11.15 closest | closestwaters

```
closest <# to keep> <mask> [solventmask <solvent mask>] [noimage]
     [first | oxygen] [center] [closestout <filename> [name <setname>]]
     [outprefix <parmprefix>] [parmout <file>]
```

<# to keep> Number of solvent molecules to keep around <mask>

<mask> Mask of atoms to search for closest waters around.

[solventmask <solvent mask>] Optional mask for selecting solvent atoms. If not specified, atoms in all molecules marked as "solvent" will be used.

[noimage] Do not perform imaging; only recommended if trajectory has previously been imaged.

[first | oxygen] Calculate distances between all atoms in <mask> and the first atom of solvent only (recommended for standard water models as it will increase speed of calculation).

[center] Search for waters closest to geometric center of <mask> instead of each atom in <mask>.

[closestout <filename>] Write information on the closest solvent molecules to <filename>.

[outprefix <prefix>] `Write corresponding topology to file with name prefix <prefix>.`

[parmout <file>] `Write corresponding topology to file with name <file>.`

`DataSet Aspects:`

[Frame] `Frame number.`

[Mol] `Original solvent molecule number.`

[Dist] `Solvent molecule distance in Å.`

[FirstAtm] `First atom number of original solvent molecule.`

Similar to the ***strip*** command, but modify coordinate frame and topology by keeping only the specified number of closest solvent molecules to the region specified by the given mask. Solvent molecules can be determined automatically by *cpptraj* (by default residues named WAT, HOH, or TIP3), can be specified prior via the ***solvent*** command (9.17 on page 57), or can be selected by **solventmask**.

The format of the **closestout** file is:

```
Frame    Molecule    Distance    FirstAtom#
```

For example, to obtain the 10 closest waters to residues 1-268 by distance to the first atom of the waters, write out which waters were closest for each frame to a file called "closestmols.dat", and write out the stripped topology with prefix "closest" containing only the solute and 10 waters:

```
closest 10 :1-268 first closestout closestmols.dat outprefix closest
```

As of version 17 this command is CUDA-enabled in CUDA versions of CPPTRAJ.

11.16 cluster

Although the **'cluster'** command can still be specified as an action, it is now considered an analysis. See 12.4 on page 169.

11.17 clusterdihedral

```
clusterdihedral [phibins <N>] [psibins <M>] [out <outfile>]
                [dihedralfile <dfile> | <mask>]
                [framefile <framefile>] [clusterinfo <infofile>]
                [clustervtime <cvtfile>] [cut <CUT>]
```

Cluster frames in a trajectory using dihedral angles. To define which dihedral angles will be used for clustering either an atom mask or an input file specified by the **dihedralfile** keyword should be used. If dihedral file is used, each line in the file should contain a dihedral to be binned with format:

```
ATOM#1 ATOM#2 ATOM#3 ATOM#4 #BINS
```

where the ATOM arguments are the atom numbers (starting from 1) defining the dihedral and #BINS is the number of bins to be used (so if #BINS=10 the width of each bin will be 36^{o}). If an atom mask is specified, only protein backbone dihedrals (Phi and Psi defined using atom names C-N-CA-C and N-CA-C-N) within the mask will be used, with the bin sizes specified by the phibins and psibins keywords (default for each is 10 bins).

Output will either be written to STDOUT or the file specified by the **out** keyword. First, information about which dihedrals were clustered will be printed. Then the number of clusters will be printed, followed by detailed information of each cluster. The clusters are sorted from most populated to least populated. Each cluster line has format

```
Cluster CLUSTERNUM CLUSTERPOP [ dihedral1bin, dihedral2bin ... dihedralNbin ]
```

followed by a list of frame numbers that belong to that cluster. If a cutoff is specified by **cut**, only clusters with population greater than CUT will be printed.

If specified by the **clustervtime** keyword, the number of clusters for each frame will be printed to <cvtfile>. If specified by the **framefile** keyword, a file containing cluster information for each frame will be written with format

```
Frame CLUSTERNUM CLUSTERSIZE DIHEDRALBINID
```

where DIHEDRALBINID is a number that identifies the unique combination of dihedral bins this cluster belongs to (specifically it is a 3*number-of-dihedral-characters long number composed of the individual dihedral bins).

If specified by the **clusterinfo** keyword, a file containing information on each dihedral and each cluster will be printed. This file can be read by SANDER for use with REMD with a structure reservoir (-rremd=3). The file, which is essentially a simplified version of the main output file, has the following format:

```
#DIHEDRALS
dihedral1_atom1 dihedral1_atom2 dihedral1_atom3 dihedral1_atom4
...
#CLUSTERS
CLUSTERNUM1 CLUSTERSIZE1 DIHEDRALBINID1
...
```

11.18 contacts

```
contacts [ first | reference | ref <ref> | refindex <#> ] [byresidue]
         [out <filename>] [time <interval>] [distance <cutoff>] [<mask>]
```

NOTE: Users are encouraged to try the ***nativecontacts*** command (on page 129), an update version of this command.

For each atom given in *mask*, calculate the number of other atoms (contacts) within the distance *cutoff*. The default cutoff is 7.0 A. Only atoms in *mask* are potential interaction partners (e.g., a mask @CA will evaluate only contacts between CA atoms). The results are dumped to *filename* if the keyword "`out`" is specified. Thereby, the time between snapshots is taken to be *interval*. In addition to the number of overall contacts, the number of native contacts is also determined. Native contacts are those that have been found either in the first snapshot of the trajectory (if the keyword "`first`" is specified) or in a reference structure (if the keyword "`reference`" is specified). Finally, if the keyword "`byresidue`" is provided, results are output on a per-residue basis for each snapshot, whereby the number of native contacts is written to *filename*.native.

11.19 createcrd

```
createcrd [<name>] [ parm <name> | parmindex <#> ]
```

This command creates a COORDS data set named <name> using trajectory frames that are associated with the specified topology.

For example, to save frames that have been previously RMS-fit to a reference structure into a COORDS set named MyCrd you would use the input:

```
rms reference :1-12@CA
createcrd MyCrd
strip :6-8
```

Note that here the ***strip*** command will have no effect on the coordinates saved in MyCrd since it occurs after the ***createcrd*** command.

11.20 createreservoir

```
createreservoir <filename> ene <energy data set> [bin <cluster bin data set>]
                temp0 <temp0> iseed <iseed> [velocity]
                [parm <parmfile> | parmindex <#>] [title <title>]
```

<filename> `File name of the reservoir to create.`

ene <energy data set> `Data set with energies corresponding to frames.`

[bin <cluster bin data set>] `Data set with bin numbers (for RREMD=3).`

temp0 <**temp0**> Reservoir temperature.

iseed <**iseed**> Reservoir random number seed.

[velocity] Include velocities in the reservoir.

[parm <**parmfile**> | **parmindex** <#>**]** Associated topology.

[title <**title**>**]** Reservoir title.

Create structure reservoir for use with reservoir REMD simulations using energies in <energy data set>, temperature <temp0> and random seed <iseed> Include velocities if [velocity] is specified. If <cluster bin data set> is specified from e.g. a previous 'clusterdihedral' command, the reservoir can be used for non-Boltzmann reservoir REMD (rremd==3).

11.21 density

```
density [out <filename>]
        [ <mask1> ... <maskN> [name <set name>] [delta <resolution>] [{x|y|z}]
          [{number|mass|charge|electron}] [{bincenter|binedge}] ]
```

[out <**filename**>**]** Output file for histogram (relative distances vs. densities for each mask) or total density.

[name <**set name**>**]** Output data set name.

<**mask1**> **...** <**maskN**> Arbitrary number of masks for atom selection; a dataset is created and the output will contain entries for each mask.

[delta <**resolution**>**]** Resolution, i.e. determines number of slices (i.e. histogram bins). (default 0.25 Å)

[{x|y|z}] Coordinate for density calculation. (default z)

[{number|mass|charge|electron}] Number, mass, partial charge (q) or electron (Ne - q) density. Electron density will be converted to e-/Å3 by dividing the average area spanned by the other two dimensions. (default number)

[{bincenter|binedge}] Determine whether histogram bin coordinates will be based on bin center (default) or bin edges.

DataSet Aspects:

[avg] Average density over coordinate.

[sd] Standard deviation of density over coordinate.

If no arguments are specified, calculate the total system density. Otherwise, calculate specified density along the given axis for atoms in specified mask(s). Defaults are shown in parentheses above. The format of the file is as follows. Comments are lines starting with '#' or empty lines. All other lines must contain the atom type followed by an integer number for the electron number. Entries must be separated by spaces or '='. Example input:

```
density out number_density.dat number delta 0.25 ":POPC@P1" ":POPC@N" \
        ":POPC@C2" ":POPC"
density out mass_density.dat mass delta 0.25 ":POPC@P1" ":POPC@N" \
        ":POPC@C2" ":POPC"
density out charge_density.dat charge delta 0.25 ":POPC@P1" ":POPC@N" \
        ":POPC@C2" ":POPC"
density out electron_density.dat electron delta 0.25 efile Nelec.in \
        ":POPC@P1" ":POPC@N" ":POPC@C2" ":POPC" ":TIP3" \
        ":POPC | :TIP3" "*"
density out ion_density.dat number delta 0.25 ":SOD" ":CLA"
```

See also $AMBERHOME/AmberTools/test/cpptraj/Test_Density.

11.22 diffusion

Note that although the syntax for ***diffusion*** *has changed as of version 16, the old syntax is still supported.*

```
diffusion [{out <filename> | separateout <suffix>}] [time <time per frame>] [noimage]
       [<mask>] [<set name>] [individual] [diffout <filename>] [nocalc]
```

[out <filename>] Write mean-square displacement (MSD) data set output to file specified by <filename>.

[separateout <suffix>] Write each MSD data set type to files with suffix <suffix>; see description below.

[time <time_per_frame>] Time in-between each coordinate frame in ps; default is 1.0.

[noimage] If specified do not perform imaging. If this is specified coordinates should be unwrapped prior to this command.

[<mask>] Mask of atoms to calculate diffusion for; default all atoms.

[<set name>] MSD data set name.

[individual] Write diffusion for each individual atom as well as average diffusion for atoms in mask.

[diffout <filename>] `Write diffusion contants calculated from fits of MSD data sets to <filename>.`

[nocalc] `Do not calculate diffusion constants.`

`DataSet Aspects:`

[X] `MSD(s) in the X direction.`

[Y] `MSD(s) in the Y direction.`

[Z] `MSD(s) in the Z direction.`

[R] `Overall MSD(s).`

[A] `Overall displacement(s).`

[D] `Diffusion constants.`

[Label] `Diffusion constant lablels.`

[Slope] `Linear regression slopes.`

[Intercept] `Linear refression Y-intercepts.`

[Corr] `Linear regression correlation coefficients.`

Compute mean square displacement (MSD) plots (using distance traveled from initial position) for the atoms in <**mask**>. By default only the diffusion averaged over all atoms in <**mask**> is calculated; if **individual** is specified diffusion for individual atoms is calculated as well.

In order to correctly calculate diffusion molecules should take continuous paths, so imaging of atoms is autoimatically performed. If the trajectory is already unwrapped (or the unwrap command is used prior to this command) the **noimage** keyword can be used.

The following types of displacements are calculated. If **separateout** is specfied the following files will be created:

x_<suffix> Mean square displacement(s) in the X direction (in Å^2/ps).

y_<suffix> Mean square displacement(s) in the Y direction (in Å^2/ps).

z_<suffix> Mean square displacement(s) in the Z direction (in Å^2/ps).

r_<suffix> Overall mean square displacement(s) (in Å^2/ps).

a_<suffix> Total distance traveled (in Å/ps).

The diffusion coefficient ***D*** can be calculated using the Einstein relation:

$$2nD = \lim_{t \to \infty} \frac{MSD}{t}$$

Where ***n*** is the number of dimensions; for overall MSD n = 3, for single dimension MSD (e.g. X) n = 1, etc. Unless **nocalc** is specified, the diffusion constant is calculated automatically from MSD data sets (and written to the file specified by **diffout**) in the following manner. The slope the plot of MSD

versus time is obtained via linear regression. To convert from units of $Å^2$/ps to 1x10^{-5} cm^2/s, the slope is multiplied by 10.0/(2n). Both the calculated diffusion constants as well as the results of the fit are reported.

Due to the fact that diffusion is currently calculated from initial positions only, diffusion calculated for small numbers of atoms will be inherently stochastic, so the results are most sensible when averaged over many atoms; for example, the diffusion of water should be calculated using all waters in the system.

For example, to calculate the diffusion of water in a system:

```
diffusion :WAT@O out WAT_O.agr WAT_O diffout DC.dat
```

11.23 dihedral

```
dihedral [<name>] <mask1> <mask2> <mask3> <mask4> [out <filename>] [mass]
         [type {alpha|beta|gamma|delta|epsilon|zeta|chi|c2p|h1p|phi|psi|omega|pchi}]
         [range360]
```

[<name>] Output data set name.

<maskX> Four atom masks selecting atom(s) to calculate dihedral for.

[out <filename>] Output file name.

[mass] Use center of mass of atoms in <maskX>; default is geometric center.

[range360] Output dihedral angle values from 0 to 360 degrees instead of -180 to 180 degrees.

[type <type>] Label dihedral as <type> for use with *statistics* analysis; note 'chi' is nucleic acid chi and 'pchi' is protein chi.

Calculate dihedral angle (in degrees) between the planes defined by atoms in <mask1>, <mask2>, <mask3> and <mask2>, <mask3>, <mask4>. To calculate multiple dihedral angles see the ***multidihedral*** command on page 123.

11.24 dihedralrms | dihrms

```
dihedralrms [<name>] <dihedral types> [out <file>]
            [ first | reference | ref <name> | refindex <#> | previous |
              reftraj <name> [parm <name> | parmindex <#>] ]
            [dihtype <name>:<a0>:<a1>:<a2>:<a3>[:<offset>] ...]
               [tgtrange <range> [refrange <range>]]
```

[<name>] Output data set name.

<dihedral types> Dihedral types to look for. Note that chip is 'protein chi', chin is 'nucleic chi'.

[out <filename>] Output file name.

[dihtype <name>:<a0>:<a1>:<a2>:<a3>[:<offset>]
Search for a custom dihedral type called <name> using atom names <a0>, <a1>, <a2>, and <a3>.
Offset: -2=<a0><a1> in previous res, -1=<a0> in previous res, 0=All <aX> in single res, 1=<a3> in next res, 2=<a2><a3> in next res.

[tgtrange <range>] Residue range to look for target dihedrals in. Default is all solute residues.

[refrange <range>] Residues range to look for reference dihedrals in. If not specified, use target range.

Calculate RMSD of selected dihedrals to dihedrals in a reference structure. See the ***multidihedral*** command syntax on page 123 for a list of all available dihedral types.

11.25 dihedralscan

This command has been replaced by ***permutedihedrals***; see 7.8 on page 33.

11.26 dipole

```
dipole <filename> {data <dsname> | <nx> <dx> <ny> <dy> <nz> <dz> [gridcenter <cx> <cy>
    [box|origin|center <mask>] [negative] [name <gridname>]
       <mask1> {origin | box} [max <max_percent>]
```

NOTE: This command is not well-tested and may be obsolete.

Same as **grid** (see 11.36 on page 106 below) except that dipoles of the solvent molecules are binned. The output file format is for Chris Bayly's discern delegate program that comes with Midas/Plus. Consult the code in Action_Dipole.cpp for more information.

11.27 distance

```
distance [<name>] <mask1> [<mask2>] [point <X> <Y> <Z>]
         [ reference | ref <name> | refindex <#> ]
      [out <filename>] [geom] [noimage] [type noe]
Options for 'type noe':
  [bound <lower> bound <upper>] [rexp <expected>] [noe_strong] [noe_medium] [noe_weak]
```

[<name>] Output data set name

<mask1> Atom mask selecting atom(s) to calculate distance between.

<mask2> If specified, second atom mask selection atom(s) to calculate distance from <mask1>.

point <**X**> <**Y**> <**Z**> If specified instead of second mask, calculate distance between <mask1> and specified XYZ coordinates.

reference | **ref** <**name**> | **refindex** <#> If specified, calculate distance between <mask1> and <mask2> in specified reference.

[out <**filename**>**]** Output filename.

[geom] Use geometric center of atoms in <mask1>/<mask2>; default is to use center of mass.

[noimage] Do not image distances across periodic boundaries.

[type noe] Mark distance as 'noe' for use with *statistics* analysis.

[bound <**lower**> **bound** <**upper**>**]** Lower and upper bounds for NOE (in Angstroms); must specify both.

[rexp <**expected**>**]** Expected value for NOE (in Angstroms); if not given '(<lower> + <upper>)' / 2.0 is used.

[noe_strong] Set lower and upper bounds to 1.8 and 2.9 Å respectively.

[noe_medium] Set lower and upper bounds to 2.9 and 3.5 Å respectively.

[noe_weak] Set lower and upper bounds to 3.5 and 5.0 Å respectively.

Calculate distance between the center of mass of atoms in <mask1> to atoms in <mask2>, between atoms in <mask1> and atoms in <mask2> in specified reference, or atoms in <mask1> and the specified point. If **geom** is specified use the geometric center instead. For periodic systems imaging is turned on by default; the **noimage** keyword disables imaging.

A distance can be labeled using 'type noe' for further analysis as an NOE using the ***'statistics'*** analysis command (12.33 on page 208).

11.28 drms | drmsd (distance RMSD)

```
drmsd [<dataset name>] [<mask> [<refmask>]] [out <filename>]
      [ first | ref <refname> | refindex <#> |
        reftraj <trajname> [parm <trajparm> | parmindex <parm#>] ]
```

[<**dataset name**>**]** Output data set name.

[<**mask**>**]** Atoms to calculate DRMSD for.

[<refmask>] Mask corresponding to atoms in reference; if not specified, <mask> is used.

[out <filename>] Output file name.

[first] Use the first trajectory frame processed as reference.

[reference] Use the first previously read in reference structure.

[ref <refname>] Use previously read in reference structure specified by <refname>.

[refindex <#>] Use previously read in reference structure specified by <#> (based on order read in).

previous Use frame prior to current frame as reference.

reftraj <name> Use frames from COORDS set <name> or read in from trajectory file <name> as references. Each frame from <name> is used in turn, so that frame 1 is compared to frame 1 from <name>, frame 2 is compared to frame 2 from <name> and so on. If <trajname> runs out of frames before processing is complete, the last frame of <trajname> continues to be used as the reference.

parm <parmname> | parmindex <#> If reftraj specifies a file associate trajectory <name> with specified topology; if not specified the first topology is used.

Calculate the distance RMSD (i.e. the RMSD of all pairs of internal distances) between atoms in the frame defined by <**mask**> (all if no <**mask**> specified) to atoms in a reference defined by <**refmask**> (<**mask**> if <**refmask**> not specified). Both <**mask**> and <**refmask**> must specify the same number of atoms, otherwise an error will occur.

Because this method compares pairs of internal distances and not absolute coordinates, it is not sensitive to translations and rotations the way that a no-fit RMSD calculation is. It can be more time consuming however, as $(N^2\text{-}N)/2$ distances must be calculated and compared for both the target and reference structures.

For example, to get the DRMSD of a residue named LIG to its structure in the first frame read in:

```
drmsd :LIG first out drmsd.dat
```

11.29 dssp

See 11.72 on page 147.

11.30 energy

```
energy [<name>] [<mask1>] [out <filename>]
    [bond] [angle] [dihedral] {[nb14] | [e14] | [v14]}
    {[nonbond] | [elec] [vdw]} [kinetic [ketype {vel|vv}] [dt <dt>]]
    [ etype { simple |
              directsum [npoints <N>] |
              ewald [cut <cutoff>] [dsumtol <dtol>] [rsumtol <rtol>]
                    [ewcoeff <coeff>] [maxexp <max>] [skinnb <skinnb>]
                    [mlimits <X>,<Y>,<Z>] [erfcdx <dx>]
              pme [cut <cutoff>] [dsumtol <dtol>] [order <order>] [ljswidth <width>]
                  [ewcoeff <coeff>] [ljpme] [ewcoefflj] [skinnb <skinnb>]
                  [nfft <nfft1>,<nfft2>,<nfft3>] [erfcdx <dx>]
            } ]
```

[<name>] Data set name.

[<mask1>] Mask of atoms to calculate energy for.

[out <filename>] File to write results to.

[bond] Calculate bond energy.

[angle] Calculate angle energy.

[dihedral] Calculate dihedral energy.

[nb14] Calculate nonbonded 1-4 energy.

[e14] Calculate 1-4 electrostatics.

[v14] Calculate 1-4 van der Waals.

[nonbond] Calculate nonbonded energy (electrostatics and van der Waals).

[elec] Calculate electrostatic energy (Coulomb potential).

[vdw] Calculate van der Waals energy (Lennard-Jones 6-12 potential).

[etype <type>] Calculate electrostatics via specified type.

[simple] Use simple Coulomb term for electrostatics, no cutoff.

[directsum] Use direct summation method for electrostatics.

[npoints <N>] Number of cells in each direction to calculate the direct sum.

[ewald] Use Ewald summation for electrostatics. If van der Waals energy will be calculated a long-range correction for periodicity will be applied.

cut <**cutoff**> Direct space cutoff in Angstroms (default 8.0).

dsumtol <**dtol**> Direct sum tolerance (default 0.00001). Used to determine Ewald coefficient.

rsumtol <**rtol**> Reciprocal sum tolerance (default 0.00005). Used to determine number of reciprocal space vectors.

ewcoeff <**coeff**> Ewald coefficient in 1/Ang.

skinnb Used to determine pairlist atoms (added to cut, so pairlist cutoff is cut + skinnb); included in order to maintain consistency with results from sander.

mlimits <**X**>,<**Y**>,<**Z**> Explicitly set the number of reciprocal space vectors in each dimension. Will be determined automatically if not specified.

erfcdx <**dx**> Spacing to use for the ERFC splines (default 0.0002 Ang.).

[pme] Use particle mesh Ewald for electrostatics. If van der Waals energy will be calculated a long-range correction for periodicity will be applied.

cut <**cutoff**> Direct space cutoff in Angstroms (default 8.0).

dsumtol <**dtol**> Direct sum tolerance (default 0.00001). Used to determine Ewald coefficient.

order <**order**> Spline order for charges.

ljswidth <**width**> If specified, use a force-switching form for the Lennard-Jones calculation from <cutoff>-<width> to <cutoff>.

ewcoeff <**coeff**> Ewald coefficient in 1/Ang.

ljpme If specified use particle mesh Ewald for calculating Lennard-Jones interactions.

ewcoefflj Ewald coefficient for Lennard-Jones PME.

skinnb Used to determine pairlist atoms (added to cut, so pairlist cutoff is cut + skinnb); included in order to maintain consistency with results from sander.

nfft <**nfft1**>,<**nfft2**>,<**nfft3**> Explicitly set the number of FFT grid points in each dimension. Will be determined automatically if not specified.

erfcdx <**dx**> Spacing to use for the ERFC splines (default 0.0002 Ang.).

DataSet Aspects:

[bond] Bond energy.

[angle] Angle energy.

[dih] Dihedral energy.

[vdw14] 1-4 van der Waals energy.

[elec14] 1-4 electrostatic energy.

[vdw] van der Waals energy.

[elec] Electrostatic energy.

[total] Total energy.

Calculate the energy for atoms in <mask>. If no terms are specified, all terms are calculated. Note that the non-bonded energy terms for **'simple'** do not take into account periodicity and there is no distance cut-off. Electrostatics can also be determined via the direct sum, Ewald, or particle-mesh Ewald summation procedures. The particle mesh Ewald functionality requires that CPPTRAJ be compiled with FFTW and a C++11 compliant compiler.

Calculation of energy terms requires that the associated topology file have parameters for any of the calculated terms, so for example angle calculations are not possible when using a PDB file as a topology, etc. All nonbonded calculations methods other than **simple** require unit cell parameters.

For example, to calculate all energy terms and write to a Grace-format file:

```
parm DPDP.parm7
trajin DPDP.nc
energy DPDP out ene.agr
```

11.31 esander

```
esander [<name>] [out <filename>] [saveforces] [parmname <file>] [keepfiles]
        [<namelist vars> ...]
```

[<name>] Data set name.

[out <filename>] File to write results to.

[saveforces] If specified, save forces to frames.
Requires writing frames in NetCDF format.

[parmname <file>] Name of temporary topology file
(default: 'CpptrajEsander.parm7').

[keepfiles] Keep temporary topology file after program
execution.

[<namelist vars>] Namelist variables supported by the
sander API in format 'var <value>'; see below.

Calculate energies for input frames using the sander API. It requires compilation with the SANDER API (sanderlib). This can be considered as a faster alternative to energy post-processing with sander (imin = 5). Currently the following sander namelist variables are supported: **extidel**, **intdiel**, **rgbmax**, **saltcon**, **cut**, **dielc**, **igb**, **alpb**, **gbsa**, **lj1264**, **ipb**, **inp**, **vdwmeth**, **ew_type**, **ntb**, **ntf**, **ntc**. See **??** on page ?? for details.

If ntb/cut/igb are not specified cpptraj will attempt to pick reasonable values based on the input system. The defaults for a non-periodic system are ntb=0, cut=9999.0, igb=1. The defaults for a periodic system are ntb=1, cut=8.0, igb=0. This currently requires writing a temporary Amber topology, the name of which can be set by **parmname**. If **keepfiles** is specified this temporary topology will not be deleted after execution.

For example, to calculate energies for a non-periodic system using igb=1 (the default) with GB surface area turned on (gbsa=1):

```
parm DPDP.parm7
trajin DPDP.nc
esander DPDP out Edpdp.dat gbsa 1
```

11.32 filter

```
filter <dataset1 arg> min <min1> max <max1>
       [<dataset2 arg> min <min2> max <max2> ...]
       [out <file> [name <setname>]]
```

<datasetX arg> Data set name(s) to use for filtering

min <minX> Allow values greater than <min> in dataset X.

max <maxX> Allow values greater than <max> in dataset X.

[out <file>] File containing 1 for frames that were allowed, 0 for frames that were filtered.

[name <setname>] Filtered data set name containing 1 for allowed frames, 0 for filtered frames.

For all following actions, only include frames that are between **<min>** and **<max>** of data sets in **<dataset arg>**. There must be at least one **<min>** and **<max>** argument, and there must be as many **<min>**/**<max>** arguments as there are specified data sets. For example, to write only frames in-between an RMSD of 0.7-0.8 Angstroms for a given input trajectory:

```
trajin ../tz2.truncoct.nc
rms R1 first :2-11
filter R1 min 0.7 max 0.8 out filter.dat
outtraj maxmin.crd
```

The output trajectory will only contain frames that meet the RMSD requirement, and the *filter.dat* file can be used to see which frames those were that were output.

A similar command that can be used with data that already exists (e.g. it has been read in with ***readdata***) is ***datafilter*** (see page 39).

11.33 fixatomorder

```
fixatomorder [outprefix <name>]
```

Cpptraj (and most of Amber) expects that atom indices in molecules to increase monotonically. However, occasionally atom indices in molecules can become disordered or non-sequential, in which case cpptraj will print an error message such as the following:

```
Error: Atom 45 was assigned a lower molecule # (1) than previous atom (2).
```

and:

```
Error: Could not determine molecule information for <topology file>.
```

. This command fixes atom ordering so that all atoms in molecules are sequential. The **outprefix** keyword will write out the re-ordered topology with name <**name**>.<**original name**>.

For example, given an out of order topology named 'outoforder.parm7' and a corresponding trajectory 'min1.crd', the following will produce a reordered topology named 'reorder.outoforder.parm7' and a reordered trajectory named 'reorder.mdcrd':

```
parm outoforder.parm7
trajin min1.crd 1 10
fixatomorder outprefix reorder
trajout reorder.mdcrd
```

11.34 fiximagedbonds

```
fiximagedbonds [<mask>]
```

<**mask**> Mask expression of atoms to check.

Fix bonds that have been split across periodic boundary conditions by imaging. It may be desirable to reimage the coordinates after this with ***autoimage***.

11.35 gist (Grid Inhomogeneous Solvation Theory)

gist [doorder] [doeij] [skipE] [refdens <rdval>] [temp <tval>]
[noimage] [gridcntr <xval> <yval> <zval>] [excludeions]
[griddim <xval> <yval> <zval>] [gridspacn <spaceval>]
[prefix <filename prefix>] [ext <grid extension>] [out <output>]
[info <info>]

[doorder] Calculate the water order parameter [3] for each voxel.

[doeij] Calculate the triangular matrix representing the water-water interactions between pairs of voxels (see below).

[skipE] Skip all energy calculations (cannot be specified with 'doeij').

[refdens rdval>] Reference density of bulk water, used in computing g_O, g_H, and the translational entropy. Default is 0.0334 molecules/Å^3.

[temp <tval>] Temperature of the input trajectory.

[noimage] Disable distance imaging in energy calculation.

[excludeions] If specified, exclude any ions from the calculation.

[gridcntr <xval> <yval> <zval>] Coordinates (Å) of the center of the grid (default 0.0, 0.0, 0.0).

[griddim <xval> <yval> <zval>] Grid dimensions along each coordinate axis (default 40, 40, 40).

[gridspacn <spaceval>] Grid spacing (linear dimension of each voxel) in Angstroms. Values greater than 0.75 Å are not recommended (default 0.5 Å).

[prefix <filename prefix>] Output file name prefix (default "gist").

[ext <grid extension>] Output grid file name extension (default ".dx").

[out <output>] Name of the main GIST output file. If not specified set to '<prefix>-output.dat'.

[info <info>] Name of main GIST info file. If not specified info is written to standard output.

DataSet Aspects:

[gO] Number density of oxygen centers found in the voxel, in units of the bulk density.

[gH] `Number density of hydrogen centers found in the voxel in units of the reference bulk density.`

[Esw] `Mean solute-water interaction energy density.`

[Eww] `Mean water-water interaction energy density.`

[dTStrans] `First order translational entropy density.`

[dTSorient] `First order orientational entropy density .`

[neighbor] `Mean number of waters neighboring the water molecules found in this voxel multiplied by the voxel number density.`

[dipole] `Magnitude of mean dipole moment (polarization).`

[order] `Average Tetrahedral Order Parameter.`

[dipolex] `x-component of the mean water dipole moment density`

[dipoley] `y-component of the mean water dipole moment density`

[dipolez] `z-component of the mean water dipole moment density`

[Eij] `Water-water interaction matrix.`

Grid Inhomogeneous Solvation Theory [4, 5] (GIST) is a method for analyzing the structure and thermodynamics of solvent in the vicinity of a solute molecule. The current implementation works for only water, but the method can be generalized to other solvents whose molecules are rigid like water, such as chloroform or dimethylsulfoxide (DMSO). GIST post-processes explicit solvent simulation data to create a three-dimensional mapping of water density and thermodynamic properties within a region of interest, which is defined by a user-specified 3D rectangular grid. The small grid boxes are referred to as voxels, and each voxel is associated with solvent properties. (See Fig. 1.) The GIST implementation incorporated into AmberTools *cpptraj* also calculates a number of other local water properties, as listed below. GIST works for the nonpolarizable water models currently supported by AMBER.

In order to carry out a GIST calculation, you must have a trajectory file generated with explicit water, as well as the corresponding topology file. To generate the most readily interpretable results, it is recommended that the solute (e.g., a protein) be restrained into essentially one conformation. GIST will then provide information about the structure and thermodynamics of the solvent for that conformation. For a room-temperature simulation of a solvent-exposed binding site, and a grid-spacing of 0.5 Å, it is recommended that the simulation be at least 10-20 ns in duration, and it is also a good idea to check for convergence of the GIST properties you are interested in by loading and then processing successively more frames of your trajectory file. Because GIST assumes that the solute of interest comprises all molecules in the simulation that are not waters, it is a good idea to remove all counterions and cosolutes with cpptraj's

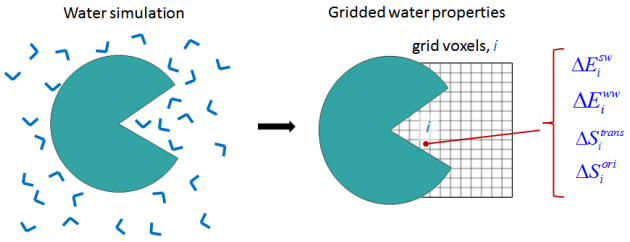

Figure 1: Diagram, in 2D, of GIST's gridded water properties in a binding site.

strip command before running GIST. A sample series of cpptraj commands for running GIST is provided below.

Although it is not mandatory to supply values of **gridcntr**, **griddim** and **gridspcn**, these parameters should be carefully chosen, because they determine the region to be analyzed (**gridcntr** and **griddim**) and the spatial resolution and convergence properties of the results (**gridspcn**). In particular, although smaller grid spacings will give finer spatial resolution, longer simulation times will be needed to converge the properties in the smaller voxels that result. A larger grid spacing will allow earlier convergence, but will smooth the spatial distributions and hence can reduce accuracy.

The reference density of water (**rdval**) is taken by default to be the experimental number density of pure water at 300 K and 1 atm. However, different water models may yield slightly different bulk densities under these conditions, and the density also depends on T and P. If you know that the bulk density of the water model you are using, at the T and P of your simulation, deviates significantly from 0.0334 water molecules/Å^3, it would be advisable to supply the actual value with the **refdens** keyword, instead of allowing GIST to supply the default value.

GIST Output

GIST generates a main output file and a collection of grid data files that by default are in Data Explorer format (.dx); this can be changed via the **ext** keyword. These grid files enable visualization of the various gridded quantities, such as with the program VMD [6]. If the **doeij** keyword is provided, GIST also writes out a matrix of water-water interactions between pairs of voxels. In addition, run details are written to stdout, which can be redirected into a log file.

Note that a number of quantities are written out as both densities and normalized quantities. For example, the output file includes both the solute-

water energy density and the normalized (per water) solute-water energy. In all cases, the normalized quantity at voxel i, $X_{i,norm}$ is related to the corresponding density, $X_{i,dens}$, by the relationship $X_{i,norm} = \rho_i X_{i,dens}$, where ρ_i is the number density of water in the voxel. The normalized quantity provides information regarding the nature of the water found in the voxel. The density has the property that, if the grid extended over the entire simulation volume, the total system quantity would be given by $X_{tot} = V_{voxel} \sum_i X_{i,dens}$, where V_{voxel} is the volume of one grid voxel.

The main output file takes the form of a space-delimited-variable file, where each row corresponds to one voxel of the grid. This file can easily be opened with and manipulated with spreadsheet programs like Excel and LibreOffice Calc. The columns are as follows.

- **index** - A unique, sequential integer assigned to each voxel
- **xcoord** - x coordinate of the center of the voxel (Å)
- **ycoord** - y coordinate of the center of the voxel (Å)
- **zcoord** - z coordinate of the center of the voxel (Å)
- **population** - Number of water molecule, n_i, found in the voxel over the entire simulation. A water molecule is deemed to populate a voxel if its oxygen coordinates are inside the voxel. The expectation value of this quantity increases in proportion to the length of the simulation.
- **g_O** - Number density of oxygen centers found in the voxel, in units of the bulk density (rdval). Thus, the expectation value of **g_O** for a neat water system is unity.
- **g_H** - Number density of hydrogen centers found in the voxel in units of the reference bulk density (2×rdval). Thus, the expectation value of **g_H** for a neat water system would be unity.
- **dTStrans-dens** - First order translational entropy density (kcal/mole/Å^3), referenced to the translational entropy of bulk water, based on the value rdval.
- **dTStrans-norm** - First order translational entropy per water molecule (kcal/mole/molecule), referenced to the translational entropy of bulk water, based on the value rdval. The quantity **dTStrans-norm** equals **dTStrans-dens** divided by the number density of the voxel.
- **dTSorient-dens** - First order orientational entropy density (kcal/mole/Å^3), referenced to bulk solvent (see below).
- **dTSorient-norm** - First order orientational entropy per water molecule (kcal/mole/water), referenced to bulk solvent (see below). This quantity equals **dTSorient-dens** divided by the number density of the voxel.

- **Esw-dens** - Mean solute-water interaction energy density (kcal/mole/Å^3). This is the interaction of the solvent in a given voxel with the entire solute. Both Lennard-Jones and electrostatic interactions are computed without any cutoff, within the minimum image convention but without Ewald summation. This quantity is referenced to bulk, in the trivial sense that the solute-solvent interaction energy is zero in bulk.
- **Esw-norm** - Mean solute-water interaction energy per water molecule. This equals **Esw-dens** divided by the number density of the voxel (kcal/mole/molecule).
- **Eww-dens** - Mean water-water interaction energy density, scaled by $\frac{1}{2}$ to prevent double-counting, and not referenced to the corresponding bulk value of this quantity (see below). This quantity is one half of the mean interaction energy of the water in a given voxel with all other waters in the system, both on and off the GIST grid, divided by the volume of the voxel (kcal/mole/Å^3). Again, both Lennard-Jones and electrostatic interactions are computed without any cutoff, within the minimum image convention.
- **Eww-norm** - Mean water-water interaction energy, normalized to the mean number of water molecules in the voxel (kcal/mole/water). See prior column definition for details.
- **Dipole_x-dens** - x-component of the mean water dipole moment density (Debye/Å^3).
- **Dipole_y-dens** - y-component of the mean water dipole moment density (Debye/Å^3).
- **Dipole_z-dens** - z-component of the mean water dipole moment density (Debye/Å^3).
- **Dipole-dens** - Magnitude of mean dipole moment (polarization) (Debye/Å^3).
- **Neighbor-dens** - Mean number of waters neighboring the water molecules found in this voxel multiplied by the voxel number density. Two waters are considered neighbors if their oxygens are within 3.5 angstroms of each other. For any given frame, the contribution to the average is set to zero if no water is found in the voxel (units of number/Å^3).
- **Neighbor-norm** - Mean number of neighboring water molecules, per water molecule found in the voxel (units of number per water).
- **Order-norm** - Average Tetrahedral Order Parameter [3], q_{tet}, for water molecules found in the voxel, normalized by the number of waters in the voxel. The order parameter for water i in a given frame is given by: $q_{tet}(i) = 1 - \frac{3}{8}\sum_{j=1}^{3}\sum_{k=j+1}^{4}(cos\phi_{ijk} + \frac{1}{3})^2$ where j and k index the 4 closest water neighbors to water i, and ϕ_{ijk} is the angle formed by water i, j, and k. If the doorder keyword is not provided or is set to FALSE, then this calculation will not be done, and the entries in this column will be set to zero.

Grid files are provided for all computed quantities listed above, except that the normalized quantities are not included. The filenames are as follows: gist-gO.dx, gist-gH.dx, gist-dTStrans-dens.dx, gist-dTSorient-dens.dx, gist-Esw-dens.dx, gist-Eww-dens.dx, gist-dipolex-dens.dx, gist-dipoley-dens.dx, gist-dipolez-dens.dx, gist-dipole-dens.dx, gist-neighbor-dens.dx, gist-neighbor-norm.dx, gist-order-norm.dx. If the **doorder** keyword is not provided, then the data in gist-order-norm.dx will all be zeroes. Note that the file of voxel water densities, gist-gO.dx, can be used as input to the program Placevent [7], in order to define spherical hydration sites based on the density distribution.

Similar grid files with other computed quantities can be generated by reading the gist.out file into a spreadsheet program, processing the numbers to generate a new column of voxel data of interest, and writing this column to an ascii text file. Then the Perl script write_dx_file.pl, which should be available on the GIST tutorial web-site, may be used to read in the column of data and create the corresponding dx file. The input format, and an example, are as follows:

```
./write_dx_file.pl [filename] [x-dimension y-dimension z-dimension]
[x-origin y-origin z-origin] [grid spacing]
./write_dx_file.pl file.dat 40 40 40 13.0 13.0 13.0 0.75
```

If the doeij keyword is provided, GIST also writes a large file, Eww_ij.dat, containing the mean water-water interaction energies between pairs of voxels, scaled by $\frac{1}{2}$. (See below.) This file has three columns. The first two columns are voxel indexes, i, j, where $j > i$, so that no pair appears more than once, and the third column is the mean interaction energy (kcal/mole) of water in voxels i and$j$, scaled by $\frac{1}{2}$. If the occupancy of either voxel is 0, such as for voxels covered by solute atoms, then the interaction energy is zero. In order to save space, such interactions are omitted from the file.

Sample cpptraj input file to run GIST

The following input file, gist.in, causes cpptraj to read a parameter file named topology.top; read in the first 5000 frames of the trajectory file named trajectoryfile.mdcrd; strip out all Na and Cl ions; and carry out a GIST run which computes order parameters, uses a 41x41x45 grid centered at (25.0, 31.0, 30.0) with a spacing of 0.5 Å, uses the default bulk water density of 0.0334 molecules/Å^3, and generates the main output file gist.out.

```
parm topology.top
trajin trajectoryfile.mdcrd 1 5000
strip @Na
strip @Cl
gist doorder doeij gridcntr 25.0 31.0 30.0 griddim 41 41 45
    gridspacn 0.50 out gist.out
```

Water Model	Mean Energy (Eww-norm) (kcal/mol/water)	Number Density ($Å^{-3}$)
TIP3P	-9.533	0.0329
TIP4PEW	-11.036	0.0332
TIP4P	-9.856	0.0332
TIP5P	-9.596	0.0329
Tip3PFW	-11.369	0.0334
SPCE	-11.123	0.0333
SPCFW	-11.873	0.0329

Table 3: Water model energy and density.

```
go
```

To execute this run in the background, use

```
cpptraj<gist.in>gist.log& or cpptraj -i gist.in>gist.log&
```

Referencing GIST results to unperturbed (bulk) water

Inhomogeneous fluid solvation theory, which is the basis of GIST, is designed to provide information on how water structure and thermodynamics around a solute molecule, such as a protein, are changed relative to the structure and thermodynamics of unperturbed (bulk) water. Accordingly, the quantities reported by GIST are most informative when the results are referenced to the corresponding bulk water properties. For the orientational entropy, the reference value is the same regardless of water model or conditions, because the first order orientational distribution of water in the bulk is always uniform. Therefore, the GIST results for orientational entropies are already referenced to bulk. However, cpptraj reports unreferenced values for those GIST quantities whose reference values depend upon the water model and the simulation conditions; i.e., the energies. The translational entropy as well as the number densities will be referenced to bulk using the input referenced density or the default density value of 0.0334. The table below provides useful reference values for these quantities, computed for various water models at P=1atm, T=300K, using GIST in order to ensure a consistent minimum image treatment of periodic boundary conditions.

Users running calculations under significantly different conditions, or with different water models, should consider generating their own reference quantities by applying GIST to a simulation of pure water under their conditions of interest. The quantities of interest can then be obtained in their most precise available form by averaging over voxels, for the pure water simulation. If the

quantity of interest is Q, then its average reference value is $Q_{reference} = \frac{\sum n_i Q_i}{\sum n_i}$, where Q_i and n_i are, respectively, GIST's reported values of the quantity and the population in voxel i. The densities, ρ_i, are referenced to the corresponding bulk densities, ρ^o, as $g_i = \rho_i/\rho^o$, while the energy and entropy terms are referenced by subtracting their bulk values.

Interpreting GIST results

GIST provides access to the first order entropies and the first- and second-order energies of inhomogeneous fluid solvation theory. Non-zero higher-order entropies exist but are not yet computationally accessible. However, for a pairwise additive force-field, such as those listed in the Table above, the energy is fully described at the second order provided by GIST.

GIST is a research tool, and its applications (to, for example, protein-ligand binding and protein function) are still being explored. The following general comments may be helpful to users studying GIST results.

1. The water in voxels near a solute (e.g., a protein) almost always has unfavorable water-water interaction energies, relative to bulk, simply because the solute displaces water, resulting in fewer proximal water-water interactions.

2. The unfavorable water-water energies mentioned in [4] may be balanced by favorable water-solute interactions. If they are not, as may occur especially for voxels in small, hydrophobic pockets, then the net energy of the water in the voxel may be unfavorable relative to bulk, in which case a ligand which displaces water from the voxel into bulk may get a boost in affinity.

3. Because the first order orientational distribution of bulk water is uniform, and a nonuniform distribution always has lower entropy than a uniform one, the solute can only lower the orientational entropy of water, relative to bulk. Thus, this term always opposes solvation, and displacing oriented water into the bulk is always favorable from the standpoint of orientational entropy.

4. Localized water, which corresponds to voxels with high water density, has a low first order translational entropy, and the translational entropy around a solute is lower than that in bulk, as a nonuniform translational distribution takes the place of the uniform translational distribution of bulk water.

5. The displacement of highly oriented (low orientational entropy) and localized (low translational entropy) water into bulk leads to a favorable increase in these entropy terms.

6. However, highly oriented and localized water is often the consequence of strongly favorable polar interactions, such as hydrogen-bonding, between water and the solute. As a consequence, the net favorability of displacing such water is frequently a balance between favorable entropic consequences and unfavorable energetic consequences.

7. The water-water energy associated with a given voxel accounts for the interactions of the waters in this voxel with all other waters in the system,

including waters in other voxels. This quantity is multiplied by $\frac{1}{2}$, so that, in a pure-water system where the GIST grid covers the entire simulation box, the sum over all voxels equals the correct mean water-water interaction energy. Note that Reference [5] does not include this factor of $\frac{1}{2}$.

8. For a typical GIST application, in which the grid occupies only part of the simulation box, the energy bookkeeping can become complicated, as discussed in Section II.B.3 (page 044101-6) of Reference [5]. That section also explains how one can compute the water-water energy associated with a region R defined by a set of voxels, E^R_{WW}. The regional water-water energy, on a normalized (per water) basis, is given by $E^R_{WW} = 2(\sum_{i\in R} E_{i,WW} - \sum_{i\in R}\sum_{j\in R, j>i} E_{i,j,WW})$ where $i \in R$ means that voxel i is in region R, $E_{i,WW}$ is the value of Eww-norm for voxel i, and $E_{i,j,WW}$ is the value of the water-water interaction energy between voxels i and j, taken from the file Eww_ij.dat. The extra factor of 2 in the present formula, relative to that in the paper, results from application of an extra factor of $\frac{1}{2}$ to the reported water-water interaction energies here.

11.36 grid

```
grid <filename>
  { data <dsname> | boxref <ref name/tag> <nx> <ny> <nz> |
    <nx> <dx> <ny> <dy> <nz> <dz> [gridcenter <cx> <cy> <cz>] }
  [box|origin|center <mask>] [negative] [name <gridname>]
  <mask> [normframe | normdensity [density <density>]]
 [pdb <pdbout> [max <fraction>]] [{byres|mymol}]
  [[smoothdensity <value>] [invert]] [madura <madura>]
```

<filename> File to write out grid to. Use ".grid" or ".xplor" extension for XPLOR format, ".dx" for OpenDX format.

Options for setting up grid:

data <dsname> Use previously calculated/loaded grid data set named <dsname>. When using this option there is no need to specify grid bins/spacing/center.

boxref <ref name/tag> <nx> <ny> <nz> Set up grid using box information from a previously loaded reference structure. Currently the only way to set up non-orthogonal grids.

<nx> <dx> <ny> <dy> <nz> <dz> Number of grid bins and spacing in the X/Y/Z directions.

[gridcenter <cx> <cy> <cz>] Location of grid center, default is origin (0.0, 0.0, 0.0).

Options for offset during grid binning (must center grid at origin):

[box] Offset each point by location of box center prior to gridding. Cannot be used with 'gridcenter'.

[origin] No offset (default)

[center <mask>] Offset each point by center of atoms in <mask> prior to gridding. Cannot be used with 'gridcenter'.

Other options:

[negative] Grid negative density instead of positive density.

[name <gridname>] Grid data set name.

<mask> Mask of atoms to grid.

[normframe] Normalize grid bins by the number of frames.

[normdensity [density <density>]] Normalize grid bins by density: GridBin = GridBin / (Nframes * BinVolume * density). Default particle density (molecules/Ang^3) for water based on 1.0 g/mL.

[pdb <pdbout> [max <fraction>]] Write a pseudo-PDB of grid points that have density greater than <fraction> (default 0.80) of the grid max value.

[{byres|bymol}] Grid the centers of mass of residues or molecules selected by <mask>.

Less common options:

[smoothdensity <smooth>] Used to smooth density. The smoothing takes the form of GridBin = 0 if GridBin < smooth, otherwise GridBin = GridBin - (GridBin * exp[-(GridBin - smooth)^2 / (0.2 * smooth^2)]).

[invert] (Only used if smoothdensity also used) Do inverse smoothing (i.e. if GridBin > smooth).

[madura <madura>] Grid values lower than <madura> become flipped in sign, exposes low density.

Data Sets Created:

<dsname> Grid data set.

Create a grid representing the histogram of atoms in *mask1* on the 3D grid that is "*nx* * *x_spacing* by *ny* * *y_spacing* by *nz* * *z_spacing* angstroms (cubed). By default the grid is centered at the origin unless **gridcenter** is specified. Grid points can be offset by either the box center (using **box**) or the center of specified atoms (using **center <mask>**); if either of these options are used the grid must be centered at the origin. Note that the ***bounds*** command (on page 79) can be very useful for determining grid dimensions.

Note that when calculating grid densities for things like solvent/ions, the solute of interest (about which the atomic densities are binned) should be rms fit, centered and imaged prior to the ***grid*** call in order to provide any meaningful representation of the density. If the optional keyword `negative` is also specified, then these density will be stored as negative numbers. Output can be in the XPLOR or OpenDX data formats.

Examples

Grid water density around a solute.

```
trajin tz2.truncoct.nc
autoimage origin
rms first :1-13
# Create average of solute to view with grid.
average avg.mol2 :1-13
grid out.dx 20 0.5 20 0.5 20 0.5 :WAT@O
```

Generate grid from bounds command.

```
trajin tz2.ortho.nc
autoimage
rms first :1-13&!@H= mass
bounds :1-13 dx .5 name MyGrid out bounds.dat
average bounds.mol2 :1-13
# Save coordinates for second pass.
createcrd MyCoords
run
# Grid using grid data set from bounds command.
crdaction MyCoords grid bounds.xplor data MyGrid :WAT@O
```

Create non-orthogonal grid:

```
trajin tz2.truncoct.nc
reference ../tz2.truncoct.nc [REF]
autoimage triclinic
grid nonortho.dx boxref [REF] 50 50 50 :WAT@O pdb nonortho.pdb
```

11.37 hbond

```
hbond [<dsname>] [out <filename>] [<mask>] [angle <acut>] [dist <dcut>]
      [donormask <dmask> [donorhmask <dhmask>]] [acceptormask <amask>]
      [avgout <filename>] [printatomnum] [nointramol] [image]
      [solventdonor <sdmask>] [solventacceptor <samask>]
      [solvout <filename>] [bridgeout <filename>] [bridgebyatom]
      [series [uuseries <filename>] [uvseries <filename>]]
```

[<dsname>] Data set name.

[out <filename>] Write # of solute-solute hydrogen bonds (aspect [UU]) vs time to this file. If searching for solute-solvent hydrogen bonds, write # of solute-solvent hydrogen bonds (aspect [UV]) and # of bridging solvent molecules (aspect [Bridge]), as well as the residue # of the bridging solvent and the solute residues being bridged with format '<solvent resnum>(<solute res1>+<solute res2>+...+),...' (aspect [ID]).

[<mask>] Atoms to search for solute hydrogen bond donors/acceptors.

[angle <acut>] Angle cutoff for hydrogen bonds (default 135°). Can be disabled by specifying -1.

[dist <dcut>] Distance cutoff for hydrogen bonds (acceptor to donor heavy atom, default 3.0 Å).

[donormask <dmask>] Use atoms in <dmask> as solute donor heavy atoms. If 'donorhmask' not specified only atoms bonded to hydrogen will be considered donors.

[donorhmask <dhmask>] Use atoms in <dmask> as solute donor hydrogen atoms. Should only be specified if 'donormask' is. Should be a 1 to 1 correspondence between donormask and donorhmask.

[acceptormask <amask>] Use atoms in <amask> as solute acceptor atoms.

[avgout <filename>] Write solute-solute hydrogen bond averages to <filename>.

[printatomnum] Add atom numbers to the output, in addtion to residue name, residue number and atom name.

[nointramol] Ignore intramolecular hydrogen bonds.

[image] Turn on imaging of distances/angles.

[solventdonor <sdmask>] Use atoms in <sdmask> as solvent donors. Can specify ions as well.

[solventacceptor <samask>] Use atoms in <samask> as solvent acceptors. Can specify ions as well.

[solvout <filename>] Write solute-solvent hydrogen bond averages to <filename>. If not specified and 'avgout' is, solute-solvent hydrogen bonds averages will be written to that file.

[bridgeout <filename>] Write information on detected solvent bridges to <filename>. If not specified, will be written to same place as 'solvout'.

[bridgebyatom] Report bridging results by atom instead of by residue.

[series] Save hydrogen bond formed (1.0) or not formed (0.0) per frame for any detected hydrogen bond. Solute-solute hydrogen bonds are saved with aspect [solutehb], solute-solvent hydrogen bonds are saved with aspect [solventhb].

[uuseries <filename>] File to write solute-solute hbond time series data to.

[uvseries <filename>] File to write solute-solvent hbond time series data to.

Data Sets Created:

<dsname>[UU] Number of solute-solute hydrogen bonds.

<dsname>[UV] (only for solventdonor/solventacceptor) Number of solute-solvent hydrogen bonds.

<dsname>[Bridge] (only for solventdonor/solventacceptor) Number of bridging solvent molecules.

<dsname>[ID] (only for solventdonor/solventacceptor) String identifying bridging solvent residues and the solute residues they bridge.

<dsname>[solutehb] (series only) Time series for solute-solute hydrogen bonds; 1 for present, 0 for not present.

<dsname>[solventhb] (series only) Time series for solute-solvent hydrogen bonds; 1 for present, 0 for not present.

*Note that **series** data sets are not generated until hydrogen bonds are actually determined (i.e. **run** is called).*

Determine hydrogen bonds in each coordinate frame using simple geometric criteria. A hydrogen bond is defined as being between an acceptor heavy atom A, a donor hydrogen atom H, and a donor heavy atom D. If the A to D distance is less than the distance cutoff and the A-H-D angle is greater than the angle cutoff a hydrogen bond is considered formed. Imaging of distances/angles is not performed by default, but can be turned on using the **image** keyword.

Potential hydrogen bond donor/acceptor atoms are searched for as follows:

1. If just <**mask**> is specified donors and acceptors will be automatically determined from <**mask**>.

2. If **donormask** is specified donors will be determined from <**dmask**> (only atoms bonded to hydrogen will be considered valid). Optionally, **donorhmask** can be used in conjunction with **donormask** to explicitly specify the hydrogen atoms bonded to donor atoms. Acceptors will be automatically determined from <**mask**>.

3. If **acceptormask** is specified acceptors will be determined from <**amask**>. Donors will be automatically determined from <**mask**>.

4. If both **acceptormask** and **donormask** are specified only <**amask**> and <**dmask**> will be used; no searching will occur in <**mask**>.

Automatic determination of hydrogen bond donors/acceptors uses the simplistic criterion that "hydrogen bonds are FON", i.e., hydrogens bonded to F, O, and N atoms are considered donors, and F, O, and N atoms are considered acceptors. Intra-molecular hydrogen bonds can be ignored using the **nointramol** keyword.

The number of hydrogen bonds present at each frame will be determined and written to the file specified by **out**. If the **series** keyword is specified the time series for each hydrogen bond (1 for present, 0 for not present) will also be saved for subsequent analysis (e.g. with ***lifetime***, see on page 191); solute-solute hydrogen bonds will be saved to '<dataset name>[solutehb]' and solute-solvent hydrogen bonds will be saved to '<dataset name>[solventhb]'. The data set legends are set with the residues and atoms involved in the hydrogen bonds. In the case of solute to non-specific solvent hydrogen bonds, a V is used in place of solvent.

If **avgout** is specified the average of each solute-solute hydrogen bond (sorted by population) formed over the course of the trajectory is printed with the format:

```
Acceptor   DonorH   Donor   Frames   Frac   AvgDist   AvgAng
```

where *Acceptor*, *DonorH*, and *Donor* are the residue and atom name of the atoms involved in the hydrogen bond, *Frames* is the number of frames the bond is present, *Frac* is the fraction of frames the bond is present, *AvgDist* is the average distance of the bond when present, and *AvgAng* is the average angle of the bond when present. The **printatomnum** keyword can be used to print atom numbers as well.

Solute to non-specific solvent hydrogen bonds can be tracked by using the **solventdonor** and/or **solventacceptor** keywords. The number of solute-solvent hydrogen bonds and number of "bridging" solvent molecules (i.e. solvent that is hydrogen bonded to two or more different solute residues at the same time) will also be written to the file specified by **out**. These keywords can also be used to track non-specific interactions with ions. If **avgout** or **solvavg** is specified the average of each solute solvent hydrogen bond will be printed with the format:

```
Acceptor   DonorH   Donor   Count   Frac   AvgDist   AvgAng
```

where *Acceptor*, *DonorH*, and *Donor* are either the residue and atom name of the solute atoms or "SolventAcc"/"SolventH"/"SolventDnr" representing solvent, *Count* is the total number of interactions between solute and solvent (note this can be greater than the total number of frames since for any given frame more than one solvent molecule can hydrogen bond to the same place on solute and vice versa), *AvgDist* is the average distance of the bond when present, and *AvgAng* is the average angle of the bond when present. If **avgout** or **bridgeout** is specified information on residues that were bridged by a solvent molecule over the course of the trajectory will be written to <bfilename> with format:

```
Bridge Res <N0:RES0> <N1:RES1> ... , <X> frames.
```

where '<N0:RES0> ...' is a list of residues that were bridged (residue # followed by residue name) and <X> is the number of frames the residues were bridged.

hbond Examples

To search for all hydrogen bonds within residues 1-22, writing the number of hydrogen bonds per frame to "nhb.dat" and information on each hydrogen bond found to "avghb.dat":

```
hbond :1-22 out nhb.dat avgout avghb.dat
```

To search for all hydrogen bonds formed between donors in residue 1 and acceptors in residue 2:

```
hbond donormask :1 acceptormask :2 out nhb.dat avgout avghb.dat
```

To search for all intermolecular hydrogen bonds only and solute-solvent hydrogen bonds, saving time series data to HB:

```
hbond HB out nhb.dat avgout solute_avg.dat \
  solventacceptor :WAT@O solventdonor :WAT \
  solvout solvent_avg.dat bridgeout bridge.dat \
  series uuseries uuhbonds.agr uvseries uvhbonds.agr
```

To search for non-specific hydrogen bonds between solute and ions named Na+:

```
hbond HB-Ion out nhb.agr avgout ion_avg.dat \
  solventacceptor :Na+ solventdonor :Na+
```

11.38 image

```
image [origin] [center] [triclinic | familiar [com <commask>]] [<mask>]
      [ bymol | byres | byatom ] [xoffset <x>] [yoffset <y>] [zoffset <z>]
```

[origin] Image to coordinate origin (0.0, 0.0, 0.0); default is to image to box center.

[center] For bymol/byres, image by center of mass; default is to image by first atom position.

[triclinic] Force imaging with triclinic code. This is the default for non-orthorhombic cells.

[familiar [com <commask>]] Image to truncated octahedron shape. If 'com <commask>' is given, image with respect to the center of mass of atoms in <commask>.

[<mask>] Image atoms/residues/molecules in mask.

[bymol] Image by molecule (default).

[byres] Image by residue.

[byatom] Image by atom.

[xoffset <x>] Shift atoms by a factor of <x> in the X-direction.

[yoffset <y>] Shift atoms by a factor of <y> in the Y-direction.

[zoffset <z>] Shift atoms by a factor of <z> in the Z-direction.

Note this command is intended for advanced use; for most cases the ***autoimage*** command should be sufficient.

For periodic systems only, image molecules/residues/atoms that are outside of the box back into the box. Currently both orthorhombic and non-orthorhombic boxes are supported. A typical use of ***image*** is to move molecules back into the box after performing ***center***. For example, the following commands move all atoms so that the center of residue 1 is at the center of the box, then image so that all molecules that are outside the box after centering are wrapped back inside:

```
center :1
image
```

The xoffset etc. keywords can be used to shift the entire unit cell in a certain direction by the given factor, which can be useful for visualizing trajectories with periodic boundary conditions. For example, to generate a trajectory that is offset by 1.0 box length in the X direction, one could use:

```
image xoffset 1.0
trajout traj.offsetx1.nc
```

11.39 jcoupling

```
jcoupling <mask> [outfile <filename>] [kfile <param file>] [out <filename>]
          [name <dsname>]
```

<mask> Atom mask in which to search for dihedrals within.

[outfile <filename>] File to write j-coupling values to with fixed format.

[kfile <param file>] File containing Karplus parameters (default is $AMBERHOME/dat/Karplus.txt).

[out <filename>] File to write data set output to.

[name <dsname>] Data set name.

*Note data sets are not generated until **run** is called.*

Calculate J-coupling values for all dihedrals found within <**mask**> (all atoms if no mask given). In order to use this function, Karplus parameters for all dihedrals which will be calculated must be loaded. By default *cpptraj* will use the data found in $AMBERHOME/dat/Karplus.txt; if this is not found *cpptraj* will look for the file specified by the $KARPLUS environment variable.

In the Karplus parameter file each parameter set consists of two lines for each dihedral with the format:

```
[<Type>]<Name1><Name2><Name3><Name4><A><B><C>[<D>]
<Resname1>[<Resname2>...]
```

The first line defines the parameter set for a dihedral. <Type> is optional; if not given the form for calculating the J-coupling will be as described by Chou et al.[8]; if 'C' the form will be as described by Perez et al.[9]. The <NameX> parameters define the four atoms involved in the dihedral. Each <NameX> parameter is 5 characters wide, starting with a plus '+', minus '-' or space ' ' character indicating the atom belongs to the next, previous, or current residue. The remaining 4 characters are the atom name. The parameters <A>, <B>, <C>, and <D> are floating point values 6 characters wide describing the Karplus parameters. For the 'C' form A, B, and C correspond to C0, C1, and C2; D is unused and should not be specified. The second line is a list of residue names (4 characters each) to which the dihedral applies. For example:

```
C HA   CA   CB   HB    5.40 -1.37  3.61
ILE VAL
```

Describes a dihedral between atoms HA-CA-CB-HB using the Perez et al. form with constants C0=5.40, C1=-1.37, C2=3.61 applied to ILE and VAL residues.

Output can be in both a fixed format (**outfile <filename>**) and using cpptraj data set/data file formatting (**out <filename>**). The fixed format has each dihedral that is defined from <**mask1**> printed along with its calculated J-coupling value for each frame, e.g.:

```
#Frame 1
1 SER HA CA CB HB2 45.334742 4.024759
1 SER HA CA CB HB3 -69.437134 1.829510
...
```

First the frame number is printed, then for each dihedral: Residue number, residue name, atom names 1-4 in the dihedral, the value of the dihedral, the J-coupling value.

In *cpptraj* format, only the J-coupling value is written.

11.40 lessplit

`lessplit [out <filename prefix>] [average <avg filename>] <trajout args>`

[out <filename prefix>] `Write split LES trajectories to <filename prefix>.X, where X is an integer.`

[average <avg filename>] `Write trajectory of averaged LES regions to <avg filename>.`

<trajout args> `Arguments for output trajectories.`

Split and/or average LES trajectory. At least one of **'out'** or **'average'** must be specified. If both are specified they share <trajout args>.

11.41 lie

`lie [<name>] <Ligand mask> [<Surroundings mask>] [out <filename>] [noelec] [novdw] [cutvdw <cutoff>] [cutelec <cutoff>] [diel <dielc>]`

`DataSet Aspects:`

[EELEC] `Electrostatic energy (kcal/mol).`

[EVDW] `van der Waals energy (kcal/mol).`

For each frame, calculate the non-bonded interactions between all atoms in <Ligand mask> with all atoms in <Surroundings mask>. Electrostatic and van der Waals interactions will be calculated for all atom pairs. A separate electrostatic and van der Waals cutoff can be applied, the default is 12.0 Angstroms for both. <dielc> is an optional dielectric constant. Either the electrostatic or van der Waals calculations can be suppressed via the keywords noelec and novdw, respectively.

The electrostatic interactions are calculated according to a simple shifting function shown below. The data file will contain two data sets—one for electrostatic interactions and one for van der Waals interactions. Periodic topologies and trajectories are required (i.e., explicit solvent is necessary). The minimum image convention is followed.

$$E_{elec} = k\frac{q_i q_j}{r_{ij}}\left(1 - \frac{r_{ij}^2}{r_{cut}^2}\right)^2$$

11.42 lipidorder

```
order out <filename> [x|y|z] [scd] [unsat <mask>]
      [taildist <filename> [delta <resolution>] tailstart <mask>
       tailend <mask>] <mask0> ... <maskN>
```

out Output file for order parameters: Sx, Sy, Sz (each succeeded by the standard deviation), and two estimates for the deuterium-order parameter |SCD| = 0.5Sz and |SCD| = -(2Sx + Sy)/3. If scd is set then the order parameter directly computed from the C-H vectors is output.

x|y|z Reference axis. (z)

unsat Mask for unsaturated bonds. Sz is calculated for vector Cn-Cn+1. This is only relevant if scd (below) is not set, i.e. order parameters are calculated from carbon position only.

scd Calculate the deuterium-order parameter |SCD| directly from the C-H vectors (masks must contain C-H-H triplets, see below). Otherwise the order parameter is estimated from carbon positions only (masks must contain only relevant carbons). (false)

taildist Optional output file for end-to-end distances.

delta Optional resolution for taildist. (0.1)

tailstart Mask for the start of the tail. Must be given if taildist.

tailend Mask for the end of the tail. Must be given if taildist.

mask0 ... maskN Masks for each group in the lipid chain.

The order parameters S_x, S_y, S_z and |SCD| are calculated. Carbons must be given in bonding order. If **scd** the masks must be made up of C-H-H triples, hence hydrogens to double bonds must be enumerated twice while methyl groups require an additional mask which will also create two entries in the output.

$S_z$is the vector joining carbons C_{n-1} and C_{n+1}, S_x the vector normal to the $C_{n-1} - C_n$ and $C_n - C_{n+1}$ plane and S_y is the third axis in the molecular coordinate system. The order parameter is then calculated from $Sc = 0.5 < 3\cos(2\theta) > -1$, where θ is the angle to the chosen reference axis. See example input file.

Example input (all atom names according to CHARMM27 force field for POPC).

sn1 chain: order parameters S_x, S_y, S_z and $|SCD| = 0.5 \times S_z$ and $|SCD| = -(2S_x + S_y)/3$

```
lipidorder out sn1.dat z taildist e2e_sn1.dat delta 0.1 \
           tailstart ":POPC@C32" tailend ":POPC@C316" \
           ":POPC@C32" ":POPC@C33" ":POPC@C34" ":POPC@C35" \
           ":POPC@C36" ":POPC@C37" ":POPC@C38" ":POPC@C39" \
           ":POPC@C310" ":POPC@C311" ":POPC@C312" ":POPC@C313" \
           ":POPC@C314" ":POPC@C315" ":POPC@C316"
```

See also `$AMBERHOME/AmberTools/test/cpptraj/Test_LipidOrder`.

11.43 lipidscd

`lipidscd [<name>] [<mask>] [{x|y|z}] [out <file>] [p2]`

<name> Output data set name.

<mask> Atom mask specifying where to search for lipids.

x|y|z Axis to calculate order parameters with respect to (default z).

out <file> File to write order parameters to.

p2 If specified, report raw <P2> values.

DataSets Generated:

<name>[H1]:<idx> Hold lipid order parameters for each C-H1. Each lipid type will have a different <idx> starting from 0.

<name>[H2]:<idx> Hold lipid order parameters for each C-H2. If no H2, the C-H1 value will be used.

<name>[H3]:<idx> Hold lipid order parameters for each C-H3. If no H3, the C-H2/C-H1 value will be used.

<name>[SDHX]:<idx> Hold standard deviation of lipid order parameters for each C-HX.

Calculate lipid order parameters SCD (|<P2>|) for lipid chains in mask <mask>. Lipid chains are identified by carboxyl groups, i.e. O-(C=O)-C1-...-CN, where C1 is the first carbon in the acyl chain and CN is the last. Order parameters will be determined for each hydrogen bonded to each carbon. If 'p2' is specified the raw <P2> values will be reported.

11.44 makestructure

`makestructure <List of Args>`

Apply dihedrals to specified residues using arguments found in <List of Args>, where an argument is 1 or more of the following arg types:

<sstype keyword>:<res range>

Apply secondary structure type (via phi/psi backbone angles) to residues in given range. If the secondary structure type is a turn, the residue range must correspond to a multiple of 2 residues.

Keyword	phi, psi (deg.)	# residues
alpha	-57.8, -47.0	1
left	-57.8, 47.0	1
pp2	-75.0, 145.0	1
hairpin	-100.0, 130.0	1
extended	-150.0, 155.0	1
typeI	-60.0, -30.0 | -90.0, 0.0	2
typeII	-60.0, 120.0 | 80.0, 0.0	2
typeVIII	-60.0, -30.0 | -120.0, 120.0	2
typeI'	60.0, 30.0 | 90.0, 0.0	2
typeII	60.0, -120.0 | -80.0, 0.0	2
typeVIa1	-60.0, 120.0 | -90.0, 0.0	2
typeVIa2	-120.0, 120.0 | -60.0, 0.0	2
typeVIb	-135.0, 135.0 | -75.0, 160.0	2

<custom ss name>:<res range>[:<phi>:<psi>]

If <phi> and <psi> are given, define a custom secondary structure conformation named <custom_ss> and apply to residues in range. If <custom_ss> has been previously defined then apply it to residues in range.

<custom turn name>:<res range>[:<phi1>:<psi1>:<phi2>:<psi2>]

If <phi1>, <psi1>, <phi2>, and <psi2> are given, defined a custom turn conformation named <custom_turn> and apply to residues in range (range must correspond to a multiple of 2 residues). If <custom_turn> has been previously defined then apply it to residues in range.

<custom dih name>:<res range>[:<dih type>:<angle>]

```
<dih type> = alpha beta gamma delta epsilon zeta nu0 nu1 nu2 nu3 nu4
             h1p c2p chin phi psi chip omega chi2 chi3 chi4 chi5
```

If <dih type> and <angle> are given, apply <angle> to selected dihedrals of type in range. If <custom dih> has been previously defined then apply it to residues in range.

<custom dih name>:<res range>[:<at0>:<at1>:<at2>:<at3>:<angle>[:<offset>]]

Apply <angle> to dihedral defined by atoms <at1>, <at2>, <at3>, and <at4>, or use previously defined <custom_dih>.

<offset>	Description
-2	<at0> and <at1> in previous residue.
-1	<at0> in previous residue.
0	All atoms in single residue.
1	<at3> in next residue.
2	<at2> and <at3> in next residue.

ref:<range>:<refname>[:<ref range>[:<dih types>]]

Apply dihedrals from residues <ref_range> in previously loaded reference structure <refname> to dihedrals in <range>. If <ref range> is specified, use those residues from reference. The dihedral types to be used (see **<dih_type>** above) can be specified in a comma-separated list; default is phi/psi. Note that in order to specify <dih types>, <ref range> must be specified.

Examples

Assign polyproline II structure to residues 1 through 13:

```
makestructure pp2:1-13
```

Make residues 1 and 12 'extended', residues 6 and 7 a type I' turn, and two custom assignments, one (custom1) for residues 2-5, the other (custom2) for residues 8-11:

```
makestructure extended:1,12 \
              custom1:2-5:-80.0:130.0:-130.0:140.0 \
              typeI':6-7 \
              custom2:8-11:-140.0:170.0:-100.0:140.0
```

Assign residue 5 phi 90 degrees, residues 6 and 7 phi=-70 and psi=60 degrees:

```
makestructure customdih:5:phi:90 custom:6,7:-70:60
```

Create a new dihedral named chi1 and assign it a value of 35 degrees in residue 8:

```
makestructure chi1:8:N:CA:CB:CG:35
```

Assign 'extended' structure to residues 1 and 12, a custom turn to residues 2-5 and 8-11, and a typeI' turn to residues 6-7:

```
makestructure extended:1,12 \
              custom1:2-5:-80.0:130.0:-130.0:140.0 \
              typeI':6-7 \
              custom1:8-11
```

Assign secondary structure from reference structure:

```
parm ../tz2.parm7
reference ../tz2.rst7
trajin pp2.rst7.save
makestructure "ref:1-13:tz2.rst7" rmsd reference
trajout fromref.pdb multi
```

11.45 mask

```
mask <mask> [maskout <filename>] [maskpdb <pdbname>] [maskmol2 <mol2name>]
```

<mask> Atom mask to process.

[maskout <filename>] Write information on atoms in <mask> to <filename>.

[maskpdb <name>] Write PDB of atoms in <mask> to <name>.X.

[maskmol2 <name>] Write Mol2 of atoms in <mask> to <name>.X.

For each frame determine all atoms that correspond to **<mask>**. This is most useful when using distance-based masks, since the atoms in the mask are updated for every frame read in. If **maskout** is specified information on all atoms in <mask> will be written to <filename> with format:

```
#Frame AtomNum Atom ResNum Res MolNum
```

where `#Frame` is the frame number, `AtomNum` is the number of the selected atom, `Atom` is the name of the selected atom, `ResNum` is the residue number of the selected atom, `Res` is the residue name, and `MolNum` is the molecule number of the selected atom.

If **maskpdb** or **maskmol2** are specified a PDB/Mol2 file corresponding to <mask> will be written out every frame with name "*<name>.frame#*".

For example, to write out all atoms within 3.0 Angstroms of residue 195 that are part of residues named WAT to "Res195WAT.dat", as well as write out corresponding PDB files:

```
mask "(:195<:3.0)&:WAT" maskout Res195WAT.dat maskpdb Res195WAT.pdb
```

11.46 matrix

```
matrix [out <filename>] [start <#>] [stop|end <#>] [offset <#>]
       [name <name>] [ byatom | byres [mass] | bymask [mass] ]
       [ ired [order <#>] ]
       [ {distcovar | idea} <mask1> ]
       [ {dist | correl | covar | mwcovar} <mask1> [<mask2>] ]
       [ dihcovar dihedrals <dataset arg> ]
```

[out <filename>] `If specified, write matrix to <filename>.`

[start <#>] [stop|end <#>] [offset <#>] `Start, stop, and offset frames to use (as a subset of all frames read in).`

[name <name>] `Name of the matrix dataset (for referral in subsequent analysis).`

byatom `Write results by atom (default). This is the sole option for covar, mwcovar, and ired.`

byres `Write results by calculating an average for each residue (mass weighted if mass is specified).`

bymask `Write average over <mask1>, and if <mask2> is specified <mask1> x <mask2> and <mask2> as well (mass weighted if mass is specified).`

Calculate matrix of the specified type from input coordinate frames:

dist <mask1> [<mask2>] Distance matrix (default).

correl <mask1> [<mask2>] Correlation matrix (aka dynamic cross correlation[10]).

covar <mask1> [<mask2>] Coordinate covariance matrix.

mwcovar <mask1> [<mask2>] Mass-weighted coordinate covariance matrix.

distcovar <mask1> Distance covariance matrix.

idea <mask1> Isotropically Distributed Ensemble Analysis matrix.[11]

ired [order <#>] Isotropic Reorientational Eigenmode Dynamics matrix[12] with Legendre polynomials of specified order (default 1). IRED vectors must have been specified previously with '***vector* ired**' (see 11.84 on page 157).

dihcovar dihedrals <dataset arg> Dihedral covariance matrix. Dihedral data sets must have been previously defined with e.g. ***dihedral*** or ***multidihedral*** commands or read in externally with ***readdata*** and marked as dihedrals.

Matrix dimensions will be of the order of N x M for **dist**, **correl**, **idea**, and **ired**, 2N x 2N for **dihcovar**, 3N x 3M for **covar** and **mwcovar**, and N(N-1) x N(N-1) / 4 for **distcovar** (with N being the number of data sets in the case of **ired** and **dihcovar** and the number of atoms in **<mask1>** otherwise, and M being the number of atoms in **<mask2>** if specified or **<mask1>** otherwise). No mask is required for **ired**; the matrix will be made up of previously defined IRED vectors (see the ***vector*** command on page 157). Similarly no mask is required for dihcovar; dihedral data sets must have been previously defined.

Only one mask can be used with **distcovar** and **idea** matrices (i.e. they can be symmetric only), otherwise one or two masks can be used (for symmetric and full matrices respectively). If two masks are specified the number of atoms covered by *mask1* must be greater than or equal to the number of atoms covered by *mask2,* and on output <**mask1**> corresponds to columns while <**mask2**> corresponds to rows.

As a simple example, a distance matrix of all CA atoms is generated and output to distmat.dat.

```
matrix dist @CA out distmat.dat
```

11.47 mindist

This functionality is now part of the ***nativecontacts*** command; see 11.53 on page 129.

11.48 minimage

```
minimage [<name>] <mask1> <mask2> [out <filename>] [geom] [maskcenter]
```

<**name**> Data set name.

<**mask1**> First atom mask.

<**mask2**> Second atom mask.

out <**filename**> File to write to.

geom (maskcenter only) If specified, use geometric center instead of center of mass.

maskcenter Calculate distance from center of masks instead of between each atom.

Data Sets Created:

<**name**> Minimum distance to an image in Ang.

<**name**>**[A1]** Atom number in mask 1 involved in minimum distance.

<**name**>**[A2]** Atom number in mask 2 involved in minimum distance.

Calculate the shortest distance to an image, i.e. the distance to a neighboring unit cell, as well as the numbers of the atoms involved in the distance. By default the distance between each atom in <mask1> and <mask2> is considered; if **maskcenter** is specified the center of the masks is used. By convention, the lower atom number is saved as A1 and the higher is saved as A2.

11.49 molsurf

```
molsurf [<name>] [<mask>] [out filename] [probe <probe_rad>]
     [radii {gb | parse | vdw}] [offset <rad_offset>]
```

[<name>] Name of surface area data set.

[<mask>] Atoms to calculate surface area of.

[out <filename>] File to write values to.

[probe <probe_rad>] Probe radius (default 1.4 Angstrom).

[offset <rad_offset>] Add <rad_offset> to each atom radius (default 0.0).

[radii {gb|parse|vdw}] Specify radii to use:

gb GB radii (default).

parse PARSE radii.

vdw van der Waals radii.

Calculate the Connolly surface area[13] of atoms in <mask> (default all atoms if no mask specified) using routines from molsurf (originally developed by Paul Beroza) using the probe radius specified by **probe** (1.4 Å if not specified). Note that if GB/VDW radii are not present in the topology file (e.g. for PDB files), then PARSE radii can be used. Also note that this routine only calculate absolute surface areas, i.e. it cannot be used to get the contribution of a subset of atoms to overall surface area; if such functionality is needed try the ***surf*** command (11.77 on page 153).

11.50 multidihedral

```
multidihedral [<name>] <dihedral types> [resrange <range>] [out <filename>] [range360]
              [dihtype <name>:<a0>:<a1>:<a2>:<a3>[:<offset>] ...]
                  Offset -2=<at0><at1> in previous res, -1=<at0> in previous res,
                          0=All <atX> in single res,
                          1=<at3> in next res, 2=<at2><at3> in next res.
              <dihedral types> =  alpha beta gamma delta epsilon zeta
                                  nu0 nu1 nu2 nu3 nu4 h1p c2p chin
                                  phi psi chip omega chi2 chi3 chi4 chi5
```

[<name>] Output data set name.

<dihedral types> Dihedral types to look for. Note that chip is 'protein chi', chin is 'nucleic chi'.

[resrange <range>] Residue range to look for dihedrals in. Default is all solute residues.

[out <filename>] Output file name.

[range360] Wrap torsion values from 0.0 to 360.0 (default is -180.0 to 180.0).

[dihtype <name>:<a0>:<a1>:<a2>:<a3>[:<offset>] Search for a custom dihedral type called <name> using atom names <a0>, <a1>, <a2>, and <a3>.
Offset: -2=<a0><a1> in previous res, -1=<a0> in previous res, 0=All <aX> in single res, 1=<a3> in next res, 2=<a2><a3> in next res.

DataSet Aspects:

[<dihedral type>]:<#> Aspect corresponds to the dihedral type name (e.g. [phi], [psi], etc). The index is the residue number.

*Note data sets are not generated until **run** is called.*

Calculate specified dihedral angle types for residues in given range. By default, dihedral angles are identified based on standard Amber atom names. The resulting data sets will have aspect equal to [<dihedral type>] and index equal to residue #. To differentiate the chi angle, chip is used for proteins and chin for nucleic acids. For example, to calculate all phi/psi dihedrals for residues 6 to 9:

```
multidihedral MyTorsions phi psi resrange 6-9 out PhiPsi_6-9.dat
```

This will generate data sets named MyTorsions[phi]:6, MyTorsions[psi]:6, MyTorsions[phi]:7, etc. Dihedrals other than those defined in **<dihedral types>** can be searched for using **dihtype**. For example to create a custom dihedral type called chi1 using atoms N, CA, CB, and CG (all in the same residue), then search for and calculate the dihedral in all residues:

```
multidihedral dihtype chi1:N:CA:CB:CG out custom.dat
```

11.51 multivector

```
multivector [<name>] [resrange <range>] name1 <name1> name2 <name2> [out <filename>]
            [ired]
```

[<name>] Data set name.

[resrange <range>] Range of residues to look for vectors in.

name1 <name1> Name of first atom in each residue.

name2 <name2> Name of second atom in each residue.

[out <filename>] File to write results to.

Search for and calculate atomic vectors between atoms named <name1> and <name2> in residues specified by the given <range>; each one is equivalent to the command 'vector <name1> <name2>'. For example, to calculate all vectors between atoms named 'N' and atoms named 'H' in residues 5-20, storing the results in data sets named NH and writing to NH.dat:

```
multivector NH name1 N name2 H ired out NH.dat resrange 5-20
```

11.52 nastruct

```
nastruct [<dataset name>] [resrange <range>] [naout <suffix>]
      [noheader] [resmap <ResName>:{A,C,G,T,U} ...] [calcnohb]
         [baseref <file>] ...
      [hbcut <hbcut>] [origincut <origincut>] [altona | cremer]
      [zcut <zcut>] [zanglecut <zanglecut>] [groovecalc {simple | 3dna}]
      [{ first | reference | ref <name> | refindex <#> | allframes | guessbp}]
         [bptype {anti | para} ...]
```

[<dataset name>] Output data set name.

[resrange <range>] Residue range to search for nucleic acids in (default all).

[naout <suffix>] File name suffix for output files; BP.<suffix> for base pair parameters, BPstep.<suffix> for base pair step parameters, and Helix.<suffix> for base pair step helical parameters.

[noheader] Do not print header to naout file.

[resmap <ResName>:{A,C,G,T,U}] Attempt to treat residues named <ResName> as if it were A, C, G, T, or U; useful for residues with modifications or non-standard residue names. This will only work if enough reference atoms are present in <ResName>.

[calcnohb] Calculate parameters between bases in base pairs even if no hydrogen bonds present between them.

[baseref <file>] Specify a custom nucleic acid base reference. One file per custom residue; multiple 'baseref' keywords may be present. See below for details.

[hbcut <hbcut>] Distance cutoff (in Angstroms) for determining hydrogen bonds between bases (default 3.5).

[origincut <origincut>] Distance cutoff (in Angstroms) between base pair axis origins for determining which bases are eligible for base-pairing (default 2.5).

[altona] Use method of Altona & Sundaralingam to calculate sugar pucker (default, see *pucker* command).

[cremer] Use method of Cremer and Pople to calculate sugar pucker (see *pucker* command).

[zcut] Distance cutoff (in Angstroms) between base reference axes along the Z axis (i.e. stagger) for determining base pairing (default 2).

[zanglecut] Angle cutoff (in degrees) between base reference Z axes for determining base pairing (default 65).

[groovecalc] Groove width calculation method:

simple Use P-P distance for major groove, O4-O4 distance for minor groove. Output to 'BP.<suffix>'.

3dna Use groove width calculation of El Hassan and Calladine[14]. Output to 'BPstep.<suffix>'.

[first] Use first frame to determine base pairing (default).

[reference | refindex <#> | ref <name>] Reference structure to use to determine base pairing.

[allframes] If specified determine base pairing each frame.

[guessbp [bptype{anti|para}]] If specified base pairing will be determined based on selected NA strands. It is assumed that consecutive strands will be base-paired and that they are arranged 5' to 3'. The specific type of base pairing between strands can be specified with one or more 'bptype' arguments.

DataSets Created:

<name>[pucker]:X Base X (residue number) sugar pucker.

Base pairs:

<name>[shear]:X Base pair X (starting from 1) shear.

<name>[stretch]:X Base pair stretch.

<name>[stagger]:X Base pair stagger.

<name>[buckle]:X Base pair buckle.

<name>[prop]:X Base pair propeller.

<name>[open]:X Base pair opening.

<name>[hb]:X Number of WC hydrogen bonds between bases in base pair.

<name>[bp]:X Contain 1 if bases are base paired, 0 otherwise.

<name>[major]:X (If groovecalc simple) Major groove width calculated between P atoms of each base.

<name>[minor]:X (If groovecalc simple) Minor groove width calculated between O4 atoms of each base.

Base pair steps:

<name>[shift]:X Base pair step X (starting from 1) shift.

<name>[slide]:X Base pair step slide.

<name>[rise]:X Base pair step rise.

<name>[title]:X Base pair step tilt.

<name>[roll]:X Base pair step roll.

<name>[twist]:X Base pair step twist.

<name>[zp]:X Base pair step Zp value.

<name>[major]:X (If groovecalc 3dna) Major groove width, El Hassan and Calladine.

<name>[minor]:X (If groovecalc 3dna) Minor groove width, El Hassan and Calladine.

Helical steps:

<name>[xdisp]:X Helical step X (starting from 1) X displacement.

<name>[ydisp]:X Helical Y displacement.

<name>[hrise]:X Helical rise.

<name>[incl]:X Helical inclination.

<name>[tip]:X Helical tip.

<name>[htwist]:X Helical twist.

Note that data sets are not created until base pairing is determined.

Calculate basic nucleic acid (NA) structure parameters for all residues in the range specified by **resrange** (or all NA residues if no range specified). Residue names are recognized with the following priority: standard Amber residue names DA, DG, DC, DT, RA, RG, RC, and RU; 3 letter residue names ADE, GUA,

CYT, THY, and URA; and finally 1 letter residue names A, G, C, T, and U. Non-standard/modified NA bases can be recognized by using the **resmap** keyword. For example, to make *cpptraj* recognize all 8-oxoguanine residues named '8OG' as a guanine-based residue:

```
nastruct naout nastruct.dat resrange 274-305 resmap 8OG:G
```

The **resmap** keyword can be specified multiple times, but only one mapping per unique residue name is allowed. Note that **resmap** may fail if the residue is missing heavy atoms normally present in the specified base type.

Base pairs are determined either once from the first frame or from a reference structure, or can be determined each frame if **allframes** is specified. Base pairing is determined first by base reference axis origin distance, then by stagger, then by angle between base Z axes, then finally by hydrogen bonding (at least one hydrogen bond must be present). Base pair parameters will only be written for determined base pairs. Both Watson-Crick and other types of base pairing can be detected. Note that although all possible hydrogen bonds are searched for, only WC hydrogen bonds are reported in the BP.<suffix> file.

The procedure used to calculate NA structural parameters is the same as 3DNA[15], with algorithms adapted from Babcok et al.[16] and reference frame coordinates from Olson et al.[17]. Given the same base pairs are determined, *cpptraj* nastruct gives the exact same numbers as 3DNA.

Calculated NA structure parameters are written to three separate files, the suffix of which is specified by **naout**. Base pair parameters (shear, stretch, stagger, buckle, propeller twist, opening, # WC hydrogen bonds, base pairing, and simple groove widths) are written to BP.<suffix>, along with the number of WC hydrogen bonds detected. Base pair step parameters (shift, slide, rise, tilt, roll, twist, Zp, and El Hassan and Calladine groove widths) are written to BPstep.<suffix>, and helical parameters (X-displacement, Y-displacement, rise, inclination, tip, and twist) are written to Helix.<suffix>. If **noheader** is specified a header will not be written to the output files. Note that although base puckering is calculated, it is not written to an output file by default. You can output pucker to a file via the create or write/writedata commands after the data has been generated, e.g.:

```
nastruct NA naout nastruct.dat resrange 1-3,28-30
run
writedata NApucker.dat NA[pucker]
```

Custom Nucleic Acid Base References

Users can now specify **baseref** <**file**> to load a custom nucleic acid base reference. The base reference files are white-space delimited, begin with the line NASTRUCT REFERENCE, and have the following format:

```
NASTRUCT REFERENCE
```

```
<base character> <res name 0> [<res name 1> ...]
<atom name> <X> <Y> <Z> <HB type> <RMS fit>
...
```

There is a line for each reference atom. Lines beginning with '#' are ignored as comments.

<**base character**> Used to identify the underlying base type: A G C T or U. If none of these, it will be considered an unknown residue (which just means WC hydrogen bonding will not be identified).

<**res name X**> Specifies what residue names this reference corresponds to. There must be at least one residue name. There can be any number of these specified.

<**atom name**> A reference atom name.

<**X**> <**Y**> <**Z**> The X Y and Z coordinates of the reference atom.

<**HB type**> Denotes if and how the atom participates in hydrogen bonding. Can be 'd'onor, 'a'cceptor, or 'n'one (or the numbers 1, 2, 0 respectively). Only the first character of the word actually matters.

<**RMS fit**> Denotes whether the atom is involved in RMS-fitting.

Here is an example for GUA:

```
NASTRUCT REFERENCE
G G G5 G3
# Modified into format readable by cpptraj nastruct
C1' -2.477   5.399   0.000 0      0
N9  -1.289   4.551   0.000 0      1
C8   0.023   4.962   0.000 0      1
N7   0.870   3.969   0.000 accept 1
C5   0.071   2.833   0.000 0      1
C6   0.424   1.460   0.000 0      1
O6   1.554   0.955   0.000 accept 0
N1  -0.700   0.641   0.000 donor  1
C2  -1.999   1.087   0.000 0      1
N2  -2.949   0.139  -0.001 donor  0
N3  -2.342   2.364   0.001 accept 1
C4  -1.265   3.177   0.000 0      1
```

11.53 nativecontacts

```
nativecontacts [<mask1> [<mask2>]] [writecontacts <outfile>] [resout <resfile>]
               [noimage] [distance <cut>] [out <filename>] [includesolvent]
               [ first | reference | ref <name> | refindex <#> ]
```

[resoffset <n>] [contactpdb <file>] [pdbcut <cut>] [mindist] [maxdist]
[name <dsname>] [byresidue] [map [mapout <mapfile>]] [series [seriesout
[savenonnative [seriesnnout <file>] [nncontactpdb <file>]]
[resseries { present | sum } [resseriesout <file>]] [skipnative]

<mask1> First mask to calculate contacts for.

[<mask2>] (Optional) Second mask to calculate contacts for.

[writecontacts <outfile>] Write information on native contacts to <outfile> (STDOUT if not specified).

[resout <resfile>] File to write contact residue pairs to.

[noimage] Do not image distances.

[distance <cut>] Distance cutoff for determining native contacts in Angstroms (default 7.0 Ang).

[out <filename>] File to write number of native contacts and non-native contacts.

[includesolvent] By default solvent molecules are ignored; this will explicitly include solvent molecules.

[first | reference | ref <name> | refindex <#>] Reference structure to use for determining native contacts.

[resoffset <n>] (byresidue only) Ignore contacts between residues spaced less than <n> residues apart in sequence.

[contactpdb <file>] Write PDB with B-factor column containing relative contact strength for native contacts (strongest is 100.0).

[pdbcut <cut>] If writing contactpdb, only write contacts with relative contact strength greater than <cut>.

[mindist] If specified, determine the minimum distance between any atoms in the mask(s).

[maxdist] If specified, determine the maximum distance between any atoms in the mask(s).

[name <dsname>] Data set name.

[byresidue] Write out the contact map by residue instead of by atom.

[map] Calculate matrices of native contacts ([nativemap]) and non-native contacts ([nonnatmap]). These matrices are normalized by the total number of

frames, so that a value of 1.0 means "contact always present". If byresidue specified, the values for each individual atom pair are summed over the residues they belong to (this means for byresidue values greater than 1.0 are possible).

[mapout <mapfile>] Write native/non-native matrices to 'native.<mapfile>' and 'nonnative.<mapfile>' respectively.

[series] Calculate native contact time series data, 1 for contact present and 0 otherwise.

[seriesout <file>] Write native contact time series data to file.

[savenonnative] Save non-native contacts; series must also be specified. This is enabled by default if skipnative specified.

[seriesnnout <file>] Write non-native contact time series data to file.

[nncontactpdb <file>] Write PDB with B-factor column containing relative contact strength for non-native contacts (strongest is 100.0).

[resseries {present | sum} Create contacts time series by residue; series must also be specified.

present Record a 1 if *any* contact is present and 0 if no contact is present for the residue pair.

sum The sum of all individual contacts is recorded for the residue pair.

[resseriesout <file>] Write residue time series data to <file>.

[skipnative] If specified, skip native contacts determination, i.e. treat all sonctacts as non-native contacts. Implies savenonnative.

Data Sets Created:

<dsname>[native] Number of native contacts.

<dsname>[nonnative] Number of non-native contacts.

<dsname>[mindist] (mindist only) Minimum observed distance each frame.

<dsname>[maxdist] (maxdist only) Maximum observed distance each frame.

<dsname>[nativemap] (map only) Native contacts matrix (2D).

<dsname>[nonnatmap] Non-native contacts matrix (2D).

<**dsname**>**[NC]** `Native contacts time series.`

<**dsname**>**[NN]** `Non-native contacts time series.`

<**dsname**>**[NCRES]** `Residue native contacts time series.`

<**dsname**>**[NNRES]** `Residue non-native contacts time series.`

Define and track "native" contacts as determined by a simple distance cut-off, i.e. any atoms which are closer than <**cut**> in the specified reference frame (the first frame if no reference specified) are considered a native contact. If one mask is provided, contacts are looked for within <**mask1**>; if two masks are provided, only contacts between atoms in <**mask1**> and atoms in <**mask2**> are looked for (useful for determining intermolecular contacts). By default only native contacts are tracked. This can be changed by specifying the **savenonnative** keyword or by specifying **skipnative**. The time series for contacts can be saved using the **series** keyword; these can be further consolidated by residue using the **resseries** keyword. When using <resseries> the data set index is calculated as (r2 * nres) + r1 so that indices can be matched between native/non-native contact pairs. Non-native residue contact legends have an nn_ prefix.

Native contacts that are found are written to the file specified by writecontacts (or STDOUT) with format:

```
# Contact Nframes Frac. Avg Stdev
```

Where `Contact` takes the form ':<residue1 num>@<atom name>_:<residue2 num>@<atom name>, `Nframes` is the number of frames the contact is present, `Frac.` is the total fraction of frames the contact is present, `Avg` is the average distance of the contact when present, and `Stdev` is the standard deviation of the contact distance when present. If **resout** is specified the total fraction of contacts is printed for all residue pairs having native contacts with format:

```
#Res1 #Res2 TotalFrac Contacts
```

Where `#Res1` is the first residue number, `#Res2` is the second residue number, `TotalFrac` is the total fraction of contacts for the residue pair, and `Contacts` is the total number of native contacts involved with the residue pair. Since `TotalFrac` is calculated for each pair as the sum of each contact involving that pair divided by the total number of frames, it is possible to have `TotalFrac` values greater than 1 if the residue pair includes more than 1 native contact.

During trajectory processing, non-native contacts (i.e. any pair satisfying the distance cut-off which is not already a native contact) are also searched for. The time series for native contacts can be saved as well, with 1 for contact present and 0 otherwise (similar to the ***hbond*** command). This data can be subsequently analyzed using e.g. 12.19 on page 191.

Contact maps (matrices) are generated for native and non-native contacts. If **byresidue** is specified, contact maps are summed over residues, and contacts between residues spaced <**resoffset**> residues apart in sequence are ignored.

If **contactpdb** is specified a PDB is generated containing relative contact strengths in the B-factor column. The relative contact strength is normalized so that a value of 100 means that atom participated in the most contacts with other atoms.

Example command looking for contacts between residues 210 to 260 and residue named NDP, using reference structure 'FtuFabI.WT.pdb' to define native contacts:

```
parm FtuFabI.parm7
trajin FtuFabI.nc
reference FtuFabI.WT.pdb
nativecontacts name NC1 :210-260&!@H= :NDP&!@H= \
    byresidue out nc.all.res.dat mindist maxdist \
    distance 3.0 reference map mapout resmap.gnu \
    contactpdb Loop-NDP.pdb \
    series seriesout native.dat
```

11.54 outtraj

```
outtraj <filename> [ trajout args ]
        [maxmin <dataset> min <min> max <max>] ...
```

<**filename**> Output trajectory file name.

[trajout args] Output trajectory arguments (see 10.5 on page 63).

[maxmin <dataset> min <min> max <max>] Only write frames to <filename> if values in <dataset> for those frames are between <min> and <max>. Can be specified for one or more data sets.

The outtraj command is similar in function to ***trajout***, and takes all of the same arguments. However, instead of writing a trajectory frame after all actions are complete outtraj writes the trajectory frame at its position in the Action queue. For example, given the input:

```
trajin mdcrd.crd
trajout output.crd
outtraj BeforeRmsd.crd
rms R1 first :1-20@CA out rmsd.dat
outtraj AfterRmsd.crd
```

three trajectories will be written: output.crd, BeforeRmsd.crd, and AfterRmsd.crd. The output.crd and AfterRmsd.crd trajectories will be identical, but the BeforeRmsd.crd trajectory will contain the coordinates of mdcrd.crd before they are RMS-fit.

The maxmin keyword can be used to restrict output using one more more data sets. For example, to only write frames for which the RMSD value is between 0.7 and 0.8:

```
trajin tz2.truncoct.nc
rms R1 first :2-11
outtraj maxmin.crd maxmin R1 min 0.7 max 0.8
```

11.55 pairdist

```
pairdist out <filename> mask <mask> [delta <resolution>]
```

Calculate pair distribution function. In the following, defaults are given in parentheses. The **out** keyword specifies output file for histogram: distance, P(r), s(P(r)). The **mask** option specifies atoms for which distances should be computed. The **delta** option specifies resolution. (0.1 Å)

11.56 pairwise

```
pairwise [<name>] [<mask>] [out <filename>] [cuteelec <ecut>] [cutevdw <vcut>]
         [ reference | ref <name> | refindex <#> ] [cutout <cut mol2 prefix>]
      [vmapout <vdw map>] [emapout <elec map>] [avgout <avg file>]
      [eout <eout file>] [pdbout <pdb file>] [scalepdbe] [printmode {only|or|and|}]
```

[<name>] Data set name; van der Waals energy will get aspect [EVDW] and electrostatic energy will get aspect [EELEC].

[<mask>] Atoms to calculate energy for.

[out <filename>] File to write total EELEC and EVDW to.

[eout <eout file>] File to write individual EELEC and EVDW interactions to.

[reference | ref <name> | refindex <#>] Specify a reference to compare frames to (i.e. calculate Eref - Eframe).

[cuteelec <cut>] Only report interaction EELEC (or delta EELEC) if absolute value is greater than <ecut> (default 1.0 kcal/mol).

[cutevdw <cutv>] Only report interaction EVDW (or delta EVDW) if absolute value is greater than <vcut> (default 1.0 kcal/mol).

[cutout <cut mol2 prefix>] Write out mol2 containing only atom pairs which satisfy <ecut> and <vcut>.

[vmapout <vdw map>] Write out interaction EVDW (or delta EVDW) matrix to file <vdw map>.

[emapout <elec map>] Write out interaction EELEC (or delta EELEC) matrix to file <elec map>.

[avgout <avg file>] Print average interaction EVDW|EELEC (or average delta EVDW|EELC) to <avg file>.

[pdbout <pdb file>] Write PDB with EVDW|EELEC in occupancy|B-factor columns to <pdb file>.

[scalepdbe] Scale energies written to PDB from 0 to 100.

[printmode {only|or|and}] Control when/how average energies are written

Data Sets Created:

<name>[EELEC] Electrostatic energy in (kcal/mol).

<name>[EVDW] van der Waals energy in (kcal/mol).

<name>[VMAP] van der Waals energy matrix.

<name>[EMAP] Electrostatic energy matrix.

This action has two related functions: 1) Calculate pairwise (i.e. non-bonded) energy (in kcal/mol) for atoms in **<mask>**, or 2) Compare pairwise energy of frames to a reference frame. This calculation does use an exclusion list but is not periodic.

When comparing to a reference frame, the **eout** file will contain the differences for each individual interaction (i.e. Eref - Eframe), otherwise the **eout** file will contain the absolute value of each individual interaction. The **cuteelc** and **cutevdw** keywords can be used to restrict printing of individual interactions to those for which the absolute value is above a cutoff. The VMAP and EMAP matrix elements will contain these values as well (differences for reference, absolute value otherwise) averaged over all frames. The **avgout** file will contain only these values averaged over all frames that satisfy the cutoffs. The **printmode** keyword controls when the average energies are written: **only** means only average energy components that satisfy cutoffs will be printed, **or** means that both energy components will be printed if either satistfy a cutoff, and **and** means that both energy components will be written only if both satisfy the cutoffs.

The **cutout** keyword can be used to write out MOL2 files each frame named '<cut mol2 prefix>.evdw.mol2.X' and '<cut mol2 prefix>.eelec.mol2.X' (where X is the frame number) containing only atoms with energies that satisfy the cutoffs. Similarly, the **pdbout** keyword can be used to write out a PDB file (with 1 MODEL per frame). The occupancy and B-factor columns will contain the total van der Waals and electrostatic energy for each atom if cutoffs are satisfied, or 0.0 otherwise.

11.57 principal

```
principal [<mask>] [dorotation] [out <filename>] [name <dsname>]
```

[<mask>] Mask of atoms used to determine principal axes (default all).

[dorotation] Align coordinates along principal axes.

[out <filename>] Write resulting eigenvalues/eigenvectors to <filename>.

[name <dsname>] Data set name (3x3 matrices).

Data Sets Created (name keyword only):

<dsname>[evec] Eigenvectors (3x3 matrix, row-major).

<dsname>[eval] Eigenvalues (vector).

Determine principal axes of each frame determined by diagonalization of the inertial matrix from the coordinates of the specified atoms. At least one of **dorotation**, **out**, or **name** must be specified. The resulting eigenvectors are sorted from largest eigenvalue to smallest, and the corresponding axes labelled using the *cpptraj* convention of X > Y > Z (similar to '***vector* principal**'). If out is specified the eigenvectors and eigenvalues will be written for each frame N with format:

```
<N> EIGENVALUES: <EX> <EY> <EZ>
<N> EIGENVECTOR 0: <Xx> <Xy> <Xz>
<N> EIGENVECTOR 1: <Yx> <Yy> <Yz>
<N> EIGENVECTOR 2: <Zx> <Zy> <Zz>
```

NOTE: The eigenvector 3x3 matrix data set could subsequently be used e.g. with the ***rotate*** action.

Example: Align system (residues 1-76) along principle axes:

```
parm myparm.parm7
trajin protein.nc
principal :1-76 dorotation out principal.dat
```

11.58 projection

```
projection [<name>] evecs <dataset name> [out <outfile>] [beg <beg>] [end <end>]
           [<mask>] [dihedrals <dataset arg>]
           [start <start>] [stop <stop>] [offset <offset>]
```

[<name>] Output data set name.

evecs <dataset name> Data set containing eigenvectors (modes).

[out <outfile>] Write projections to <outfile>.

[beg <beg>] First eigenvector/mode to use (default 1).

[end <end>] Final eigenvector/mode to use (default 2).

[<mask>] (Not dihedral covariance) Mask of atoms to use in projection; MUST CORRESPOND TO HOW EIGENVECTORS WERE GENERATED.

[dihedrals <dataset arg>] (Dihedral covariance only) Dihedral data sets to use in projection; MUST CORRESPOND TO HOW EIGENVECTORS WERE GENERATED.

[start <start>] Frame to start calculating projection.

[stop <stop>] Frame to stop calculating projection.

[offset <offset>] Frames to skip between projection calculations.

Data Sets Created:
DataSet indices correspond to mode #.

<name> (All execpt IDEA) Projection data set.

<name>[X] X component of mode (IDEA modes only).

<name>[Y] Y component of mode (IDEA modes only).

<name>[Z] Z component of mode (IDEA modes only).

<name>[R] Magnitude of mode (IDEA modes only).

Projects snapshots onto eigenvectors obtained by diagonalizing covariance or mass-weighted covariance matrices. Eigenvectors are taken from previously generated (e.g. with ***diagmatrix***) or previously read-in (e.g. with ***readdata***) eigenvectors with name <dataset name>. The user has to make sure that the atoms selected by **<mask>** agree with the ones used to calculate the modes (i.e., if mask = '@CA' was used in the "**matrix**" command, mask = '@CA' needs to be set here as well). See 13 on page 217 for examples using the ***projection*** command.

11.59 pucker

```
pucker [<name>] <mask1> <mask2> <mask3> <mask4> <mask5> [<mask6>] [geom]
   [out <filename>] [altona | cremer] [amplitude] [theta]
    [range360] [offset <offset>]
```

<name> Output data set name.

<maskX> Five (optionally six) atom masks selecting atom(s) to calculate pucker for.

[geom] Use geometric center of atoms in <maskX> (default is center of mass).

[out <filename>] Output file name.

[altona] Use method of Altona & Sundaralingam (5 masks only).

[cremer] Use method of Cremer and Pople (5 or 6 masks). This is the default when 6 masks are specified.

[amplitude] Also calculate amplitude.

[theta] (6 masks only) Also calculate theta.

[range360] Wrap pucker values from 0.0 to 360.0 (default is -180.0 to 180.0).

[offset <offset>] Add <offset> to pucker values.

Data Sets Created:

<name> Pucker in degrees.

<name>[Amp] Amplitude (if amplitude was specified).

<name>[Theta] Theta (if theta and 6 masks were specified).

Calculate the pucker (in degrees) for atoms in <mask1>, <mask2>, <mask3>, <mask4>, <mask5> using the method of Altona & Sundarlingam[18, 19] (default for 5 masks, or if **altona** specified), or the method of Cremer & Pople[20] (default for 6 masks, or if **cremer** is specified). If the **amplitude** or **theta** keywords are given, amplitudes/thetas will be calculated in addition to pucker. The results from ***pucker*** can be further analyzed with the ***statistics*** analysis.

By default, pucker values are wrapped to range from -180 to 180 degrees. If the **range360** keyword is specified values will be wrapped to range from 0 to 360 degrees. Note that the Cremer & Pople convention is offset from Altona & Sundarlingam convention (with nucleic acids) by +90.0 degrees; the **offset** keyword will add an offset to the final value and so can be used to convert between the two. For example, to convert from Cremer to Altona specify "**offset** 90".

To calculate nucleic acid pucker specify C1' first, followed by C2', C3', C4' and O4'. For example, to calculate the sugar pucker for nucleic acid residues 1 and 2 using the method of Altona & Sundarlingam, with final pseudorotation values ranging from 0 to 360:

```
pucker p1 :1@C1' :1@C2' :1@C3' :1@C4' :1@O4' range360 out pucker.dat
pucker p2 :2@C1' :2@C2' :2@C3' :2@C4' :2@O4' range360 out pucker.dat
```

11.60 radgyr | rog

```
radgyr [name>] [<mask>] [out <filename>] [mass] [nomax] [tensor]
```

[<name>] Data set name.

[<mask>] Atoms to calculate radius of gyration for; default all atoms.

[out <filename>] Write data to <filename>.

[mass] Mass-weight radius of gyration.

[nomax] Do not calculate maximum radius of gyration.

[tensor] Calculate radius of gyration tensor, output format 'XX YY ZZ XY XZ YZ'.

Data Sets Created:

<**name**> Radius of gyration in Ang.

<**name**>**[Max]** Max radius of gyration in Ang.

<**name**>**[Tensor]** Radius of gyration tensor; format 'XX YY ZZ XY XZ YZ'.

Calculate the radius of gyration of specified atoms. For example, to calculate only the mass-weighted radius of gyration (not the maximum) of the non-hydrogen atoms of residues 4 to 10 and print the results to "RoG.dat":

```
radgyr :4-10&!(@H=) out RoG.dat mass nomax
```

11.61 radial | rdf

```
radial [out <outfilename>] <spacing> <maximum> <solvent mask1>
       [<solute mask2>] [noimage] [density <density> | volume] [<dataset name>]
       [intrdf <file>] [rawrdf <file>]
       [{{center1|center2|nointramol} | [byres1] [byres2] [bymol1] [bymol2]}]]
```

[out <outfilename>] File to write RDF to.

<**spacing**> Bin spacing, required.

<**maximum**> Max bin value, required.

<**solvent mask1**> Atoms to calculate RDF for, required.

[<solute mask2>] (Optional) If specified calculate RDF of all atoms in <solvent mask1> to each atom in <solute mask2>.

[noimage] Do not image distances.

[density <density>] Use density value of <density> for normalization (default 0.033456 molecules $Å^{-3}$).

[volume] Determine density for normalization from average volume of input frames.

[<setname>] Name of output data sets.

[center1] Calculate RDF from geometric center of atoms in <solvent mask1> to all atoms in <solute mask2>.

[center2] Calculate RDF from geometric center of atoms in <solute mask2> to all atoms in <solvent mask1>.

[nointramol] Ignore intra-molecular distances.

[intrdf <file>] Calculate integral of RDF bin values (averaged over # of frames but otherwise not normalized) and write to <file> (can be same as <output_filename>).

[rawrdf <file>] Write raw (non-normalized) RDF values to <file>.

[byres1] Calculate using the centers of mass of each residue in the first mask.

[bymol1] Calculate using the centers of mass of each molecule in the first mask.

[byres2] Calculate using the centers of mass of each residue in the second mask.

[bymol2] Calculate using the centers of mass of each molecule in the second mask.

DataSet Aspects:

<**setname**> The radial distribution function.

<**setname**>**[int]** (intrdf only) Integral of RDF bin values.

<**setname**>**[raw]** (rawrdf only) Raw (non-normalized) RDF values.

Calculate the radial distribution function (RDF, aka pair correlation function) of atoms in <**solvent mask1**> (note that this mask does not need to be solvent, but this nomenclature is used for clarity). If an optional second mask (<**solute mask2**>) is given, calculate the RDF of ALL atoms in <**solvent mask1**> to EACH atom in <**solute mask2**>. If desired, the geometric center of atoms in <**solvent mask1**> or <**solute mask2**> can be used by specifying the **center1** or **center2** keywords respectively, or alternatively intra-molecular distances can be ignored by specifying the **nointramol** keyword.

The RDF is calculated from the histogram of the number of particles found as a function of distance R, normalized by the expected number of particles at that distance. The normalization is calculated from:

$$Density * \frac{4\pi}{3}\left((R+dR)^3 - R^3\right)$$

where dR is equal to the bin spacing. Some care is required by the user in order to normalize the RDF correctly. The default density value is 0.033456 molecules Å^{-3}, which corresponds to a density of water approximately equal to 1.0 g mL^{-1}. To convert a standard density in g mL^{-1}, multiply the density by $\frac{0.6022}{M_r}$, where M_r is the mass of the molecule in atomic mass units. Alternatively, if the **volume** keyword is specified the density is determined from the average volume of the system over all Frames.

Note that correct normalization of the RDF depends on the number of atoms in each mask; if multiple topology files are being processed that result in changes in the number of atoms in each mask, the normalization will be off.

11.62 randomizeions

```
randomizeions <mask> [around <mask> by <distance>] [overlap <value>]
              [noimage] [seed <value>]
```

This can be used to randomly swap the positions of solvent and single atom ions. The "`overlap`" specifies the minimum distance between ions, and the "`around`" keyword can be used to specify a solute (or set of atoms) around which the ions can get no closer than the distance specified. The optional keywords "`noimage`" disable imaging and "`seed`" update the random number seed. An example usage is

```
randomizeions @NA around :1-20 by 5.0 overlap 3.0
```

The above will swap Na^+ ions with water getting no closer than 5.0 Å from residues $1-20$ and no closer than 3.0 Å from any other Na^+ ion.

11.63 replicatecell

```
replicatecell [out <traj filename>] [parmout <parm filename>] [name <dsname>]
              { all | dir <XYZ> [dir <XYZ> ...] } [<mask>]
```

out <**traj filename**> `Write replicated cell to output trajectory file.`

parmout <**parm filename**> `Write replicated cell topology to topology file. This file will not be viable to use for simulations.`

name <**dsname**> `If specified save replicated cell to COORDS data set.`

all `Replicate cell once in all possible directions.`

dir <**XYZ**> `Repicate cell once in specified directions. <XYZ> should consist of 3 numbers with no spaces in between them and are restricted to values of -1, 1, and 0. May be specified more than once.`

<**mask**> `Mask of atoms to replicate.`

Create a trajectory where the unit cell is replicated in 1 or more directions (up to 27). The resulting coordinates and topology can be written to a trajectory/topology file. They can also be saved as a COORDS data set for subsequent processing. Currently replication is only allowed 1 axis length in either

direction. The **all** keyword will replicate the cell once in all directions. The **dir** keyword can be used to restrict replication to specific directions, e.g. 'dir 10-1' would replicate the cell once in the +X, -Z directions.

For example, to replicate a cell in all directions, writing out to NetCDF trajectory cell.nc:

```
parm ../tz2.truncoct.parm7
trajin ../tz2.truncoct.nc
replicatecell out cell.nc parmout cell.parm7 all
```

11.64 rms | rmsd

```
rmsd [<name>] <mask> [<refmask>] [out <filename>] [mass]
  [nofit | norotate | nomod]
  [savematrices [matricesout <file>]]
  [savevectors {combined|separate} [vecsout <file>]]
  [ first | reference | ref <name> | refindex <#> | previous |
    reftraj <name> [parm <name> | parmindex <#>] ]
  [perres perresout <filename> [perresavg <avgfile>]
   [range <resRange>] [refrange <refRange>]
   [perresmask <additional mask>] [perrescenter] [perresinvert]
```

[<name>] Output data set name.

[<mask>] Mask of atoms to calculate RMSD for; if not specified, calculate for all atoms.

[<refmask>] Reference mask; if not specified, use <mask>.

[out <filename>] Output data file name.

[mass] Mass-weight the RMSD calculation.

[nofit] Do not perform best-fit RMSD.

[norotate] If calculating best-fit RMSD, translate but do not rotate coordinates.

[nomod] If calculating best-fit RMSD, do not modify coordinates.

[savematrices] If specified save rotation matrices to data set with aspect [RM].

matricesout <file> Write rotation matrices to specified file.

[savevectors {combined|separate}] If specified save translation vectors: combined means save target-to-origin plus the origin-to-reference translation vectors, separate means save target-to-origin as Vx, Vy, Vz and save origin-to-reference as Ox Oy Oz in the output vector data set.

vecsout <**file**> Output translation vector data set to <file>.

Reference keywords:

first Use the first trajectory frame processed as reference.

reference Use the first previously read in reference structure (refindex 0).

ref <**name**> Use previously read in reference structure specified by filename/tag.

refindex <#> Use previously read in reference structure specified by <#> (based on order read in).

previous Use frame prior to current frame as reference.

reftraj <**name**> Use frames from COORDS set <name> or read in from trajectory file <name> as references. Each frame from <name> is used in turn, so that frame 1 is compared to frame 1 from <name>, frame 2 is compared to frame 2 from <name> and so on. If <trajname> runs out of frames before processing is complete, the last frame of <trajname> continues to be used as the reference.

parm <**parmname**> | **parmindex** <#> If reftraj specifies a trajectory file, associate it with specified topology; if not specified the first topology is used.

Per-residue RMSD keywords:

perres Activate per-residue no-fit RMSD calculation.

perresout <**perresfile**> Write per-residue RMSD to <perresfile>.

perresavg <**avgfile**> Write average per-residue RMSDs to <avgfile>.

range <**res range**> Calculate per-residue RMSDs for residues in <res range> (default all solute residues).

refrange <**ref range**> Calculate per-residue RMSDs to reference residues in <ref range> (use <res range> if not specified).

perresmask <**additional mask**> By default residues are selected using the mask ':X' where X is residue number; this appends <additional mask> to the mask expression.

perrescenter Translate residues to a common center of mass prior to calculating RMSD.

perresinvert Make X-axis residue number instead of frame number.

Data Sets Created:

<**name**> RMSD of atoms in mask to reference.

<**name**>**[RM]** (savematrices only) Rotation matrices of target to reference.

<**name**>**[TV]** (savevectors only) Translation vector.

<**name**>**[res]** (perres only) Per-residue RMSDs; index is residue number.

<**name**>**[Avg]** (perres only) Average per-residue RMSD for each residue.

<**name**>**[Stdev]** (perres only) Standard deviation of RMSD for each residue.

*Note that **perres** data sets are not generated until **run** is called.*

Calculate the coordinate RMSD of input frames to a reference frame (or reference trajectory). Both <mask> and <refmask> must specify the same number of atoms, otherwise an error will occur.

For example, say you have a trajectory and you want to calculate RMSD to two separate reference structures. To calculate the best-fit RMSD of the C, CA, and N atoms of residues 1 to 20 in each frame to the C, CA, and N atoms of residues 3 to 23 in StructX.crd, and then calculate the no-fit RMSD of residue 7 to residue 7 in another structure named Struct-begin.rst7, writing both results to Grace-format file "rmsd1.agr":

```
reference StructX.crd [structX]
reference md_begin.rst7 [struct0]
rmsd BB :1-20@C,CA,N ref [structX] :3-23@C,CA,N out rmsd1.agr
rmsd Res7 :7 ref [struct0] out rmsd1.agr nofit
```

Per-residue RMSD calculation

If the **perres** keyword is specified, after the initial RMSD calculation the no-fit RMSD of specified residues is also calculated. So for example:

```
rmsd :10-260 reference perres perresout PRMS.dat range 190-211 perresmask &!(@H=)
```

will first perform a best-fit RMSD calculation to the first specified reference structure using residues 10 to 260, then calculate the no-fit RMSD of residues 190 to 211 (excluding any hydrogen atoms), writing the results to PRMS.dat. Two additional recommendations for the 'perres' option: 1) try not including backbone atoms by using the 'perresmask' keyword, e.g. "perresmask &!@H,N,CA,HA,C,O", and 2) try using the 'perrescenter' keyword, which centers each residue prior to the 'nofit' calculation; this is useful for isolating changes in residue conformation.

11.65 rms2d | 2drms

Although the *'rms2d'* command can still be specified as an action, it is now considered an analysis. See 12.28 on page 201.

11.66 rmsavgcorr

Although the *'rmsavgcorr'* command can still be specified as an action, it is now considered an analysis. See 12.29 on page 203.

11.67 rmsf | atomicfluct

See 11.5 on page 74.

11.68 rotate

```
rotate [<mask>] { [x <xdeg>] [y <ydeg>] [z <zdeg>]  |
               axis0 <mask0> axis1 <mask1> <deg> |
               usedata <set name> [inverse] }
```

[<mask>] Rotate atoms in <mask> (default all).

[x <xdeg] Degrees to rotate around the X axis.

[y <xdeg] Degrees to rotate around the Y axis.

[z <xdeg] Degrees to rotate around the Z axis.

axis0 <mask0> Mask defining the beginning of a user-defined axis.

axis1 <mask1> Mask defining the end of a user-defined axis.

<deg> Value in degrees to rotate around user defined axis.

usedata <set name> If specified, use 3x3 rotation matrices in specified data set to rotate coordinates.

[inverse] Perform inverse rotation from input rotation matrices.

Rotate specified atoms around the X, Y, and/or Z axes by the specified amounts, around a user-defined axis (specified by <mask0> and <mask1>), or use a previously read in or generated data set of 3x3 matrices to perform rotations.

For example, to rotate the entire system 90 degrees around the X axis:

```
rotate x 90
```

To rotate residue 270 90 degrees around the axis defined between atoms C1, C2, C3, C4, C5, and C6 in residue 270 and atoms C7, C8, C9, C10, C11, and C12 in residue 270:

```
rotate :270 axis0 :270@C1,C2,C3,C4,C5,C6 axis1 :270@C7,C8,C9,C10,C11,C12 90.0
```

To rotate the system with rotation matrices read in from rmatrices.dat:

```
trajin tz2.norotate.crd
readdata rmatrices.dat name RM mat3x3
rotate usedata RM
```

11.69 rotdif

The ***'rotdif'*** command is now an analysis (see 12.30 on page 204), and requires that rotation matrices be generated via an ***rmsd*** action. For example:

```
reference avgstruct.pdb
trajin tz2.nc
rms R0 reference @CA,C,N,O savematrices
rotdif rmatrix R0[RM] rseed 1 nvecs 10 dt 0.002 tf 0.190 \
       itmax 500 tol 0.000001 d0 0.03 order 2 rvecout rvecs.dat \
       rmout matrices.dat deffout deffs.dat outfile rotdif.out
```

11.70 runavg | runningaverage

```
runavg [window <window_size>]
```

Note that for backwards compatibility with ptraj "runningaverage" is also accepted.

Replaces the current frame with a running average over a number of frames specified by **window** <window_size> (5 if not specified). This means that in order to build up the correct number of frames to calculate the average, the first <window_size> minus one frames will not be processed by subsequent actions. So for example given the input:

```
runavg window 3
rms first out rmsd.dat
```

the rms command will not take effect until frame 3 since that is the first time 3 frames are available for averaging (1, 2, and 3). The next frame processed would be an average of frames 2, 3, and 4, etc.

11.71 scale

```
scale x <sx> y <sy> z <sz> <mask>
```

Scale the X|Y|Z coordinates of atoms in <mask> by <sx>|<sy>|<sz>.

11.72 secstruct

```
secstruct [<name>] [out <filename>] [<mask>] [sumout <filename>]
          [assignout <filename>] [totalout <filename> [ptrajformat]
          [namen <N name>] [nameh <H name>] [nameca <CA name>]
          [namec <C name>] [nameo <O name>]
```

[<name>] Output data set name.

[out <filename>] Output file name for secondary structure vs time.

[<mask>] Atom mask in which residues should be looked for.

[sumout <sumfilename>] Write average secondary structure values for each residue to <sumfilename>; if not specified <filename>.sum is used.

[assignout <filename>] Write overall secondary structure assignment (based on dominant secondary structure type for each residue) to file.

[ptrajformat] Write secondary structure as a string of characters for each frame, similar to ptraj output.

[namen <N name>] Backbone amide nitrogen atom name (default 'N').

[nameh <H name>] Backbone amide hydrogen atom name (default 'H').

[nameca <CA name>] Backbone alpha carbon atom name (default 'CA').

[namec <C name>] Backbone carbonyl carbon atom name (default 'C').

[nameo <O name>] Backbone carbonyl oxygen atom name (default 'O').

Data Sets Created:

<name>[res] Residue secondary structure per frame; index corresponds to residue number. If ptrajformat specified these will be characters, otherwise integers (see table below).

<name>[avgss] Average of each type of secondary structure; index corresponds to secondary structure type (see table below; no index for "None").

<name>[None] Total fraction of residues with no structure vs time.

<**name**>**[Para]** `Total fraction of residues with parallel beta structure vs time.`

<**name**>**[Anti]** `Total fraction of residues with anti-parallel beta structure vs time.`

<**name**>**[3-10]** `Total fraction of 3-10 helical structure vs time.`

<**name**>**[Alpha]** `Total fraction of alpha helical structure vs time.`

<**name**>**[Pi]** `Total fraction of Pi helical structure vs time.`

<**name**>**[Turn]** `Total fraction of turn structure vs time.`

<**name**>**[Bend]** `Total fraction of bend structure vs time.`

Note that when not using ***ptrajformat****, data sets are not generated until* ***run*** *is called.*

Calculate secondary structural propensities for residues in <**mask**> (or all solute residues if no mask given) using the DSSP method of Kabsch and Sander[21], which assigns secondary structure types for residues based on backbone amide (N-H) and carbonyl (C=O) atom positions. By default *cpptraj* assumes these atoms are named "N", "H", "C", and "O" respectively. If a different naming scheme is used (e.g. amide hydrogens are named "HN") the backbone atom names can be customized with the **nameX** keywords (e.g. 'nameH HN'). Note that it is expected that some residues will not have all of these atoms (such as proline); in this case *cpptraj* will print an informational message but the calculation will proceed normally.

Results will be written to filename specified by **out** with format:

```
<#Frame>    <ResX SS> <ResX+1 SS> ... <ResN SS>
```

where <#Frame> is the frame number and <ResX SS> is an integer representing the calculated secondary structure type for residue X. If the keyword **ptrajformat** is specified, the output format will instead be:

```
<#Frame>    STRING
```

where STRING is a string of characters (one for each residue) where each character represents a different structural type (this format is similar to what *ptraj* outputs). The various secondary structure types and their corresponding integer/character are listed below:

Character	Integer	DSSP Char	SS type
0	0	' '	None
b	1	'E'	Parallel Beta-sheet
B	2	'B'	Anti-parallel Beta-sheet
G	3	'G'	3-10 helix
H	4	'H'	Alpha helix
I	5	'I'	Pi (3-14) helix
T	6	'T'	Turn
S	7	'S'	Bend

Average structural propensities over all frames for each residue will be written to the file specified by **sumout** (or "*<filename>.sum*" if **sumout** is not specified). The total structural propensity over all residues for each secondary structure type will be written to the file specified by **totalout**. If assignout is specified, the overall secondary structure assignment for each residue will be printed in two line chunks of 50 residues, with the first line containing the residue number the line starts with and one character residue names, and the second line containing secondary structure assignment using DSSP-style characters, like so:

```
1 KCNTATCATQ RLANFLVHSS NNFGAILSST NVGSNTRn
    SSS   TH HHHTTSBBBB TTTBBBB SS     S
```

The output of secstruct command is amenable to visualization with gnuplot. To generate a 2D map-style plot of secondary structure vs time, with each residue on the Y axis simply give the output file a ".gnu" extension. For example, to generate a 2D map of secondary structure vs time, with different colors representing different secondary structure types for residues 1-22:

```
secstruct :1-22 out dssp.gnu
```

The resulting file can be visualized with gnuplot:

```
gnuplot dssp.gnu
```

Similarly, the **sumout** file can be nicely visualized using xmgrace (use ".agr" extension).

```
secstruct :1-22 out dssp.gnu sumout dssp.agr
xmgrace dssp.agr
```

11.73 spam

```
spam <filename> [solv <solvname>] [reorder] [name <name>]
     [purewater] [cut <cut>] [info <infofile>] [summary <summary>]
     [site_size <size>] [sphere] [out <datafile>]
     [dgbulk <dgbulk>] [dhbulk <dhbulk>] [temperature <T>]
```

<**filename**> File with the peak locations present (XYZ-format)

<**solvname**> Name of the solvent residues

<**cut**> Non-bonded cutoff for energy evaluation

<**dgbulk**> SPAM free energy of the bulk solvent in kcal/mol; default is -30.3 kcal/mol (SPC/E water).

<**dhbulk**> SPAM enthalpy of the bulk solvent in kcal/mol; default is -22.2 kcal/mol (SPC/E water).

<**T**> Temperature at which SPAM calculation was run.

<**infofile**> File with stats about which sites are occupied when.

<**size**> Size of the water site around each density peak.

[sphere] Treat each site like a sphere.

[purewater] The system is pure water---used to parametrize the bulk values.

[reorder] The solvent should be re-ordered so the same solvent molecule is always in the same site.

<**summary**> File with the summary of all SPAM results. If not specified, no SPAM energies will be calculated.

<**datafile**> Data file with all SPAM energies for each snapshot.

Perform profiling of bound water molecules via SPAM analysis[22]. Briefly, this method identifies and estimates the free energy profiles of bound waters via calculation of the distribution of interaction energies between the water and it's environment from explicit solvent MD trajectories. The interaction energies are calculated using a force- and energy-shifted electrostatic term with a hard cutoff.

Prior to this command, the ***volmap*** command should be run with the **peakfile** keyword (see 11.86 on page 160) to generate the peaks file. If not using peaks from the ***volmap*** command, the peaks file should have one line per peak with format:

```
C <X> <Y> <Z> <Density>
```

Values of **dgbulk** and **dhbulk** for different water models can be calculated from pure water simulations with the **purewater** keyword.

11.74 setvelocity

```
setvelocity [<mask>]
            [{ tempi <temperature> |
```

```
                scale [factor <fac>] [sx <xfac>] [sy <yfac>] [sz <zfac>] |
                none |
                modify}]
             [[ntc <#>]] [[dt <time>] [epsilon <eps>]]
          [zeromomentum] [ig <random seed>]
```

<mask> Mask of atoms to assign velocities to.

tempi <temperature> Assign velocities at specified temperature (default 300.0 K).

scale Scale existing velocities

[factor <fac>] Factor to scale velocities by.

[sx <xfac>] Factor to scale X component of velocities by.

[sy <yfac>] Factor to scale Y component of velocities by.

[sz <zfac>] Factor to scale Z component of velocities by.

none Remove any velocities.

modify If specified, do not set, just modify any existing velocities (via 'ntc' or 'zeromomentum').

ig <random seed> Random seed to use to generate velocity distribution.

ntc <#> Correct set velocities for SHAKE constraints. Numbers match sander/pmemd: 1 = no SHAKE, 2 = SHAKE on hydrogens, 3 = SHAKE on all atoms.

dt <time> Time step for SHAKE correction.

epsilon <eps> Epsilon for SHAKE correction

zeromomentum If specified adjust velocities so the total momentum of atoms in <mask> is zero.

Set velocities in frame for atoms in <mask> using Maxwellian distribution based on given temperature.

11.75 stfcdiffusion

```
stfcdiffusion mask <mask> [out <file>] [time <time per frame>]
              [mask2 <mask> [lower <distance>] [upper <distance>]]
              [nwout <file>]) [avout <file>] [distances] [com]
              [x|y|z|xy|xz|yz|xyz]
```

mask Atoms for which MSDs will be computed.

out Output file: time vs. MSD.

time Time step in the trajectory. (1.0 ps)

mask2 Compute MSDs only within the lower and upper limit of mask2. IMPORTANT: may be very slow!!!

lower Smaller distance from reference point(s). (0.01 Å)

upper Larger distance from reference point(s). (3.5 Å)

nwout Output file containing number of water molecules in the chosen region, see mask2. (off)

avout Output file containing average distances. (off)

x|y|z|xy|xz|yz|xyz Computation of the mean square displacement in the chosen dimension. (xyz)

distances Dump un-imaged distances. By default only averages are output. (off)

com Calculate MSD for centre of mass. (off)

Calculate diffusion for selected atoms using code based on the 'diffusion' routine developed by Hannes Loeffler at STFC (http://www.stfc.ac.uk/CSE).

11.76 strip

```
strip <mask> [outprefix <name>] [<parmout> file>] [nobox]
```

<**mask**> Remove atoms specified by mask from the system.

[outprefix <prefix>] Write out stripped topology file with name '<prefix>.<Original Topology Name>'.

[parmout <file>] Write corresponding topology to file with name <file>.

[nobox] Remove any box information from the stripped topology.

Strip all atoms specified by <**mask**> from the frame and modify the topology to match for any subsequent Actions. The **outprefix** keyword can be used to write stripped topologies; stripped Amber topologies are fully-functional.

Note that stripping a system renumbers all atoms and residues, so for example after this command:

```
strip :1
```

residue 1 will be gone, and the former second residue will now be the first, and so on.

For example, to strip all residues named WAT from each topology/coordinate frame:

```
strip :WAT
```

The next example uses a distance-based mask to strip atoms in a single frame. Note that with the exception of the ***mask*** command, distance-based masks do not update on a per-frame basis. To strip all residues outside of 6.0 from any atom in residues 1 to 14 and write out the stripped topology and coordinates, both with no box information:

```
parm parm7
trajin frame_1000.rst.1
reference frame_1000.rst.1
strip !(:1-14<:6.0) outprefix f1.1 nobox
trajout f1.1.x restart nobox
```

11.77 surf

```
surf [<name>] [<mask1>] [out <filename>] [solutemask <mask>]
[offset <offset>] [nbrcut <cut>]
```

<name> Output data set name.

<mask1> Atoms to calculate surface area for.

out <filename> File to write surface area to.

solutemask <mask> If specified, calculate the contribution of <mask1> to <mask>.

offset <offset> Increment van der Waals radii by <offset>; 1.4 Ang. is the default (as used by Amber).

nbrcut <cut> Only atoms with van der Waals radii greater than <cut> are considered to have neighbors (2.5 Ang Amber default).

Calculate the surface area in $Å^2$ of atoms in <mask> (if no mask specified, all atoms not marked as 'solvent' that are part of a molecule > 1 atom in size) using the LCPO algorithm of Weiser et al.[23]. In order for this to work, the topology needs to have bond information and atom type information.

Note that even if <mask> does not include all solute atoms, the neighbor list is still calculated for all solute atoms so the surface area calculated reflects the contribution of atoms in <mask> to the overall surface area, not the surface area of <mask> as an isolated system. As a result, it may be possible to obtain a negative surface area if only a small fraction of the solute is selected.

For example, to calculate the overall surface area of all solute atoms, as well as the contribution of residue 1 to the overall surface area, writing both results to "surf.dat":

```
surf out surf.dat
surf :1 out surf.dat
```

11.78 symmrmsd

```
symmrmsd  [<name>] [<mask>] [<refmask>] [out <filename>] [nofit] [mass] [remap]
          [ first | reference | ref <name> | refindex <#> | previous |
            reftraj <name> [parm <parmname> | parmindex <#>] ]
```

[<name>] Output data set name.

[<mask>] Mask of atoms to calculate RMSD for; if not specified, calculate for all atoms.

[<refmask>] Reference mask; if not specified, use <mask>.

[out <filename>] Output data file name.

[nofit] Do not perform best-fit RMSD (not recommended).

[mass] Mass-weight the RMSD calculation.

[remap] Re-arrange atoms according to symmetry. See below for more details.

Reference keywords:

first Use the first trajectory frame processed as reference.

reference Use the first previously read in reference structure (refindex 0).

ref <name> Use previously read in reference structure specified by filename/tag.

refindex <#> Use previously read in reference structure specified by <#> (based on order read in).

previous Use frame prior to current frame as reference.

reftraj <name> Use frames from COORDS set <name> or read in from trajectory file <name> as references. Each frame from <name> is used in turn, so that frame 1 is compared to frame 1 from <name>, frame 2 is compared to frame 2 from <name> and so on. If <trajname> runs out of frames before processing is complete, the last frame of <trajname> continues to be used as the reference.

parm <parmname> | parmindex <#> If reftraj specifies a file associate trajectory <name> with specified topology; if not specified the first topology is used.

Perform symmetry-corrected RMSD calculation. This is done by identifying potential symmetric atoms in each residue, performing an initial best-fit, then

determining which configuration of symmetric atoms will give the lowest RMSD using atomic distance to reference atoms.

Note that when re-mapping, all atoms in the residues of interest should be selected to prevent cases where selected symmetric atoms are swapped but the atoms they are bonded to are not. Also, occasionally larger symmetric structures (e.g. 6 membered rings) may become distorted due to only part of the residue being corrected for symmetry. This appears to happen about 4% of the time but does not overly inflate the RMSD. The ***'check'*** command can be used after ***symmrmsd*** to look for such distortions.

Warning: the symmetry correction is generally robust enough to account for symmetries in the standard amino and nucleic acid residues, but has not been extensively tested on residues with more extended types of symmetry.

11.79 temperature

`temperature [<name>] {frame | [<mask>] [ntc <#>]} [out <filename>]`

[<name>] `Data set name.`

frame `Do not calculate temperature; use existing frame temperature.`

[<mask>] `Atoms to calculate temperature for.`

[ntc <#>] `Value of SHAKE bond constraint: 1 - none, 2 - bonds to H, 3- all bonds (equivalent to SANDER/PMEMD).`

[out <filename>] `File to write values to.`

Calculate temperature in frame based on velocity information. If **'frame'** is specified just use frame temperature (read in from e.g. REMD trajectory).

11.80 time

`time {time0 <initial time> dt <step> [update] | remove}`

time0 <initial time> `Time of the first frame (ps).`

dt <step> `Time step between frames (ps).`

[update] `If specified, modify any existing time info.`

remove `Remove any time info from frame.`

Either add time information to frames, modify existing time information in frames, or remove existing time information from frames. Note that currently COORDS data sets do not store time information, so using this command with the ***crdaction*** command will have no effect.

11.81 trans | translate

```
translate [<mask>] [x <dx>] [y <dy>] [z <dz>]
```

Translate atoms in <**mask**> (all atoms if no mask specified) <**dx**> Å in the X direction, <**dy**> Å in the Y direction, and <**dz**> Å in the Z direction.

11.82 unstrip

```
unstrip
```

Requests that the original topology and frame be used for all following actions. This has the effect of undoing any command that modifies the state (such as strip). For example, the following code takes a solvated complex and uses a combination of strip, unstrip, and outtraj commands to write out separate dry complex, receptor, and ligand files:

```
parm Complex.WAT.pdb
trajin Complex.WAT.pdb
# Remove water, write complex
strip :WAT
outtraj Complex.pdb pdb
# Reset to solvated Complex
unstrip
# Remove water and ligand, write receptor
strip :WAT,LIG
outtraj Receptor.pdb pdb
# Reset to solvated Complex
unstrip
# Remove water and receptor, write ligand
strip :WAT
strip !(:LIG)
outtraj Ligand.pdb pdb
```

11.83 unwrap

```
unwrap [center] [{bymol | byres | byatom}]
       [ reference | ref <name> | refindex <#> ] [<mask>]
```

[center] Unwrap by center of mass; otherwise unwrap by first atom position.

bymol Unwrap by molecule (default).

byres Unwrap by residue.

byatom Unwrap by atom.

[reference | **ref** <**name**> | **refindex** <#> **]** Reference structure to use in unwrapping.

[<mask>] Selection to unwrap.

Under periodic boundary conditions, MD trajectories are not continuous if molecules are wrapped(imaged) into the central unit cell. Especially, in sander, with *iwrap*=1, molecular trajectories become discontinuous when a molecule crosses the boundary of the unit cell. This command, **unwrap** processes the trajectories to force the *mask*ed molecules continuous by translating the molecules into the neighboring unit cells. It is the opposite function of **image**, but this command can also be used to place molecules side by side, for example, two strands of a DNA duplex. However, this command fails when the *mask*ed molecules travel more than half of the box size within a single frame.

If the optional argument "reference" is specified, then the first frame is unwrapped according to the reference structure. Otherwise, the first frame is not modified.

As an example, assume that :1-10 is the first strand of a DNA duplex and :11-20 is the other strand of the duplex. Then the following commands could be used to create system where the two strands are not separated artificially:

```
unwrap :1-20
center :1-20 mass origin
image origin center familiar
```

11.84 vector

```
vector [<name>] <Type> [out <filename> [ptrajoutput]] [<mask1>] [<mask2>]
    [magnitude] [ired]
       <Type> = { mask      | minimage  | dipole | center   | corrplane |
                  box       | boxcenter | ucellx | ucelly   | ucellz
              momentum | principal [x|y|z]  | velocity | force }
```

[<name>] Vector data set name.

<Type> Vector type; see below.

[out <filename>] Write vector data to <filename> with format 'Vx Vy Vz Ox Oy Oz' where V denotes vector coordinates and 'O' denotes origin coordinates.

[ptrajoutput] Write vector data in *ptraj* style (Vx Vy Vz Ox Oy Oz Vx+Ox Vy+Oy Vz+Oz). This prevents additional formatting of <filename> and is not compatible with 'magnitude'.

[<mask1>] Atom mask, required for all types except 'box'.

[<mask2>] Second atom mask, only required for type 'mask'.

[magnitude] Store the magnitude of the vector with aspect [Mag].

[ired] `Mark this vector for subsequent IRED analysis with commands 'matrix ired' and 'ired'.`

`Data Sets Created:`

<**name**> `Vector data set.`

<**name**>**[Mag]** `(magnitude only) Vector magnitude.`

This command will keep track of a vector value (and its origin) over the trajectory; the data can be referenced for later use based on the *name* (which must be unique). The types of vectors that can be calculated are:

mask (Default) Store vector from center of mass of atoms in <**mask1**> to atoms in <**mask2**>.

minimage Store minimum-imaged vector from center of mass of atoms in <**mask1**> to atoms in <**mask2**>.

dipole Store the dipole and center of mass of the atoms specified in <**mask1**>. The vector is not converted to appropriate units, nor is the value well-defined if the atoms in the mask are not overall charge neutral.

center Store the center of mass of atoms in <**mask1**>. The reference point is the origin (0.0, 0.0, 0.0).

corrplane This defines a vector perpendicular to the (least-squares best) plane through the atoms in <**mask1**>. The reference point is the center of mass of atoms in <**mask1**>.

box (No mask needed) Store the box lengths of the trajectory. The reference point is the origin (0.0, 0.0, 0.0).

boxcenter (No mask needed) Store the center of the box as a vector.

ucell{x|y|z}: (No mask needed) Store specified unit cell (i.e. box) vector.

momentum Store momentum of atoms selected by <mask1> (requires velocities).

principal [x|y|z] Store one of the principal axis vectors determined by diagonalization of the inertial matrix from the coordinates of the atoms specified by <**mask1**>. The eigenvector with the largest eigenvalue is considered "`x`" (i.e., the hardest axis to rotate around) and the eigenvector with the smallest eigenvalue is considered "`z`". If none of `x` or `y` or `z` are specified, then the "x" principal axis is stored. The reference point is the center of mass of atoms in <**mask1**>.

velocity Store velocity of atoms in <mask1> (requires velocities).

force Store force of atoms in <mask1> (requires forces).

Cpptraj supports writing out vector data in a pseudo-trajectory format for easy visualization. Once a vector data set has been generated the writedata command can be used with the vectraj keyword (see 6 on page 23 for more details) to write a pseudo trajectory consisting of two atoms, one for the vector origin and one for the vector from the origin (i.e. V+O). For example, to create a MOL2 containing a pseudo-trajectory of the minimum-imaged vector from residue 4 to residue 11:

```
trajin tz2.nc
vector v8 minimage out v8.dat :4 :11
run
writedata v8.mol2 vectraj v8 trajfmt mol2
```

Auto-correlation or cross-correlation functions can be calculated subsequently for vectors using either the ***corr*** analysis command or the ***timecorr*** analysis command (to calculate via spherical harmonic theory).

11.85 velocityautocorr

```
velocityautocorr [<set name>] [<mask>] [usevelocity] [out <filename>] [diffout <file>]
                 [maxlag <frames>] [tstep <timestep>] [direct] [norm]
```

[<set name>] Data set name.

[<mask>] Atoms(s) to calculate velocity autocorrelation (VAC) function for.

[usevelocity] Use velocity information in frame if present. This will only give sensible results if the velocities are recorded close to the order of the simulation time step.

[out <filename>] Write VAC function to <filename>.

[diffout <file>] File to write diffusion constants to.

[maxlag <frames>] Maximum lag in frames to calculate VAC function for. Default is half the total number of frames.

[tstep <timestep>] Time between frames in ps (default 1.0).

[direct] Calculate VAC function directly instead of via FFT (will be much slower).

[norm] Normalize resulting VAC function to 1.0.

DataSet Aspects:

[D] Diffusion constant calculated from integral over VAC function in $1x10^{-5}$ cm^2/s.

Calculate the velocity autocorrelation (VAC) function averaged over the atoms in <mask>. Pseudo-velocities are calculated using coordinates and the specified time step. As with all time correlation functions the statistical noise will increase if the maximum lag is greater than half the total number of frames. In addition to calculating the velocity autocorrelation function, the self-diffusion coefficient will be reported in the output, calculated from the integral over the VAC function.

11.86 volmap

```
volmap filename dx dy dz <mask> [radscale <factor>]
    { data <existing set> |
      name <setname> { size <x,y,z> [center <x,y,z>] |
                       centermask <mask> [buffer <buffer>] } }
    [peakcut <cutoff>] [peakfile <xyzfile>]
```

filename The name of the output file with the grid density.

dx, dy, dz The grid spacing (Angstroms) in the X-, Y-, and Z-dimensions, respectively

<mask> The atom selection from which to calculate the number density.

radscale <factor> Factor by which to scale radii (by division). To match the atomic radius of Oxygen used by the VMD volmap tool, a scaling factor of 1.36 should be used. Default 1.0.

data <setname> Name of existing grid data set to use.

name <setname> Name of grid set that will be created (size/center or centermask/buffer keywords).

size <x,y,z> Specify the size of the grid in the X-, Y-, and Z-dimensions. Must be used alongside the center argument.

center <x,y,z> Specify the grid center explicitly. Note, the size argument must be present in this case. Default is the origin.

centermask <mask> The mask around which the grid should be centered (via geometric center). If this is omitted and the center and size are not specified, the default <mask> entered (see above) is used in its place.

buffer <buffer> A buffer distance, in Angstroms, by which the edges of the grid should clear every atom of the centermask (or default mask if centermask is omitted) in every direction. The default value is

3. The buffer is ignored if the center and size are specified (see below).

peakcut <**cutoff**> The minimum density required to consider a local maximum a 'density peak' in the outputted peak file (default 0.05).

peakfile <**xyzfile**> A file in XYZ-format that contains a carbon atom centered at the grid point of every local density maximum. This file is necessary input to the spam action command.

Grid data as a volumetric map, similar to the 'volmap' command in VMD. The density is calculated by treating each atom as a 3-dimensional Gaussian function whose standard deviation is equal to the van der Waals radius. The density calculated is the number density averaged over the entire simulation. The grid can be specified in one of three ways:

1. An existing grid data set (from e.g. bounds), specified with the **data** keyword.
2. Via the sizes and center specified by the **size** and **center** keywords (comma-separated strings, e.g. '20,20,20').
3. Centered on the atoms in the mask given by **centermask** with an additional buffer in each direction specified by **buffer**.

11.87 volume

`volume [<name>] [out <filename>]`

<**name**> Data set name.

out <**filename**> Output file name.

Calculate unit cell volume.

11.88 watershell

`watershell <solutemask> [out <filename>] [lower <lower cut>] [upper <upper cut>] [noimage] [<solventmask>]`

<**solutemask**> Atom mask corresponding to solute of interest (required).

[out <**filename**>**]** Output file name.

[lower <**lower cut**>**]** Cutoff for the first water shell (default 3.4 Angstroms).

[upper <**upper cut**>**]** Cutoff for the second water shell (default 5.0 Angstroms).

[noimage] Do not image distances.

[<solventmask>] Optional atom mask corresponding to solvent.

DataSet Aspects:

[lower] Number of solvent molecules in first solvent shell.

[upper] Number of solvent molecules in second solvent shell.

This option will count the number of waters within a certain distance of the atoms in the <solutemask> in order to represent the first and second solvation shells. The optional <solventmask> can be used to consider other atoms as the solvent; the default is ":WAT".

This action is often used prior to the ***closest*** command in order to determine how many waters around a solute should be retained to maintain the first and/or second water shells.

As of version 17 this command is CUDA-enabled in CUDA versions of CPPTRAJ.

11.89 xtalsymm

```
xtalsymm <mask> group <space group> [collect [centroid]]
      [ first | reference | ref <name> | refindex <#> ]
      [na <na>] [nb <nb>] [nc <nc>]
```

<mask> Atom mask defining the asymmetric unit within the larger system (required).

group <space group> The space group to which the system belongs. Omit spaces in the name. Example: "P22(1)2(1)".

[collect] Optional flag to have all solvent particles, not just the asymmetric units, re-imaged. This will trigger cpptraj to compute the unit cell volume that constitutes the aymmetric unit and thereby classify all particles for re-imaging.

[centroid] If specified along with collect, re-image solvent molecules by centroids, not individual atom coordinates. This is useful for keeping water molecules intact.

[first | reference | ref <name> | refindex <#>] Reference structure to use for determining crystal symmetry.

[na <na>] [nb <nb>] [nc <nc>] The number of times the crystal unit cell is replicated along the "a," "b," or "c" axes (for orthorhombic unti cells, these are

```
the x, y, and z axes) of the simulation; default is
1.  Many crystal unit cells are too small in one or
more dimensions for our simulation cutoffs, and
replicating the unit cell is an effective way to
counter imaging artifacts even for larger unit
cells.
```

Calculate the optimal approach for superimposing symmetry-related subunits of the simulation back onto one another. The calculation assumes that the system is a simulation of an X-ray structure in its native crystal lattice, finds all copies of the asymmetric unit among the entire system, and devises plans for re-imagining their coordinates to superimpose them back on the original asymmetric unit. The space group information can be found in a PDB X-ray structure used as the initial coordinates for a simulation. All 230 space groups are supported, and a scan of the PDB was made to ensure that common variants of the names are included (P2(1)22(1) is the same as P22(1)2(1), but with different axis conventions). If your space group is not understood, contact the Amber mailing list. This command is compute intensive, especially for simulations that are "supercells" containing many crystallographic unit cells.

This command will cause cpptraj to locate all asymmetric units from within the topology, then determine what wrapping, if any, has occurred in order to bring about an optimal re-alignment based on the space group symmetry operations. The user need not worry about wrapping or drift of the simulation over time–the asymmetric units will be re-imaged frame by frame. Coordinate modifications due to this action are permanent and will affect the results of subsequent actions and analyses.

12 Analysis Commands

Analyses in *cpptraj* operate on data sets which have been generated by Actions in a prior Run or read in with a ***readdata*** command (8.19 on page 45). Unlike *ptraj*, Analysis commands in *cpptraj* do not need to be prefaced with `'analysis'`. The exception to this is '***analyze matrix***' in order to differentiate it from the ***matrix*** Action command; users are encouraged to use the new command ***diagmatrix*** instead.

Like Actions, when an Analysis command is issued it is by default added to the Analysis queue and is not executed until after trajectory processing is completed; a complete list of data sets available for analysis is shown after trajectory processing (prefaced by 'DATASETS') or can be shown with the '***list dataset***' command. Analyses can also be executed immediately via the ***runanalysis*** command (8.24 on page 47).

Note that for Analysis commands that use COORDS data sets, if no COORDS data set is specified then a default one will be automatically created from frames read in by ***trajin*** commands.

Command	Description	Se
autocorr	Calculate autocorrelation function for multiple data sets.	N
avg	Calculate average, standard deviation, min, and max for (or over) data sets.	N
calcstate	Calculate states based on given data sets and criteria.	N
cluster	Perform cluster analysis.	COORI
corr, correlationcoe	Calculate auto or cross correlation for 1 or 2 data sets.	1D s
cphstats	Calculate statistics for constant pH data sets.	pH
crank, crankshaft	Calculate crankshaft motion between two data sets.	2
crdfluct	Calculate atomic fluctuations (RMSF) for atoms over time blocks.	C
crosscorr	Calculate a matrix of Pearson product-moment coefficients between given data sets.	N
curvefit	Perform non-linear curve fitting on given data set.	1
diagmatrix	Calculate eigenvectors and eigenvalues from given symmetric matrix.	symr
divergence	Calculate Kullback-Leibler divergence between two data sets.	2
FFT	Perform a fast Fourier transform on data sets.	N
hausdorff	Calculate the Hausdorff distance for given matrix data set(s).	N 2
hist, histogram	Calculate N-dimensional histogram for N given data sets.	N
integrate	Perform integration on each of the given data sets.	N
ired	Perform isotropic reorientational eigenmode dynamics analysis using given IRED vectors.	N II
kde	Calculate 1D histogram from given data set using a kernel density estimator. Also time-dependent Kullback-Leibler divergence analysis with another set.	1 or
lifetime	Perform lifetime analysis on given data sets.	N
lowestcurve	For each given data set, calculate a curve that traces the lowest N points over specified bins.	N
meltcurve	Calculate a melting curve from given data sets assuming simple 2 state kinetics.	N
modes	Perform various analyses on eigenmodes (from e.g. ***diagmatrix***).	ei
multicurve	Perform non-linear curve fitting for multiple input data sets.	N
multihist	Calculate 1D histograms (optionally with a kernel density estimator) from multiple input data sets.	N
phipsi	Calculate and plot the average phi and psi values from input dihedral data sets.	N phi
regress	Perform linear regression on multiple input data sets.	N
remlog	Calculate various statistics from a replica log data set.	r
rms2d, 2drms	Calculate 2D RMSD between frames in 1 or 2 COORDS data sets.	1 or
rmsavgcorr	Calculate RMS average correlation curve for a COORDS data set.	C
rotdif	Calculate rotational diffusion using given rotation matrices (from e.g. ***rms***).	rotat
runningavg	Calculate running average for given data sets using given window size.	N
spline	Calculate cubic splines for given data sets.	N
stat, statistics	Calculate various statistics for given data sets.	N
ti	Peform Gaussian quadrature integration for given DV/DL data sets.	N
timecorr	Calculate auto/cross-correlation functions for given	1

	vector(s) using spherical harmonics.	
vectormath	Perform math on given vector data sets.	
wavelet	Perform wavelet analysis on coordinates from given COORDS set.	C

12.1 autocorr

```
autocorr [name <dsetname>] <dsetarg0> [<dsetarg1> ...] [out <filename>]
         [lagmax <lag>] [nocovar] [direct]
```

<dsetarg0> [dsetarg1> ...] Argument(s) specifying datasets to be used.

[name <dsetname>] Store results in dataset(s) named <dsetname>:X.

[out <filename>] Write results to file named <filename>.

[lagmax] Maximum lag to calculate for. If not specified all frames are used.

[nocovar] Do not calculate covariance.

[direct] Do not use FFTs to calculate correlation; this will be much slower.

This is for integer/double/float datasets only; for vectors see the ***'timecorr'*** *command.*

Calculate auto-correlation (actually auto-covariance by default) function for datasets specified by one or more dataset arguments. The datasets must have the same # of data points.

12.2 avg

```
avg <dset0> [<dset1> ...] [torsion] [out <file>] [oversets]
    [name <name>] [nostdout]
```

<dsetX> Data set(s) to calculate the average for.

[torsion] If the data sets are not already marked periodic (e.g. if read in via 'readdata'), treat them as periodic torsion.

[out <file>] File to write results to.

[oversets] If specified, calculate the average over all inpout sets instead of each input set.

[name <name>] Output data set name.

[nostdout] If 'nostdout' specified do not write averages to STDOUT when 'out' not specified.

DataSets Created (not oversets):

<**name**>**[avg]** Average of each set.

<**name**>**[sd]** Standard deviation of each set.

<**name**>**[ymin]** Y minimum of each set.

<**name**>**[ymax]** Y maximum of each set.

<**name**>**[yminidx]** Index of minimum Y value.

<**name**>**[ymaxidx]** Index of maximum Y value.

<**name**>**[names]** Name of each set.

DataSets Created (oversets)

<**name**> Average over all input sets for each frame.

<**name**>**[SD]** Standard deviation over all input sets for each frame.

Calculate the average, standard deviation, min, and max of given 1D data sets. Alternatively, if **oversets** is specified the average over each set for each point is calculated; this requires all input sets be the same size.

For example, to read in data from a file named perres.peptide.dat and calculate the averages etc for all the input sets:

```
readdata perres.peptide.dat
avg perres.peptide.dat out output.dat name V
```

12.3 calcstate

```
calcstate {state <ID>,<dataset>,<min>,<max>[,<dataset1>,<min1>,<max1>]} ...
       [out <state v time file>] [name <setname>]
       [curveout <curve file>] [stateout <states file>]
       [transout <transitions file>] [countout <count file>]
```

state <**ID**>,<**dataset**>,<**min**>,<**max**> Define a state according to given data set and criteria. Multiple states can be given, and each state can have multiple criteria. If multiple criteria are specified, each one must be satisfied in order to assign the state. If the same state is defined multiple times, the state will be assigned if either criteria match.

<**ID**> Name to give each state index. State indices start at 0. -1 means "undefined state".

<**dataset**> Data set to use.

<min>,<max> Frames with data set value above <min> and below <max> will be assigned <ID>.

[out <state v time file>] File to write state index vs frame to.

[name <setname>] Data set name.

[curveout <curve file>] File to write state lifetime and transition curves to.

[stateout <states file>] File to write state lifetime data to.

[transout <transitions file>] File to write state transition data to.

[countout <state count file>] File to write state counts (i.e. how many frames each state was observed) to.

DataSets Created:

<setname> State index vs frame.

<setname>[Count] Number of frames each state was observed.

<setname>[Frac] Fraction of time each state was observed

<setname>[Nlifetimes] Number of times each state was reached.

<setname>[Avglife] Average lifetime length for each state.

<setname>[Maxlife] Maximum lifetime of each state.

<setname>[Name] Name (<ID>) of each state.

<setname>[Xlifetimes] Number of times each state transitioned to each other state.

<setname>[Xavglife] Average lifetime of each state before transitioning to each other state.

<setname>[Xmaxlife] Maximum lifetime of each state before transitioning to each other state.

<setname>[Xname] Name of each transition, format "StateA->StateB".

<setname>[sCurve]:X State curves; lifetime curve for transitions from given state to any other state.

<setname>[tCurve]:X Transition curves; lifetime curve for transitions from given state to other specific state.

Data for the specified data set(s) that matches the given criteria will be assigned a state index. State indices start from 0 and match the order in which **state** keywords were given. The -1 state index is reserved for "undefined state". For example, the following input:

```
parm DPDP.parm7
trajin DPDP.nc
distance d1 :19@O :12@N
angle a1 :19@O :12@H :12@N
calcstate state D,d1,3.0,4.0 state A,a1,100,120 out state.dat curveout curve.agr \
                stateout States.dat transout States.dat name d1_a1
run
```

Defines two states. State index 0 is defined as a state named "D" based on the distance from ':19@O' to ':12@N' (data set d1) being between 3 and 4 Angstroms. State index 1 is defined as a state named "A" based on the angle between ':19@O', ':12@H', and ':12@N' (data set a1) being between 100 and 120 degrees. The output in state.dat might look like:

```
#Frame          d1_a1
       1           -1
       2            0
       3            0
       4            0
       5           -1
       6            1
       7           -1
       8           -1
       9            0
      10           -1
```

where the values in column d1_a1 refer to state index: -1 is undefined, 0 is state "D", and 1 is state "A".

To define a state State1 as having a distance named "dist" between 2.5 and 5.0 Ang. and an angle named "ang" between 30 and 60 degrees OR having a distance named "distA" between 0.0 and 3.0 Ang.:

```
calcstate state State1,dist,2.5,5.0,ang,30,60 \
          state State1,distA,0.0,3.0
```

Lifetime curves (see 12.19 on page 191 for further explanation) are calculated for transitions from each state to any other state (aspect [sCurve]) and each state to each other state (aspect [tCurve]). In this case there will be 3 sCurves and 4 tCurves:

```
d1_a1[sCurve]:0 "Undefined" (double), size is 10
d1_a1[sCurve]:1 "D" (double), size is 3
```

```
d1_a1[sCurve]:2 "A" (double), size is 1
d1_a1[tCurve]:0 "Undefined->D" (double), size is 10
d1_a1[tCurve]:1 "D->Undefined" (double), size is 3
d1_a1[tCurve]:2 "Undefined->A" (double), size is 1
d1_a1[tCurve]:3 "A->Undefined" (double), size is 1
```

Lifetime analysis from each state to any other state is directed to the file specified by **stateout** and has format:

```
#Index N Average Max State
```

Where `#Index` is the state index, `N` is the number of lifetimes in that state, `Average` is the average lifetime while in that state (in frames), `Max` is the maximum lifetime while in that state (in frames) and `State` is the name of the state.

Finally, lifetime analysis of transitions from each state to each other state is directory to the file specified by transout and has format:

```
#N Average Max Transition
```

Where `#N` is the number of transitions, `Average` is the average lifetime (in frames) in the first state before transitioning to the second state, `Max` is the max lifetime (in frames) before transitioning to the second state, and `Transition` is the name of the transition.

12.4 cluster

```
cluster [crdset <crd set> | nocoords]
  Algorithms:
[hieragglo [epsilon <e>] [clusters <n>] [linkage|averagelinkage|complete]
      [epsilonplot <file>] [includesieved_cdist]]
[dbscan minpoints <n> epsilon <e> [sievetoframe] [kdist <k> [kfile <prefix>]]]
    [dpeaks epsilon <e> [noise] [dvdfile <density_vs_dist_file>]
  [choosepoints {manual | auto}]
  [distancecut <distcut>] [densitycut <densitycut>]
  [runavg <runavg_file>] [deltafile <file>] [gauss]]
    [kmeans clusters <n> [randompoint [kseed <seed>]] [maxit <iterations>]
    [{readtxt|readinfo} infofile <file>]
  Distance options:
{[[rms | srmsd] [<mask>] [mass] [nofit]] | [dme [<mask>]] |
      [data <dset0>[,<dset1>,...]]}
[sieve <#> [random [sieveseed <#>]]] [loadpairdist] [savepairdist] [pairdist <file>]
    [pairwisecache {mem | none}] [includesieveincalc] [pwrecalc]
  Output options:
[out <cnumvtime>] [gracecolor] [summary <summaryfile>] [info <infofile>]
[summarysplit <splitfile>] [splitframe <comma-separated frame list>]
    [bestrep {cumulative|centroid|cumulative_nosieve}] [savenreps <#>]
```

```
    [clustersvtime <filename> cvtwindow <window size>]
[cpopvtime <file> [normpop | normframe] [lifetime]]
    [sil <silhouette file prefix>] [assignrefs [refcut <rms>] [refmask <mask>]]
  Coordinate output options:
[ clusterout <trajfileprefix> [clusterfmt <trajformat>] ]
[ singlerepout <trajfilename> [singlerepfmt <trajformat>] ]
[ repout <repprefix> [repfmt <repfmt>] [repframe] ]
    [ avgout <avgprefix> [avgfmt <avgfmt>] ]
```

[crdset <crd set>] Name of previously generated COORDS data set. If not specified the default COORDS set will be used unless nocoords has been specified.

[nocoords] Do not use a COORDS data set; distance metrics that require coordinates and coordiante output will be disabled.

Algorithms:

hieragglo (Default) Use hierarchical agglomerative (bottom-up) approach.

- **[epsilon <e>]** Finish clustering when minimum distance between clusters is greater than <e>.
- **[clusters <n>]** Finish clustering when <n> clusters remain.
- **[linkage]** Single-linkage; use the shortest distance between members of two clusters.
- **[averagelinkage]** Average-linkage (default); use the average distance between members of two clusters.
- **[complete]** Complete-linkage; use the maximum distance between members of two clusters.
- **[epsilonplot <file>]** Write number of clusters vs epsilon to <file>.
- **[includesieved_cdist]** Include sieved frames in final cluster distance calculation (may be very slow).

dbscan Use DBSCAN clustering algorithm of Ester et al.[24]

- **minpoints <n>** Minimum number of points required to form a cluster.
- **epsilon <e>** Distance cutoff between points for forming a cluster.
- **[sievetoframe]** When restoring sieved frames, compare frame to every frame in a cluster instead of the centroid; slower but more accurate.

[kdist <k>] Generate K-dist plot for help in determining DBSCAN parameters (see below).

[kfile <prefix>] Prefix for K-dist plot file.

dpeaks Use the density peaks algorithm of Rodriguez and Laio[25]

epsilon <e> Cutoff for determining local density in Angstroms.

[noise] If specified, treat all points within epsilon of another cluster as noise.

[dvdfile <density_vs_dist_file>] File to write density versus minimum distance to point with next highest density. This can be used to determine appropriate cutoffs for distance and density in a subsequent step with choosepoints manual.

[choosepoints {manual | auto}] Specify whether clusters will be chosen based on specified distance/density cutoffs, or automatically. If not specified only the density vs distance file will be written and no clustering will be performed. Currently manual is recommended.

[distancecut <distcut>] [densitycut <densitycut>] If choosepoints manual, points with minimum distance greather than or equal to <distcut> and density greater than or equal to <densitycut> will be chosen.

[runavg <runavg file>] If choosepoints automatic, the calculated running average of density versus distance will be written to <runavg file>.

[deltafile <file>] If choosepoints automatic, distance minus the running average for each point will be written to this file.

[gauss] Calculate density with Gaussian kernels instead of using discrete density.

kmeans Use K-means clustering algorithm.

clusters <n> Finish clustering when number of clusters is <n>.

[randompoint] Randomize initial set of points used (recommended).

[kseed <seed>] Random number generator seed for randompoint.

[maxit <iteration>] Algorithm will run until frames no longer change clusters of <iteration> iterations are reached (default 100).

readtxt|**readinfo** No clustering - read in previous cluster results.

infofile <**file**> Cluster info file to read.

Distance Metric Options:

[rms | **srmsd[**<**mask**>**]]** (Default rms) Distance between frames calculated via best-fit coordinate RMSD using atoms in <mask>. If srmsd specified use symmetry-corrected RMSD (see 11.78 on page 154).

[mass] Mass-weight the RMSD.

[nofit] Do not fit structures onto each other prior to calculating RMSD.

dme [<**mask**>**]** Distance between frames calculated using distance-RMSD (aka DME, *distrmsd*) using atoms in <mask>.

[data <**dset0**>**[,**<**dset1**>**,...]** Distance between frames calculated using specified data set(s) (Euclidean distance).

[sieve <#>**]** Perform clustering only for every <#> frame. After clustering, all other frames will be added to clusters.

[random] When sieve is specified, select initial frames to cluster randomly.

[sieveseed <#>**]** Seed for random sieving; if not set the wallclock time will be used.

[pairdist <**file**>**]** File to use for loading/saving pairwise distances.

[loadpairdist] Load pairwise distances from <file> (CpptrajPairDist if pairdist not specified).

[savepairdist] Save pairwise distances from <file> (CpptrajPairDist if pairdist not specified). NOTE: If sieving was performed only the calculated distances are saved.

[pairwisecache {mem | **disk** | **none}]** Cache pairwise distance data in memory (default), to disk, or disable pairwise caching. No caching will save memory but be extremely slow. Caching to disk will likely be slow unless writing to a fast storage device (e.g. SSD) - data is saved to a file named 'CpptrajPairwiseCache'.

[includesieveincalc] Include sieved frames when calculating within-cluster average (may be very slow).

[pwrecalc] If a loaded pairwise distance file does not match the current setup, force recalculation.

Output Options:

[out <cnumvtime>] Write cluster # vs frame to <cnumvtime>. Algorithms that calculate noise (e.g. DBSCAN) will assign noise points a value of -1.

[gracecolor] Instead of cluster # vs frame, write cluster# + 1 (corresponding to colors used by XMGRACE) vs frame. Cluster #s larger than 15 are given the same color. Algorithms that calculate noise (e.g. DBSCAN) will assign noise points a color of 0 (blank).

[summary <summaryfile>] Summarize each cluster with format '#Cluster Frames Frac AvgDist Stdev Centroid AvgCDist':

#Cluster Cluster number starting from 0 (0 is most populated).

Frames # of frames in cluster.

Frac Size of cluster as fraction of total trajectory.

AvgDist Average distance between points in the cluster.

Stdev Standard deviation of points in the cluster.

Centroid Frame # of structure in cluster that has the lowest cumulative distance to every other point.

AvgCDist Average distance of this cluster to every other cluster.

[info <infofile>] Write ptraj-like cluster information to <infofile>. This file has format:

```
#Clustering: <X> clusters <N> frames
#Cluster <I> has average-distance-to-centroid <AVG>
...
#DBI: <DBI>
#pSF: <PSF>
#Algorithm: <algorithm-specific info>
<Line for cluster 0>
...
#Representative frames: <representative frame list>
```

Where <X> is the number of clusters, <N> is the number of frames clustered, <I> ranges from 0 to <X>-1, <AVG> is the average distance of all frames in that cluster to the centroid, <DBI> is the

Davies-Bouldin Index, <pSF> is the pseudo-F statistic, and <representative frame list> contains the frame # of the representative frame (i.e. closest to the centroid) for each cluster. Each cluster has a line made up of characters (one for each frame) where '.' means 'not in cluster' and 'X' means 'in cluster'.

[summarysplit <splitfile>] Summarize each cluster based on which of its frames fall in portions of the trajectory specified by splitframe with format '#Cluster Total Frac C# Color NumInX ... FracX ... FirstX':

#Cluster Cluster number starting from 0 (0 is most populated).

Total # of frames in cluster.

Frac Size of cluster as a fraction of the total trajectory.

C# Grace color number.

Color Text description of the color (based on standard XMGRACE coloring).

NumInX Number of frames in Xth portion of the trajectory.

FracX Fraction of frames in Xth portion of the trajectory.

FirstX Frame in the Xth portion of the trajectory where the cluster is first observed.

[splitframe <frame>] For summarysplit, frame or comma-separated list of frames to split the trajectory at, e.g. '100,200,300'.

[bestrep {cumulative|centroid|cumulative_nosieve}] Method for choosing cluster representative frames.

cumulative Choose by lowest cumulative distance to all other frames in cluster. Default when not sieving.

centroid Choose by lowest distance to cluster centroid. Default when sieving.

cumulative_nosieve Choose by lowest cumulative distance to all other frames, ignoring sieved frames.

[savenreps <#>] Number of best representative frames to choose (default 1).

[clustersvtime <filename>] Write number of unique clusters observed in a given time window to <filename>.

[cvtwindow <windowsize>] Window size for clustersvtime output.

[cpopvtime <file> [normpop | normframe]] Write cluster population vs time to <file>; if normpop specified normalize each cluster to 1.0; if normframe specified normalize cluster populations by number of frames.

[sil <prefix>] Write average cluster silhouette value for each cluster to '<prefix>.cluster.dat' and cluster silhouette value for each individual frame to '<prefix>.frame.dat'.

assignrefs In summary/summarysplit, assign clusters to loaded representative structures if RMSD to that reference is less than specified cutoff.

[refcut <rms>] RMSD cutoff in Angstroms.

[refmask <mask>] Mask to use for RMSD calculation. If not specified the default mask is all heavy atoms.

Coordinate Output Options:

clusterout <trajfileprefix> Write frames in each cluster to files named <trajfileprefix>.cX, where X is the cluster number.

clusterfmt <trajformat> Format keyword for clusterout (default Amber Trajectory).

singlerepout <trajfilename> Write all representative frames to single trajectory named <trajfilename>.

singlerepfmt <trajformat> Format keyword for singlerepout (default Amber Trajectory).

repout <repprefix> Write representative frames to separate files named <repprefix>.X.<ext>, where X is the cluster number and <ext> is a format-specific filename extension.

repfmt <trajformat> Format keyword for repout (default Amber Trajectory).

repframe Include representative frame number in repout filename.

avgout <avgprefix> Write average structure for each cluster to separate files named <avgprefix>.X.<ext>, where X is the cluster number and <ext> is a format-specific filename extension.

avgfmt <**trajformat**> `Format keyword for avgout.`

`DataSet Aspects:`

[Pop] `Cluster population vs time; index corresponds to cluster number.`

Note cluster population vs time data sets are not generated until the analysis has been run.

Cluster input frames using the specified clustering algorithm and distance metric. In order to speed up clustering of large trajectories, the **sieve** keyword can be used. In addition, subsequent clustering calculations can be sped up by writing/reading calculated pair distances between each frame to/from a file specified by **pairdist** (or "CpptrajPairDist" if **pairdist** not specified).

Example: cluster on a specific distance:

```
distance endToEnd :1 :255
cluster data endToEnd clusters 10 epsilon 3.0 summary summary.dat info info.dat
```

Example: cluster on the CA atoms of residues 2-10 using average-linkage, stopping when either 3 clusters are reached or the minimum distance between clusters is 4.0, writing the cluster number vs time to "cnumvtime.dat" and a summary of each cluster to "avg.summary.dat":

```
cluster C1 :2-10 clusters 3 epsilon 4.0 out cnumvtime.dat summary avg.summary.dat
```

Clustering Metrics

The Davies-Bouldin Index (DBI) measures sum over all clusters of the within cluster scatter to the between cluster separation; **the smaller the DBI, the better**. The DBI is defined as the average, for all clusters X, of fred, where fred(X) = max, across other clusters Y, of (Cx + Cy)/dXY. Here Cx is the average distance from points in X to the centroid, similarly Cy, and dXY is the distance between cluster centroids.

The pseudo-F statistic (pSF) is another measure of clustering goodness. It is intended to capture the 'tightness' of clusters, and is in essence a ratio of the mean sum of squares between groups to the mean sum of squares within group. **High values are good**. Generally, one selects a cluster-count that gives a peak in the pseudo-f statistic. Formula: A/B, where A = (T - P)/(G-1), and B = P / (n-G). Here n is the number of points, G is the number of clusters, T is the total distance from the all-data centroid, and P is the sum (for all clusters) of the distances from the cluster centroid.

The cluster silhouette is a measure of how well each point fits within a cluster. Values of 1 indicate the point is very similar to other points in the cluster, i.e. it is well-clustered. Values of -1 indicate the point is dissimilar and may fit better in a neighboring cluster. Values of 0 indicate the point is on a border between two clusters.

Hints for setting DBSCAN parameters with 'kdist'

It is not always obvious what parameters to set for DBSCAN. You can get a rough idea of what to set 'mindist' and 'epsilon' to by generating a so-called "K-dist" plot with the 'kidst <k>' option. The K-dist plot shows for each point (X axis) the Kth farthest distance (Y axis), sorted by decreasing distance. You supply the same distance metric and sieve parameters you want to use for the actual clustering, but nothing else. For example:

```
cluster C0 dbscan kdist 4 rms :1-4@CA sieve 10 loadpairdist pairdist CpptrajPairDist
```

The K-dist plot will be named <prefix>.<k>.dat, with the default prefix being 'Kdist' (in this case the file name would be Kdist.4.dat). The K-dist plot usually looks like a curve with an initially steep slope that gradually decreases. Around where the initial part of the curve starts to flatten out (indicating an increas in density) is around where epsilon should be set; minpoints is set to whatever <k> was. It has been suggested that the shape of the K-dist curve doesn't change too much after Kdist=4, but users are encouraged to experiment.

Using 'dpeaks' clustering

The 'dpeaks' (density peaks) algorithm attempts to find clusters by identifying points in high density regions which are far from other points of high density[25]. There are two ways these points can be chosen. The first and recommended way is manually. In this method, clustering if first run with **choosepoints** not specified to generate a plot containing density versus minimum distance to point with next highest density (the decision graph). Appropriate cut offs for distance and density can then be chosen based on visual inspection; cutoffs should be chosen so that they select points that have both a high density and a high distance to point with next highest density. Clustering can then be run again with **distancecut** and **densitycut** set.

The second way is automatically; cpptraj will attempt to identify outliers in the density vs distance plot based on distance from the running average. Although this only requires a single pass, this method of choosing points is not well-tested and currently not recommended.

The CpptrajPairDist file format

The CpptrajPairDist file is binary; the exact format depends on what version of cpptraj generated the file (since earlier versions had no concept of 'sieve'). The CpptrajPairDist file starts with a 4 byte header containing the characters 'C' 'T' 'M' followed by the version number. A quick way to figure out the version is to use the linux 'od' command to output the first 4 bytes as hexadecimal, e.g.:

```
$ od -t x1 -N 4 CpptrajPairDist 0000000 43 54 4d 02
```

So the CpptrajPairDist file version in the above example is 2.

The next few numbers describe the matrix size and depend on the version.

Version 0: Two 4-byte integers: # of rows and # of elements.

Version 1: Two 8-byte unsigned integers (equivalent to size_t on most systems): # of rows and # of elements.

Version 2: Three 8 byte unsigned integers: original # of rows, actual # of rows, and sieve value.

This is followed by the actual matrix data, stored as a single array of floats (4 bytes). For versions 1 and 2 the number of elements is explicitly stored. For version 2, to calculate the number of matrix elements you need to read:

```
Elements = (actual_rows * (actual_rows - 1)) / 2
```

The cluster pair-distance matrix is an upper-right triangle matrix without the diagonal (in row-major order), so the first element is the distance between elements 0 and 1, the second is between elements 0 and 2, etc.

In version 2 files, if the sieve value is greater than 1 that means original_rows > actual_rows and there is an additional array of characters original_nrows long, with 'T' if the row is being ignored (i.e. it was sieved out) and 'F' if the row is active (i.e. is active in the actual pairwise-distance matrix).

The code that cpptraj uses to read in CpptrajPairDist files is in ClusterMatrix::LoadFile() (ClusterMatrix.cpp).

12.5 cphstats

```
cphstats <pH sets> [name <name>] [statsout <statsfile>] [deprot]
      [fracplot [fracplotout <file>]]
```

<**pH sets**> Previously read in pH data sets.

name <**name**> Output set name.

statsout <**statsfile**> Write pH statistics to <statsfile>

deprot If specified, calculate fraction deprotonated instead of protonated.

fracplot If specified, calculate fraction protonated/deprotonated vs pH.

fracplotout <**file**> File to write fraction plots to.

Data Sets Generated

<**name**>**[Frac]:**<**idx**> Fraction protonated/deprotonated for residue <idx>.

Calculate statistics for constant pH simulation data previously read in with ***readdata*** (see 6.11 on page 29). Statistics are calculated for each residue at each input pH. Output format is as follows:

```
Solvent pH is <pH>
<residue name> <residue number> : Offset <offset from predicted> Pred <predicted pH> Fr
...
Average total molecular protonation: <avg>
```

A line is printed for each residue. This functionality is similar to the **cphstats** utility that comes with Amber (see **??** on page ??).

Note that data from constant pH REMD must be sorted prior to use with ***cphstats***. See the ***readensembledata*** (8.20 on page 45) and ***sortensembledata*** (8.27 on page 48) commands for more details.

For example, to read in constant pH data from constant pH REMD, sort and analyze:

```
readensembledata ExplicitRemd/cpout.001 cpin ExplicitRemd/cpin name PH
sortensembledata PH
runanalysis cphstats PH[*] statsout stats.dat fracplot fracplotout frac.agr deprot
```

12.6 corr | correlationcoe

```
corr out <outfilename> <dataset1> [<dataset2>]
     [lagmax <lag>] [nocovar] [direct]
```

out <**outfilename**> Write results to file named <outfilename>. The datasets must have the same # of data points.

<**dataset1**> [<**dataset2**>] Data set(s) to calculate correlation for. If one dataset or the same dataset is given twice, the auto-correlation will be calculated, otherwise cross-correlation.

[lagmax] Maximum lag to calculate for. If not specified all frames are used.

[nocovar] Do not calculate covariance.

[direct] Do not use FFTs to calculate correlation; this will be much slower.

DataSet Aspects:

[<**dataset1**>] (Auto-correlation) The aspect will be the name of each of the input data set.

[<**dataset1**>-<**dataset2**>] (Cross-correlation) The aspect will be the names of each of the input data sets joined by a dash ('-').

DataSet Aspects:

[coeff] Correlation coefficient.

Calculate the auto-correlation function for data set named <dataset1> or the cross-correlation function for data sets named <dataset1> and <dataset2> up to <lagmax> frames (all if **lagmax** not specified), writing the result to file specified by **out**. The two datasets must have the same # of datapoints.

12.7 crank | crankshaft

```
crank {angle | distance} <dsetname1> <dsetname2> info <string>
      [out <filename>] [results <resultsfile>]
```

angle Analyze angle data sets.

distance Analyze distance data sets.

<**dsetname1**> Data set to analyze.

<**dsetname2**> Data set to analyze.

info <**string**> Title the analysis <string>.

[out <**filename**>**]** Write frame-vs-bin to <filename>.

[results <**resultsfile**>**]** Write results to <resultsfile>.

Calculate crankshaft motion between two data sets.

12.8 crdfluct

```
[crdset <crd set>] [<mask>] [out <filename>] [window <size>] [bfactor]
```

Calculate atomic positional fluctuations for atoms in <**mask**> over windows of size <**size**>. If **bfactor** is specified, the fluctuations are weighted by $\frac{8}{3}\pi^2$(similar but not necessarily equivalent to crystallographic B-factor calculation). Units are Å, or $Å^2 x \frac{8}{3}\pi^2$ if **bfactor** specified.

12.9 crosscorr

```
crosscorr [name <dsetname>] <dsetarg0> [<dsetarg1> ...] [out <filename>]
```

[name <**dsetname**>**]** The resulting upper-triangle matrix
is stored with name <dsetname>.

<**dsetarg0**> **[**<**dsetarg1**> **...]** Argument(s) specifying
datasets to be used.

[out <**filename**>**]** Write results to file named
<filename>.

Calculate the Pearson product-moment correlation coefficients between all specified datasets.

12.10 curvefit

```
curvefit <dset> { <equation>  |
                  name <dsname> { gauss | nexp <m> [form {mexp|mexpk|mexpk_penalty} ] }
        [AX=<value> ...] [out <outfile>] [resultsout <results>]
        [maxit <max iterations>] [tol <tolerance>]
        [outxbins <NX> outxmin <xmin> outxmax <xmax>]
```

<dset> Data set to fit.

<equation> Equation to fit of form <Variable> = <Equation>. See 5.2 on page 22 for more details on equations *cpptraj* understands.

name <dsname> Final data set name (required if using nexp or gauss).

gauss Fit to Gaussian of form A0 * exp(-((X - A1)^2) / (2 * A2^2))

nexp <m> Fit to specified number of exponentials.

- **form <type>** Fit to specified exponential form:
  - **mexp** Multi-exponential, SUM(m)[An * exp(An+1 * X)]
  - **mexpk** Multi-exponential plus constant, A0 + SUM(m)[An * exp(An+1 * X)]
  - **mexpk_penalty** Same as mexpk except sum of prefactors constrained to 1.0 and exponential constants constrained to < 0.0.

AX=<value> Value of any constants in specified equation with X starting from 0 (can specify more than one).

out <outfile> Write resulting fit curve to <outfile>.

resultsout <results> Write details of the fit to <results> (default STDOUT).

maxit <max iterations> Number of iterations to run curve fitting algorithm (default 50).

tol <tolerance> Curve-fitting tolerance (default 1E-4).

outxbins <NX> Number of points to use when generating final curve (default same number of points as input data set).

outxmin <xmin> Minimum X value to use for final curve (default same number of points as input data set).

outxmax <xmax> Maximum X value to use for final curve (default same number of points as input data set).

Perform non-linear curve fitting for the specified data set using the Levenberg-Marquardt algorithm. Any equation form that cpptraj understands (see 5.2 on page 22) can be used, or several preset forms can be used. Similar to Grace (http://plasma-gate.weizmann.ac.il/Grace/), an equation can contain constants for curve fitting termed AX (with X being a numerical digit, one for each constant), and is assigned to a variable which then becomes a data set. For example, to fit a curve to data from a file named Data.dat to a data set named 'FitY':

```
readdata Data.dat
runanalysis curvefit Data.dat \
  "FitY = (A0 * exp(X * A1)) + (A2 * exp(X * A3))" \
  A0=1 A1=-1 A2=1 A3=-1 \
  out curve.dat tol 0.0001 maxit 50
```

To perform the same fit but to a multi-exponential curve with two exponentials:

```
readdata Data.dat
runanalysis curvefit Data.dat nexp 2 name FitY \
  A0=1 A1=-1 A2=1 A3=-1 \
  out curve1.dat tol 0.0001 maxit 50
```

12.11 diagmatrix

```
diagmatrix <name> [out <filename>] [thermo [outthermo <filename>]]
           [vecs <#>] [name <modesname>] [reduce]
           [nmwiz [nmwizvecs <#>] [nmwizfile <filename>]]
```

<name> Name of symmetric matrix to diagonalize.

[out <filename>] Write results to <filename>.

[thermo [outthermo <filename>]] Mass-weighted covariance (mwcovar) matrix only. Calculate entropy, heat capacity, and internal energy from the structure of a molecule (average coordinates, see above) and its vibrational frequencies using standard statistical mechanical formulas for an ideal gas. Results are written to <filename> if specified, otherwise results are written to STDOUT. Note that this converts the units of the calculated eigenvalues to frequencies (cm^{-1}).

[vecs <#>] Number of eigenvectors to calculate. Default is 0, which is only allowed when 'thermo' is specified.

[name <modesname>] Store resulting modes data set with name <modesname>.

[reduce] Covariance (covar/mwcovar/distcovar) matrices only. For coordinate covariance (covar/mwcovar) matrices, each eigenvector element is reduced via Ei = Eix^2 + Eiy^2 + Eiz^2. For distance covariance (distcovar) the eigenvectors are reduced by taking the sum of the squares of each row. See Abseher & Nilges, JMB 1998, 279, 911-920 for further details. They may be used to compare results from PCA in distance space with those from PCA in cartesian-coordinate space.

[nmwiz] Generate output in *.nmd* format file for viewing with NMWiz[26]. See http://prody.csb.pitt.edu/tutorials/nmwiz_tutorial/ for further details.

[nmwizvecs <#>] Number of vectors to write out for nmwiz output, starting with the lowest frequency mode (default 20).

[nmwizfile <filename>] Name of nmwiz file to write to (default 'out.nmd').

[nmwizmask <mask>] Mask of atoms corresponding to eigenvectors - should be the same one used to generate the matrix.

Calculate eigenvectors and eigenvalues for the specified symmetric matrix. This is followed by Principal Component Analysis (in cartesian coordinate space in the case of a covariance matrix or in distance space in the case of a distance-covariance matrix), or Quasiharmonic Analysis (in the case of a mass-weighted covariance matrix). Diagonalization of distance, correlation, idea, and ired matrices are also possible. Eigenvalues are given in cm^{-1} in the case of a mass-weighted covariance matrix and in the units of the matrix elements in all other cases. In the case of a mass-weighted covariance matrix, the eigenvectors are mass-weighted.

For quasi-harmonic analysis the input must be a mass-weighted covariance matrix. Thermodynamic quantities are calculated based on statistical mechanical formulae that assume the input system is oscillating in a single energy well: see Statistical Thermodynamics by D. A. McQuarrie, particularly chapters 4, 5, and 6 for more details.[27] For an in-depth discussion of the accuracy of thermodynamic parameters obtained via quasi-harmonic analysis see Chang et al..[28]

Note that the maximum number of non-zero eigenvalues obtainable depends on the number of frames used to generate the input matrix; the number of frames should be equal to or greater than the number of columns in the matrix in order to obtain all eigenmodes.

Results may include average coordinates (in the case of covar, mwcovar, correl), average distances (in the case of distcovar), main diagonal elements (in the case of idea and ired), eigenvalues, and eigenvectors.

For example, in the following a mass-weighted covariance matrix of all atoms is generated and stored internally with the name mwcvmat; the matrix itself is written to mwcvmat.dat. Subsequently, the first 20 eigenmodes of the matrix are calculated and written to evecs.dat, and quasiharmonic analysis is performed at 300.0 K, with the results written to thermo.dat.

```
matrix mwcovar name mwcvmat out mwcvmat.dat
diagmatrix mwcvmat out evecs.dat vecs 20 \
           thermo outthermo thermo.dat temp 300.0
```

Output Format

The "modes" or "evecs" output file is a text file with the following format:

```
[Reduced] Eigenvector file: <Type> nmodes <#> width <width>
  <# Avg Coords> <Eigenvector Size>
<Average Coordinates>
```

Where <Type> is a string identifying what kind of matrix the eigenvectors/eigenvalues were determined from, nmodes is how many eigenvectors are in the file, and <Average Coordinates> are in lines 7 columns wide, with each element having width specified by <width>. Then for each eigenvector:

```
 ****
<Eigenvector#> <Eigenvalue>
<Eigenvector Coordinates>
...
```

Where <Eigenvector Coordinates> are in lines 7 columns wide, with each element having width specified by <width>.

12.12 divergence

```
divergence ds1 <ds1> ds2 <ds2>
```

Calculate Kullback-Leibler divergence between specified data sets.

12.13 fft

```
fft <dset0> [<dset1> ...] [out <outfile>] [name <outsetname>] [dt <samp_int>]
```

<dset0> [<dset1 ...] Argument(s) specifying datasets to be used.

[out <outfile>] Write results to file named <outfile>.

[name <outsetname>] The resulting transform will be stored with name <outsetname>.

[dt <samp_int>] Set the sampling interval (default is 1.0).

Perform fast Fourier transform (FFT) on specified data set(s). If more than 1 data set, they must all have the same size.

12.14 hausdorff

```
hausdorff <set arg0> [<set arg1> ...]
          [outtype {basic|trimatrix nrows <#>|fullmatrix nrows <#> [ncols <#>]}]
       [name <output set name>] [out <file>] [outab <file>] [outba <file>]
```

<set arg0> ... Input matrix data set(s) to calculate Hausdorff distance(s) for.

[outtype] Specify the output type.

basic Output the Hausdorff distance for each input matrix as scalar 1D data.

trimatrix nrows <#> Output Hausdorff distances for each input matrix as a 2D upper-triangular matrix with the given number of rows. Must have (nrows * (nrows-1)) / 2 input sets.

fullmatrix nrows <#> **ncols** <#> Output Hausdorff distances for each input matrix as a full matrix with the given number of columns and rows. If ncols is not given, use nrows. Must have nrows * ncols input sets.

[name <output set name>] Name of output data sets.

[out <file>] File to write Hausdorff distances to.

[outab <file>] File to write directed A->B Hausdorff distances to.

[outba <file>] File to write directed B->A Hausdorff distances to.

Calculate the symmetric Hausdorff distance for one or more matrices. The results can be saved as an array or as a full or upper-triangular matrix with the specified dimensions. The Hausdorff distance H is determined from:

```
H = max{dH(A,B), dH(B,A)]
```

Where dH(A,B) is the directed Hausdorff distance between sets A and B, etc. Colloquially speaking, the directed Hausdorff distance between A and B is determined as follows:

1. What is the closest approach (distance) of each point in A to any point in B?

2. Choose the largest distance from among those distances.

If desired, the output can be formed into a matrix, which can be useful e.g. when doing multiple 2D rms calculations on different regions of a trajectory. For example, the following input divides a 100 frame trajectory into 10 frame chunks, calcultes the 2D RMS matrix for each chunk, then performs Hausdorff analysis on the resulting matrices and forms a full output matrix.

```
parm ../DPDP.parm7
for beg=1;beg<100;beg+=10 end=10;end+=10 i=1;i++
  loadcrd ../DPDP.nc \$beg \$end name Chunk\$i
done
# Do the 2drms in chunks
for i=1;i<11;i++
  for j=1;j<11;j++
    2drms crdset Chunk\$i reftraj Chunk\$j M\$i.\$j
  done
done
hausdorff M* out hausdorff.fullmatrix.gnu title hausdorff.matrix.gnu \
  outtype fullmatrix nrows 10
runanalysis
```

This type of calculation lends itself well to parallelization. The **parallelanalysis** command can be used to run all the **2drms** calculations in parallel with MPI-enabled cpptraj:

```
parm ../DPDP.parm7
for beg=1;beg<100;beg+=10 end=10;end+=10 i=1;i++
  loadcrd ../DPDP.nc \$beg \$end name Chunk\$i
done
# Do the 2drms in chunks
for i=1;i<11;i++
  for j=1;j<11;j++
    2drms crdset Chunk\$i reftraj Chunk\$j M\$i.\$j
  done
done
parallelanalysis sync
runanalysis hausdorff M* out hausdorff.fullmatrix.gnu title hausdorff.matrix.gnu \
  outtype fullmatrix nrows 10
```

12.15 hist | histogram

```
hist <dataset_name>[,<min>,<max>,<step>,<bins>] ...
[free <temperature>] [norm | normint] [gnu] [circular] out <filename>
[amd <amdboost_data>] [name <outputset name>]
    [traj3d <file> [trajfmt <format>] [parmout <file>]]
[min <min>] [max <max>] [step <step>] [bins <bins>] [nativeout]
```

<dataset_name>[,<min>,<max>,<step>,<bins>] Dataset(s) to be histogrammed. Optionally, the min, max, step, and/or number of bins can be specified for this dimension after the dataset name separated by commas. It is only necessary to specify the step or number of bins, an asterisk '*' indicates the value should be calculated from available data.

[free <temperature>] If specified, estimate free energy from bin populations using $G_i = -k_B T \ln\left(\frac{N_i}{N_{Max}}\right)$, where K_B is Boltzmann's constant, T is the temperature specified by <temperature>, N_i is the population of bin i and N_{Max} is the population of the most populated bin. Bins with no population are given an artificial barrier equivalent to a population of 0.5.

[norm] If specified, normalize bin populations so the sum over all bins equals 1.0.

[normint] Normalize bin populations so the integral over them is 1.0.

[gnu] Internal output only; data will be gnuplot-readable, i.e. a space will be printed after the highest order coordinate cycles.

[circular] Internal output only; data will wrap, i.e. an extra bin will be printed before min and after max in each direction. Useful for e.g. dihedral angles.

out <filename> Write results to file named <filename>.

[amd <amdboost_data>] Reweight bins using AMD boost energies in data set <amdboost_data> (in KT).

[name <outputset name>] Output histogram data set name.

[traj3d <file> [trajfmt <format>]] (3D histograms only) Write a pseudo-trajectory of the 3 data sets (1 atom) to <file> with format <format>.

[parmout <file>] (3D histograms only) Write a topology corresponding to the pseudo-trajectory to <file>.

[min <min>] Default minimum to bin if not specified.

[max <max>] Default max to use if not specified.

[step <step>] Default step size to use if not specified.

[bins <bins>] Default bin size to use if not specified.

[nativeout] Do not use cpptraj data file framework; only necessary for writing out histograms with > 3 dimensions.

Create an N-dimensional histrogram, where N is the number of datasets specified. For 1-dimensional histograms the xmgrace '.agr' file format is recommended; for 2-dimensional hisograms the gnuplot '.gnu' file format is recommended; for all other dimensions plot formatting is disabled and the routine uses its own internal output format; this is also enabled if **gnu** or **circular** is specified.

For example, to create a two dimensional histogram of two datasets 'phi' and 'psi':

```
dihedral phi :2@C :3@N :3@CA :3@C
dihedral psi :3@N :3@CA :3@C :4@N
hist phi,-180,180,*,72 psi,-180,180,*,72 out hist.gnu
```

In this case the number of bins (72) has been specified for each dimension and '*' has been given for the step size, indicating it should be calculated based on min/max/bins. The following 'hist' command is equivalent:

```
hist phi psi min -180 max 180 bins 72 out hist.gnu
```

12.16 integrate

```
integrate <dset0> [<dset1> ...] [out <outfile>] [name <outsetname>]
```

Integrate specified data set(s) using trapezoid integration.

12.17 ired

```
ired [relax freq <MHz> [NHdist <distnh>]] [order <order>]
  tstep <tstep> tcorr <tcorr> out <filename> [norm] [drct]
  modes <modesname> [name <output sets name>] [ds2matrix <file>]
```

[relax freq <MHz> [NHdist <distnh>]] Should only be used when ired vectors represent N-H bonds; calculate correlation times τ_m for each eigenmode and relaxation rates and NOEs for each N-H vector. 'freq <MHz>' (required) is the Lamor frequency of the measurement. 'NHdist <distnh>' specifies the length of the NH bond in Angstroms (default is 1.02).

order <order> Order of the Legendre polynomials to use when calculating spherical harmonics (default 2).

tstep <**tstep**> Time between snapshots in ps (default 1.0).

tcorr <**tcorr**> Maximum time to calculate correlation functions for in ps (default 10000.0).

out <**filename**> Name of file to write output to.

[norm] Normalize all correlation functions, i.e., $C_l(t=0) = P_l(t=0) = 1.0$.

[drct] Use the direct method to calculate correlations instead of FFT; this will be much slower.

modes <**modesname**> Name of previously calculated eigenmodes corresponding to IRED vectors.

[name <**name**>**]** Output data set name.

[ds2matrix <**file**>**]** If specified, write full delta*S^2 matrix (# IRED vector rows by # eigenmodes columns) to <file>.

DataSets Created:

<**name**>**[S2]** S2 order parameters for each vector.

<**name**>**[Plateau]** Plateau values for each vector.

<**name**>**[TauM]** TauM values for each vector.

<**name**>**[dS2]** Full delta*S^2 matrix.

<**name**>**[T1]** T1 relaxation values for each vector.

<**name**>**[T2]** T2 relaxation values for each vector.

<**name**>**[NOE]** NOEs for each vector.

<**name**>**[Cm(t)]:X** Cm(t) function for vector X.

<**name**>**[Cj(t)]:X** Cj(t) function for vector X.

Peform IRED[12] analysis on previously defined IRED vectors (see vector ired) using eigenmodes calculated from those vectors with a previous 'diagmatrix' command. The number of defined IRED vectors should match the number of eigenmodes calculated. Autocorrelation functions for each mode and the corresponding correlation time τ_m will be written to *filename.cmt*. Autocorrelation functions for each vector will be written to *filename.cjt*. Relaxation rates and NOEs for each N-H vector will be written to <filename> or added to the the end of the standard output. For the calculation of τ_m the normalized correlation functions and only the first third of the analyzed time steps will be used. For further information on the convergence of correlation functions see [Schneider, Brünger, Nilges, *J. Mol. Biol.* **285**, 727 (1999)].

Example of IRED in Cpptraj

In *cpptraj*, IRED analysis[12] can now be performed in one pass (as opposed to the two passes previously required in *ptraj*). First, IRED vectors are defined (in this case for N-H bonds) and an IRED matrix is calculated and analyzed. The IRED vectors are then projected onto the calculated IRED eigenvectors in the ***ired*** analysis command to calculate the time correlation functions. If the parameter *order* is specified, order parameters based on IRED are calculated. By specifying the *relax* parameter, relaxation rates and NOEs can be obtained for each N-H vector. Note that the order of the IRED matrix should be the same as the one specified for IRED analysis.

```
# Define N-H IRED vectors
vector v0 @5 ired @6
vector v1 @7 ired @8
...
vector v5 @15 ired @16
vector v6 @17 ired @18‘
# Define IRED matrix using all previous IRED vectors
matrix ired name matired order 2
# Diagonalize IRED matrix
diagmatrix matired vecs 6 out ired.vec name ired.vec
# Perform IRED analysis
ired relax NHdist 1.02 freq 500.0 tstep 1.0 tcorr 100.0 out v0.out noefile noe order 2
```

12.18 kde

```
kde <dataset> [bandwidth <bw>] [out <file>] [name <dsname>]
    [min <min>] [max <max] [step <step>] [bins <bins>] [free]
    [kldiv <dsname2> [klout <outfile>]] [amd <amdboost_data>]
```

[bandwidth <bw>] Bandwidth to use for KDE; if not specified bandwidth will be estimated using the normal distribution approximation.

[out <file>] Output file name.

[name <dsname>] Output data set name.

[min <min>] Minimum bin.

[max <max>] Maximum bin.

[step <step>] Bin step.

[bins <bins>] Number of bins.

[free] Calculate free energy from bin population.

[kldiv <dsname2> [klout <outfile>]] Calculate Kullback-Leibler divergence over time of <dataset> distribution to <dsname2> distribution. Output to <outfile> if klout specified.

[amd <amdboost_data>] Reweight histogram using AMD boost data from data set <amdboost_data> (in KT).

Histogram 1D data set using a Gaussian kernel density estimator.

12.19 lifetime

```
lifetime [out <filename>] <dsetarg0> [ <dsetarg1> ... ]
      [window <windowsize> [name <setname>]] [averageonly]
      [cumulative] [delta] [cut <cutoff>] [greater | less] [rawcurve]
         [fuzz <fuzzcut>] [nosort]
```

[out <filename>] Write results to file named <filename>, and lifetime curves to 'crv.<filename>'. If performing windowed lifetime analysis, <filename> contains the fraction present over time windows, and 2 additional files are written: 'max.<filename>', containing max lifetime over windows, and 'avg.<filename>', containing average lifetime over windows.

<dsetarg0> [<dsetarg1> ...] Argument(s) specifying datasets to be used.

[window <windowsize>] Size of window (in frames) over which to calculate lifetimes/averages. If not specified lifetime/average will be calculated over all frames.

[name <setname>] Store results in data sets with name <setname>.

[averageonly] Just calculate averages (no lifetime analysis).

[cumulative] Calculate cumulative lifetimes/averages over windows.

[delta] Calculate difference from previous window average.

[cut <cutoff>] Cutoff to use when determining if data is 'present' (default 0.5).

[greater] Data is considered present when above the cutoff (default).

[less] Data is considered present when below the cutoff.

[rawcurve] Do not normalize lifetime curves to 1.0.

[fuzz <fuzzcut>] Ignore changes in lifetime state that are less than <fuzzcut> frames.

[nosort] Do not sort data sets by name.

Data Sets Created:

<**setname**> Number of lifetimes for each set, or if window specified fraction present over time windows.

<**setname**>**[max]** Maximum lifetime for each set, or if window specified maximum lifetime over time windows.

<**setname**>**[avg]** Average lifetime for each set, or if window specified average lifetime over time windows.

<**setname**>**[curve]** Lifetime curves.

The following are created only if window not specified:

<**setname**>**[frames]** Total number of frames lifetime present for each set.

<**setname**>**[name]** Name of each set.

Perform lifetime analysis for specified data sets. Lifetime data can either be determined for the entire set, or for time windows of specified size within the set if **window** specified.

A "lifetime" is defined as the length of time something remains 'present'; data is considered present when above or below a certain cutoff (the default is greater than 0.5, useful for analysis of ***hbond*** time series data). For example, in the case of a hydrogen bond **'series'** data set, if a hydrogen bond is present during a frame the value is 1, otherwise it is 0. Given the hbond time series data set {1 1 1 0 1 0 0 0 1 1}, the overall fraction present is 0.6. However, there are 3 lifetimes of lengths 3, 1, and 2 ({1 1 1}, {1}, and {1 1}). The maximum lifetime is 3 and the average lifetime is 2.0, i.e. (3 + 1 + 2) / 3 lifetimes = 2.0. One can also construct a "lifetime curve", which is constructed as the sum of all individual lifetimes. By default these curves are normalized to 1.0, but the raw curve can be obtained using the **rawcurve** keyword. For the example data set here the raw lifetime curve would be 3 frames long:

```
       1 1 1
       1
       1 1
Curve: 3 2 1
```

By default data sets are sorted by name unless **nosort** is specified. The lifetime command can calculate lifetimes over specific time windows by using the **window** keyword. This can be particularly useful if one wants to get a sense for how lifetimes are changing over the course of very long time series data. In addition, averages can be calculated instead of lifetimes by specifying **averageonly**. Cumulative averages over windows can be obtained using the **cumulative** keyword, or the change from the average value in the previous window can be obtained using the **delta** keyword.

The **fuzz** keyword can be used to try and smooth the input data by ignoring changes in state that occur for fewer frames than <fuzzcut>. For example,

in the above example hbond time series data set there is a one frame change in state between the first and second lifetimes which could be interpreted as a transient breaking of the hydrogen bond. Using a <fuzzcut> value of 1, this one frame change in state would be ignored, and the data set would effectively appear to lifetime as {1 1 1 1 1 0 0 0 1 1}. The state change between the second and third lifetimes is longer than <fuzzcut> (3 frames) and so it would remain.

If **window** is not specified, two files are output: <filename> and crv.<filename>. The file <filename> contains overall lifetime stats for each set with format:

```
#Set <setname> <setname>[max] <setname>[avg] <setname>[frames] <setname>[name]
```

where `<setname>` denotes the total number of lifetimes, `<setname>[max]` denotes the maximum lifetime, `<setname>[avg]` denotes the average lifetime, `<setname>[frames]` denotes the total number of frames present in all lifetimes, and `<setname>[name]` is the data set name. The file crv.<filename> contains the lifetime curves for each set.

If **window** is specified, four files are output: <filename>, max.<filename>, avg.<filename>, and crv.<filename>. <filename> contains the fraction "present" over each time window for each set, max.<filename> contains the maximum lifetime in each time window for each set, avg.<filename> contains the average lifetime over each window for each set, and crv.<filename> contains the overall lifetime curves for each set. For window output, Gnuplot format is recommended.

Example: hbond lifetime analysis

```
parm DPDP.parm7
trajin DPDP.nc
hbond HB out hbond.dat @N,H,C,O series uuseries solutehb.agr \
  avgout hbavg.dat printatomnum
# 'run' is used here to process the trajectory and generate hbond data
run
# Perform lifetime analysis
runanalysis lifetime HB[solutehb] out lifehb.dat
```

Calculate ion lifetimes from hbond over windows of size 100 frames:

```
hbond ION out ion.dat solventdonor :WAT solventacceptor :WAT@O series
run
lifetime HB[solventhb] out ion.lifetime.100.gnu window 100
```

12.20 lowestcurve

```
lowestcurve points <# lowest> [step <stepsize>] <dset0> [<dset1> ...]
                   [out <file>] [name <setname>]
```

<# **lowest**> Number of lowest points in each bin to
average over.

[step <stepsize>] Bin step size

<dset0> [<dset1> ...] Data set(s) to use.

[out <file>] File to write lowest curve to.

[name <setname>] Output lowest curve set name.

Calculate a curve of the average of the # lowest points in bins of stepsize. Essentially each input data set is binned over bins of stepsize, then the lowest <#> points are averaged over for each bin.

12.21 meltcurve

```
meltcurve <dset0> [<dset1> ...] [out <outfile>] [name <outsetname>] cut <cut>
```

Calculate melting curve from input data sets (i.e. fraction 'folded' for each data set) assuming a simple 2-state transition model, using data below <cut>as 'folded' and data above <cut> as 'unfolded'.

12.22 modes

```
modes {fluct|displ|corr|eigenval|trajout|rmsip} name <modesname> [name2 <modesname>]
   [beg <beg>] [end <end>] [bose] [factor <factor>] [calcall]
   [out <outfile>] [setname <name>]
    Options for 'trajout': (Generate pseudo-trajectory)
   [trajout <name> parm <name> | parmindex <#>
     [trajoutfmt <format>] [trajoutmask <mask>]
    [pcmin <pcmin>] [pcmax <pcmax>] [tmode <mode>]]
    Options for 'corr': (Calculate dipole correlation)
   { maskp <mask1> <mask2> [...] | mask1 <mask> mask2 <mask> }
   parm <name> | parmindex <#>
Types of Calculations:
```

fluct RMS fluctuations (X, Y, Z, and total) for each atom across specified normal modes.

displ Displacement of cartesian coordinates in the X, Y and Z directions for each atom across specified normal modes.

corr Dipole-dipole correlation functions. Must also specify maskp (see below).

eigenval Calculate eigenvalue fractions.

trajout Create a pseudo-trajectory along the given mode from the average structure.

rmsip Calculate the root-mean-square inner product between modes specified by name and name2.

Options:

name <modesname> Previously read-in or generated Modes data set name.

[beg <beg>] [end <end>] If modes taken from datafile, beginning and end modes to read. Default for *beg* is 7 (which skips the first 6 zero-frequency modes in the case of a normal mode analysis); for *end* it is 50.

[bose] Use quantum (Bose) statistics in populating the modes.

[factor <factor>] multiplicative constant on the amplitude of displacement/pseudo-trajectory, default 1.0.

[calcall] If specified use all eigenvectors; otherwise eigenvectors associated with zero or negative eigenvalues will be skipped.

[out <outfile>] File to write data results to. If not given results are written to STDOUT.

[setname <name>] Output data set name.

Options for 'trajout':

<name> Output trajectory file name.

[parm <parmfile/tag>|parmindex <#>] Topology file to use (default first Topology loaded).

[trajoutfmt <format>] Output trajectory format.

[trajoutmask <mask>] Mask of atoms that correspond to how modes were originally generated.

[pcmin <pcmin>] Lowest principal component projection value to use for output trajectory.

[pcmax <pcmax>] Highest principal component projection value to use for output trajectory.

[tmode <mode>] Mode to generate pseudo-trajectory for.

Options for 'corr':

[maskp <mask1> <mask2> [...]] If corr, pairs of atom masks (*mask1*, *mask2*; each pair preceded by "maskp" and each mask defining only a single atom) have to be given that specify the atoms for which the correlation functions are desired.

mask1 <**mask**> **mask2** <**mask**> Instead of maskp, specify two masks; atoms from the first mask will be paired up with atoms from the second mask.

DataSets Created (fluct)

<**name**>**[rmsX]** RMS fluctuations in the X direction.

<**name**>**[rmsY]** RMS fluctuations in the Y direction.

<**name**>**[rmsZ]** RMS fluctuations in the Z direction.

<**name**>**[rms]** Total RMS fluctuations.

DataSets Created (displ)

<**name**>**[displX]** Displacement in X direction.

<**name**>**[displY]** Displacement in Y direction.

<**name**>**[displZ]** Displacement in Z direction.

DataSets Created (eigenval)

<**name**>**[Frac]** Fraction eigenvalue contributes to overall motion.

<**name**>**[Cumulative]** Cumulative fraction.

<**name**>**[Eigenval]** Value of eigenvlue.

DataSets Created (rmsip)

<**name**> Result of RMSIP calculation.

Analyze previously calculated eigenmodes obtained from principal component analyses (of covariance matrices) or quasiharmonic analyses (diagmatrix analysis command). Modes are taken from a previously generated data set (i.e. from ***diagmatrix***) or read in from a data file with ***readdata***. By default, classical (Boltzmann) statistics are used in populating the modes. A possible series of commands would be "**matrix** `covar | mwcovar` ..." to generate the matrix, "`diagmatrix` ..." to calculate the modes, and, finally, "`modes` ...".

For example, to calculate the RMS fluctuations or displacements of the first 3 eigenmodes caluclated from a mass-weighted covariance matrix:

```
matrix mwcovar name mwcvmat out mwcvmat.dat
diagmatrix mwcvmat name evecs vecs 5
modes fluct out rmsfluct.dat name evecs beg 1 end 3
modes displ out resdispl.dat name evecs beg 1 end 3
```

Additionally, dipole-dipole correlation functions for modes obtained from principle component analysis or quasiharmonic analysis can be computed.

```
modes corr out cffromvec.dat name evecs beg 1 end 3 \
      maskp @1 @2 maskp @3 @4 maskp @5 @6
```

or

```
mode corr out cffromvec.dat name evecs beg 1 end 3 mask1 @1,3,5 mask2 @2,4,6
```

If **eigenval** is specified, the fraction contribution of each eigenvector to the total motion is calculated and output with format:

```
#Mode Frac. Cumulative Eigenval
```

where `#Mode` is the eigenvector number, `Frac.` is the eigenvalue over the sum of all eigenvalues, `Cumulative` is the cumulative sum of `Frac.`, and `Eigenval` is the eigenvalue itself. Note that in order to get an idea for how much each eigenvector contributes to all motion, this is best used when all possible eigenvectors have been determined for a system.

In order to visualize eigenvectors, pseudo-trajectories along eigenvectors can be created using average coordinates with the **trajout** keyword. For example, to write a pseudo-trajectory of the first principal component from principal component value of -100 to 100 for a previously calculated Modes data set corresponding to heavy atoms (no hydrogens) for residues 1 to 36:

```
parm ../GAAC.nowat.parm7
readdata evecs.dat
runanalysis modes name evecs.dat trajout test.nc trajoutfmt netcdf \
            trajoutmask :1-36&!@H= pcmin -100 pcmax 100 tmode 1
```

12.23 multicurve

```
multicurve set <dset> [set <dset> ...]
           <dset> { <equation> |
                    name <dsname> nexp <m> [form {mexp|mexpk|mexpk_penalty} }
           [AX=<value> ...] [out <outfile>] [resultsout <results>]
           [maxit <max iterations>] [tol <tolerance>]
           [outxbins <NX> outxmin <xmin> outxmax <xmax>]
```

set <**dset**> **[set** <**dset**> **...]** Data set(s) to fit.

<**equation**> Equation to fit of form <Variable> = <Equation>. See 5.2 on page 22 for more details on equations *cpptraj* understands.

name <**dsname**> Name of output data sets (required if using nexp).

nexp <**m**> Fit to specified number of exponentials.

form <**type**> Fit to specified exponential form:

- **mexp** Multi-exponential, SUM(m)[An * exp(An+1 * X)]
- **mexpk** Multi-exponential plus constant, A0 + SUM(m)[An * exp(An+1 * X)]

mexpk_penalty Same as mexpk except sum of prefactors constrained to 1.0 and exponential constants constrained to < 0.0.

AX=<value> Value of any constants in specified equation with X starting from 0 (can specify more than one).

out <outfile> Write resulting fit curve to <outfile>.

resultsout <results> Write details of the fit to <results> (default STDOUT).

maxit <max iterations> Number of iterations to run curve fitting algotrithm (default 50).

tol <tolerance> Curve-fitting tolerance (default 1E-4).

outxbins <NX> Number of points to use when generating final curve (default same number of points as input data set).

outxmin <xmin> Minimum X value to use for final curve (default same number of points as input data set).

outxmax <xmax> Maximum X value to use for final curve (default same number of points as input data set).

Fit each input data set <dset> to <equation>. See the ***curvefit*** command on page 181 for more details.

12.24 multihist

```
multihist [out <filename>] [name <dsname>] [norm | normint] [kde]
          [min <min>] [max <max>] [step <step>] [bins <bins>] [free <T>]
          <dsetarg0> [ <dsetarg1> ... ]
```

out <filename> Output file.

name <dsname> Name for resulting histogram data sets.

norm (Only used if not kde) Normalize so that max bin is 1.0.

normint (Default for kde) Normalize integral over histogram to 1.0.

kde Use kernel density estimator to construct histogram.

min <min> Histogram minimum (default data set minimum).

max <**max**> Histogram maximum (default data set maximum).

step <**step**> Histogram step.

bins <**bins**> Number of histogram bins.

free <**T**> Calculate free energy from bin populations as G = -R * <T> * ln(Ni / Nmax).

<**dsetargX**> Data set argument - may specify more than one.

Histogram each data set separately in 1D. Must specify at least **bins** or **step**.

12.25 phipsi

```
phipsi <dsarg0> [<dsarg1> ...] resrange <range> [out <file>]
```

<**dsargX**> Argument selecting data sets. Can specify more than 1.

resrange <**range**> Residue range to use (actually uses data set index).

[out <**file**>**]** Output file.

Calculate the average and standard deviation of [phi] and [psi] data set pairs, write to <file> with format:

```
#Phi Psi SD(Phi) SD(Psi) Legend
```

Where Phi is the average value of phi, Psi is the average value of psi, SD(Phi) is the standard deviation of phi, SD(psi) is the standard deviation of psi, and Legend contains text describing the phi and psi data sets used in the calculation. Periodicity is taken into account during averaging. The data sets must have been internally labeled as type 'phi'/'psi' and must have a data set index set (actions like dihedral and multidihedral do this automatically). For example:

```
parm ../DPDP.parm7
trajin ../DPDP.nc
multidihedral DPDP phi psi
run
phipsi DPDP[phi] DPDP[psi] out phipsi.dat resrange 1-22
```

12.26 regress

```
regress <dset0> [<dset1> ...] [name <name>] [nx <nxvals>]
     [out <filename>] [statsout <filename>]
```

dsetX Data set(s) to perform linear regression for.

name <**name**> Data set name for resulting linear fits.

nx <**nxvals**> Number of X values to use in output data set(s) (ranging from input set min to max X). If not specified, input X values used.

out <**filename**> File to write fit lines to.

statsout <**filename**> File to write fit statistics to.

DataSets Generated:

<**name**>:<**idx**> Output fit line(s) (indexed by input set order if more than one input set).

<**name**>**[slope]**:<**idx**> Output fit line slope(s).

<**name**>**[intercept]**:<**idx**> Output fit line intercept(s).

Perform linear regression on the specified data set(s). The fit line is calculated using either the input X values or <**nxvals**> values ranging from the input set minimum to maximum X. Statistics for the fit(s) are saved to the file specified by **statsout** or reported to STDOUT.

For example, to fit data read in from a file and then create a set using the fit parameters:

```
readdata esurf_vs_rmsd.dat.txt index 1 name XY
runanalysis regress XY name FitXY statsout statsout.dat
createset "Y = FitXY[slope] * X + FitXY[intercept]" xstep .2 nx 100
writedata Y.dat Y
```

12.27 remlog

```
remlog {<remlog dataset> | <remlog filename>} [out <filename>] [crdidx | repidx]
    [stats [statsout <file>] [printtrips] [reptime <file>]] [lifetime <file>]
    [reptimeslope <n> reptimeslopeout <file>] [acceptout <file>] [name <setname>]
       [edata [edataout <file>]]
```

<**remlog dataset**> Previously read-in REM log data.

<**remlog filename**> REM log file name to read in.

[out <**filename**>**]** Write replica/coordinate index versus time to <filename>.

crdidx Print coordinate index vs exchange; output sets contain replica indices.

repidx Print replica index vs exchange; output sets contain coordinate indices.

stats [statsout <**file**>**]** Calculate round-trip statistics and optionally write to <file>.

printtrips Print details of each individual round trip.

[reptime <file>] Write time spent at each replica to <file>.

[lifetime <file>] Print lifetime data at each replica to <file>.

[reptimeslope <n>] Calculate the slope of time spent at each replica every <n> exchanges.

[reptimeslopeout <file>] File to write reptimeslope output to.

[acceptout <file>] Write overall exchange acceptances to <file>.

[name <setname>] Output data set name.

[edata [edataout <file>]] Extract energy data from replica log, optionally write to file.

DataSets created:

<setname>:<idx> Replica/coordinate index vs exchange.

<setname>[E]:<idx> If 'edata' specified, energy data from replica log.

Analyze previously read in (via ***readdata***) M-REMD/T-REMD/H-REMD replica log data. Statistics calculated include round-trip time, which is the time needed for a coordinate set to travel from the lowest replica to the highest and back, and the number of exchanges each coordinate spent at each replica. For example, to read in REM log data from an Amber M-REMD run and analyze it:

```
readdata rem.log.1.save rem.log.2.save dimfile remd.dim as remlog nosearch
remlog rem.log.1.save stats reptime mremdreptime.dat
```

For an example of ***remlog*** analysis applied to actual REMD data, see Roe et al.[29].

12.28 rms2d | 2drms

```
rms2d [crdset <crd set>] [<name>] [<mask>] [out <filename>]
      [dme | nofit | srmsd] [mass]
      [reftraj <traj> [parm <parmname> | parmindex <parm#>] [<refmask>]]
      [corr <corrfilename>]
```

[crdset <crd set>] Name of previously generated COORDS DataSet. If not specified the default COORDS set will be used.

[<mask>] Mask of atoms to calculate 2D-RMSD for. Default is all atoms.

[out <filename>] Write results to <filename>.

[dme] Calculate distance RMSD instead of coordinate RMSD; this is substantially slower.

[nofit] Calculate RMSD without fitting.

[srmsd] Calculate symmetry-corrected RMSD (see 11.78 on page 154).

[mass] Mass-weight RMSD.

[reftraj <traj>] Calculate 2D RMSD to frames in trajectory <traj> instead (can also be another COORDS set).

[parm <parmname> | parmindex <#>] Topology to use for <traj>; only useful in conjunction with reftraj.

[<refmask>] Mask of atoms in reference; only useful in conjunction with reftraj.

[corr <corrfilename>] Calculate pseudo-auto-correlation C for 2D-RMSD as $C(i) = \frac{\sum_{j=0}^{j<N-i} \exp(-RMSD(j,j+i))}{N-i}$, where i is the lag, j is the frame #, and N is the total number of frames. An exponential is used to weight the RMSD since 0.0 RMSD is equivalent to correlation of 1.0. This can only be done if reftraj is not used.

DataSet Aspects:

[Corr] (corr only) Pseudo-auto-correlation.

Note: For backwards compatibility with ptraj the command '2drms' will also work.

Calculate the best-fit RMSD of each frame in <crd set> (the default COORDS set if none specified) to each other frame. This creates an upper-triangle matrix named <name> (or a full matrix if **reftraj** specified). The output of the rms2d command can be best-viewed using gnuplot; a gnuplot-formatted file can be produced by giving <filename> a '.gnu' extension. For example, to calculate the RMSD of non-hydrogen atoms of each frame in trajectory "test.nc" to each other frame, writing to a gnuplot-viewable file "test.2drms.gnu":

```
trajin test.nc
rms2d !(@H=) out test.2drms.gnu
```

To calculate the RMSD of atoms named CA of each frame in trajectory "test.nc" to each frame in "ref.nc" (assuming test.nc and ref.nc are using the default topology file):

```
trajin test.nc
rms2d @CA out test.2drms.gnu reftraj ref.nc
```

12.29 rmsavgcorr

```
rmsavgcorr [crdset <crd set>] [<name>] [<mask>] [out <filename>] [mass]
        [stop <maxwindow>] [offset <offset>]
        {reference <ref file> parm <parmfile> | first}
```

[crdset <crd set>] COORDS data set to use (if not specified the default COORDS set will be used).

[<name>] Output data set name.

[<mask>] Atoms to calculate RMS average correlation for.

[out <filename>] Output filename.

[mass] Mass weight the RMSD calculation.

[stop <maxwindow>] Only calculate RMS average correlation up to <maxwindow>.

[offset <offset>] Skip every <offset> windows in calculation.

[first] Use first averaged frame as reference for each window (default).

[reference <ref file> [parm <parmfile>] Use reference file (with specified parm) as reference for each window.

The RMS average correlation[1] (RAC) is calculated as the average RMSD of running-averaged coordinates over increasing window sizes (or lag). Output has format:

```
<WindowSize> <RAC>
```

The first entry has a window size of 1, and so is just the average RMSD of all frames to the specified reference structure. The second entry has a window size of two, so it is the average RMSD of all frames averaged over two adjacent windows to the specified reference, and so on. The RAC will be calculated up to the number of frames minus 1 or the value specified by **stop**, whichever is lower. The offset can be used to speed up the calculation by skipping window sizes. To calculate mass-weighted RMSD specify **mass**. Note that to reduce memory costs it can be useful to strip all coordinates not involved in the RMS fit from the system prior to specifying 'rmsavgcorr'. For example, to calculate the correlation of C-alpha RMSD of residues 2 to 12:

```
strip !(:2-12@CA)
rmsavgcorr out rmscorr.dat
```

The curve generated by RAC decays towards zero due to the way RAC is defined. By the time the "lag" is N-1 (where N is the total number of frames) you have only two averaged coordinates: call them Avg1 (averaged over 1 though N-1 frames) and Avg2 (averaged over 2 through N frames). Barring any extraordinary circumstances the RMSD between Avg1 and Avg2 will almost certainly be quite low.

The RAC is a way to probe the time scales of interesting events. Any deviation from a smoothly decaying curve is an indication that there are some significant structural differences occurring over that time interval. RAC curves can be particularly useful when comparing independent simulations of the same system.

One thing to keep in mind that since the underlying metric is RMSD, it can be sensitive to the reference frame you choose. It may be useful to try looking at both RAC from the first frame, as well as an averaged reference frame. For an example of use see Galindo-Murillo et al.[30], in particular Figure 2.

12.30 rotdif

```
rotdif [outfile <outfilename>] [usefft]
  Options for generating random vectors:
[nvecs <nvecs>] [rvecin <randvecIn>] [rseed <random seed>]
[rvecout <randvecOut>] [rmatrix <set name> [rmout <rmOut>]]
  Options for calculating vector time correlation functions:
[order <olegendre>] [ncorr <ncorr>] [corrout <corrOut>]
  *** The options below only apply if 'usefft' IS NOT specified. ***
  Options for calculating local effective D, small anisotropy:
[deffout <deffOut>] [itmax <itmax>] [tol <tolerance>] [d0 <d0>]
[nmesh <NmeshPoints>] dt <tfac> [ti <ti>] tf <tf>
  Options for calculating D with full anisotropy:
[amoeba_tol <tolerance>] [amoeba_itmax <iterations>]
[amoeba_nsearch <n>] [scalesimplex <scale>] [gridsearch]
  *** The options below only apply if 'usefft' IS specified. ***
  Options for curve-fitting:
[fit_tol <tolerance>] [fit_itmax <max # iterations>]
```

outfile <**outfilename**> File to write all output from rotdif command to.

Options for generating random vectors:

nvecs <**nvecs**> Number of random vectors to generate (default 1000).

rvecin <**randvecIn**> File to read random vectors from (format is 1 per line, 4 columns, <#> <VX> <VY> <VZ>).

rseed <**random seed**> Seed for random number generator (default 80531). Specify -1 to use wallclock time.

rvecout <**randvecOut**> File to write random vectors to (format is 1 per line, 4 columns, <#> <VX> <VY> <VZ>).

rmatrix <**set name**> Data set to read rotation matrices from. Rotation matrices will be used to rotate random vectors.

rmout <**rmOut**> Write rotation matrices to file, 1 per line, frame # followed by matrix in row-major order.

Options for calculating vector time correlation functions:

order <**olegendre**> The order of Legendre polynomials to use when calculating vector time correlation functions (default 2).

ncorr <**ncorr**> Maximum length of time correlation functions in frames. If this is not specified it will be set to (tf - ti) / dt (recommended).

corrout <**corrOut**> If specified write vector time correlation functions to <corrOut>.X with format: <Time> <Px>

Options for calculating local effective D, small anisotropy:

deffout <**deffOut**> File to write out local effective diffusion constants determined in the limit of small anisotropy.

itmax <**itmax**> Maximum number of iterations to determine each local effective diffusion constant (small anisotropy) assuming fit to single exponential form (default 500).

tol <**tolerance**> Tolerance for determining local effective diffusion constant (small anisotropy) assuming fit to single exponential form (default 1E-6).

d0 <**d0**> Initial guess for small anisotropy diffusion constant in radians^2/ns (default 0.03).

nmesh <**NmeshPoints**> Number of points per frame to use when creating cubic-splined-smoothed forms of vector time correlation curves (default 2).

dt <**tfac**> Time interval between frames (used in integrating vector time correlation curves) in ns.

ti <**ti**> Initial time value in ns for integrating the time correlation functions (default 0.0).

tf <**tf**> Final time value in ns for integrating the time correlation functions. It is recommended this be less than the maximum simulation time since the tails of time correlation functions tend to be noisy.

Options for calculating D with full anisotropy:

amoeba_tol <**tolerance**> Tolerance for downhill-simplex minimizer (default 1E-7).

amoeba_itmax <**iterations**> Number of iterations to run downhill-simplex minimizer (default 10000).

amoeba_nsearch <**n**> Number of searches to perform with downhill-simplex minimizer (default 1).

scalesimplex <**scale**> Factor to use when scaling simplexes (default 0.5).

gridsearch If specified, perform a brute-force grid search to attempt to find a better solution for diffusion tensor with full anisotropy (may be expensive).

Evaluate rotational diffusion properties of a molecule over a trajectory according to an expanded version of the procedure laid out by Wong & Case[31]. Briefly, random vectors (representing the orientation of the molecule) are rotated according to rotation matrices obtained from an RMS fit to a reference structure (typically an averaged structure). For each random vector the time correlation function of the rotated vector is calculated using Legendre polynomials of the specified order. The integral over this time correlation function (which may be smoothed using cubic splines to improve the integration) is then used to find the effective diffusion constant (D) in the limit of small anisotropy. Then, using each calculated D, the diffusion tensor is determined with full anisotropy. Finally, a downhill simplex minimizer is used to optimize D with full anisotropy; (this last step is not described in the original paper).

Rotation matrices are generated via an RMS fit to a reference structure (see 11.64 on page 142). It is recommended that the RMS fit be done to an average structure (see 11.8 on page 77). These rotation matrices are used to rotate each random vector M times (where M is the total number of frames), which creates a time series for each random vector. The time correlation functions are calculated for each random vector time series using Legendre polynomials of the specified order (default 2). The maximum length of the correlation function (or lag) can be specified by **ncorr** (in frames). If **ncorr** is not specified it will be set internally based on the specified values of **ti**, **tf**, and **dt**; this is recommended. Note that if **ncorr** is specified it should be set to a number less than the total number of frames since noise in time correlation functions increases as **ncorr** approaches the # of frames. The integration over the correlation function is from **ti** (in whatever units are used of **dt**, generally ns; 0.0 ns if not specified)

to **tf** (same units as **ti**), with the time between frames specified by **dt**; the final time should be less than the total simulation time (see example below).

The relative size of the mesh used with cubic spline interpolation for integration is controlled by **nmesh** (size of the mesh is **ncorr** points * **nmesh**); **nmesh** = 1 means no interpolation, default is 2. The iterative solver for effective value of the diffusion constant from the correlation functions is controlled by **itmax**, **tol**, and **d0**, where **itmax** specifies the number of iterations to perform (default 500), **tol** specifies the tolerance (default 1E-6), and **d0** specifies the initial guess for the diffusion constant in radians^2 / ns (default 0.03). Effective diffusion constants for each random vector can be written out to a file specified by **deffout**. Results are printed to the file specified by **outfile**. Details on the Q and D tensors are given, as well as observed and calculated tau for each random vector. First, results are printed for analysis in the limit of small anisotropy. Next, results are printed for analysis with full anisotropy. The results of the full anisotropic calculation are first given using results from the small anisotropic analysis as an initial guess, followed by the final results after minimization using the downhill simplex (amoeba) minimizer.

Example

There are two important things to keep in mind when using rotdif analysis:

1. When calculating any kind of diffusive property it is best to simulate in the microcanonical (NVE) ensemble with a shorter time step and increased SHAKE tolerance; thermostats and barostats will effect diffusion calculations.

2. Time correlation functions become noisier as the length of the function approaches the maximum. Therefore in general one should choose parameters for the time correlation function that are much shorter than the total simulation length.

For example, given a trajectory 'mdcrd.nc' containing 10000 frames with a total simulation time of 200 ns (so the time between frames is 0.02 ns), to calculate rotational diffusion using 100 vectors using rotation matrices generated via an RMS fit to 'avgstruct.pdb', computing and integrating the time correlation function for each vector from 0 to 5 ns (1/40th of the simulation), and writing out the effective diffusion constants and results to 'deffs.dat' and 'rotdif.out' respectively:

```
reference avgstruct.pdb [avg]
rms R0 @CA,C,N,O ref [avg] savematrices
trajin mdcrd.nc
rotdif nvecs 100 rmatrix R0[RM] \
       ti 0.0 tf 5.0 dt 0.02 deffout deffs.dat \
       outfile rotdif.out
```

12.31 runningavg

```
runningavg <dset1> [<dset2> ...] [name <dsetname>] [out <filename>]
                   [ [cumulative] | [window <window>] ]
```

<dset1> [<dset2> ...] Data set(s) to calculate running average for.

[name <dsetname>] Output running average data set name.

[out <filename>] File to write results to.

[cumulative] Calculate cumulative running average instead.

[window <window>] Size in frames of window over which to calculate running average.

Calculate running average over windows of given size for data in selected data set(s).

12.32 spline

```
spline <dset0> [<dset1> ...] [out <outfile>] [meshsize <n> | meshfactor <x>]
[meshmin <mmin>] [meshmax <mmax>]
```

<dsetX> Data set(s) to perform splining on.

[out <outfile>] Write splined data to <outfile>.

[meshsize<n>] Size of the mesh to use for splining.

[meshfactor <x>] If meshsize is not given, use a mesh of data set size * <x>.

[meshmin <mmin>] Mesh X minimum value.

[meshmax <mmax>] Mesh X maximum value.

Cubic spline the given data sets.

12.33 statistics | stat

```
stat {<name> | ALL} [shift <value>] [out <filename>] [noeout <filename>]
 [ignorenv] [name <noe setname>]
```

<name> Name of data set to analyze.

ALL analyze all data sets.

shift <value> Subtract <value> from all elements in each data set.

[out <filename>] Write analysis results to <filename> (STDOUT if not specified).

[noeout <filename>] (Type 'noe' only) Write summary of NOE results to <filename>.

[ignorenv] (Type 'noe' only) Ignore negative NOE violations (i.e. shorter-than-expected distances).

[name <noe setname>] (Type 'noe' only) Name for output NOE data sets.

DataSet Aspects for type 'noe' output:

[R6] Averaged $1/r^6$distance for each set.

[NViolations] Number of violations based on given bounds for each set.

[AvgViolation] $1/r^6$ averaged distance minus expected distance for each set.

[NOEnames] Name of each set.

Analyze angles, dihedrals, distances, and/or puckers and calculate various properties. More specific analyses can be obtained by labelling distances/dihedrals/puckers (from e.g. the ***distance***, ***dihedral***, ***pucker*** commands or with the ***dataset*** command) with the 'type <label>' keyword:

dihedral type labels: alpha, beta, gamma, delta, epsilon, zeta, chi, c2p h1p, phi, psi, omega, pchi

distance type labels: noe

pucker type labels: pucker

For each input data set, the average, standard deviation, initial and final values will be reported. The cyclic nature of dihedral/pucker data sets is taken into consideration when averaging.

12.33.1 Torsion Analysis

A table will be written in ASCII format showing the distribution of torsion values for each data set. More specific information may be printed based on the set type. Values in the output marked SNB are from those defined by Schneider, Neidle, and Berman.[32] For more information on nucleic acid torsion as pertains to RNA see further work by Schneider et al..[33]

For example, to perform in-depth analysis on some nucleic acid dihedral angles:

```
dihedral g0 out dihedrals.dat :1@O5' :1@C5' :1@C4' :1@C3' type gamma
dihedral d0 out dihedrals.dat :1@C5' :1@C4' :1@C3' :1@O3' type delta
dihedral c0 out dihedrals.dat :1@O4' :1@C1' :1@N9 :1@C4 type chi
analyze statistics all out stat.dat
```

12.33.2 Distance Analysis

A table will be written in ASCII format showing the distribution of distance values < 6.5. If a distance is labled as 'type noe' a compact time series will be printed in ASCII format showing the NOE as strong, medium, or weak. In addition the <r^-6>^(-1/6) averaged value will be reported, as well as the number of upper/lower bound violations. If **'noeout'** is specified, a summary of these results will be written with format:

```
<#NOE> <R6> <Nviolation> <AvgViolation> <Name>
```

Where <#NOE> is an index, <R6> is the <r^-6>^(-1/6) averaged distance, <Nviolation> is the total number of bounds violations, <AvgViolation> is the average difference from expected distance *Rexp* when the distance is violated (note that if not explicitly set, *Rexp* is set to the upper bound when the lower bound is 0.0, or the average of upper and lower bounds otherwise), and <Name> is the data set legend.

For example, the following input could be used to check certain distances for NOE violations:

```
distance :3@HB= :10@HG= type noe noe_medium
distance :3@HE= :10@HG= type noe noe_strong
distance :3@HA :12@HA type noe noe_medium
distance :3@HD= :12@HG= type noe noe_medium
distance :3@HE= :12@HA type noe noe_strong
analyze statistics all out dpdp.noe.dat noeout noe_graph.dat name Res3_NOE
```

12.33.3 Pucker Analysis

A table will be written in ASCII format showing the distribution of pucker phases for each data set.

12.34 ti

```
ti <dset0> [<dset1> ...] {nq <n quad pts> | xvals <x values>}
[name <set name>] [out <file>] [curveout <ti curve file>]
[nskip <#s to skip>]
[avgincrement <#> [avgmax <#>] [avgskip <#>]]
[bs_samples <samples> [bs_points <points>] [bs_seed <#>]
 [bs_fac <factor>]]
```

<dset0> [<dset1> ...] Data set arguments specifying input DV/DL values.

nq <n quad pts> Number of points for Gaussian quadrature integration. Expect one data set per point.

xvals <**x values**> Comma-separated list of X values for integration. Expect one data set per value.

name <**set name**> Output data set name.

out <**file**> File to write results of integration to.

curveout <**ti curve file**> File to write TI curves to.

nskip <**#s to skip**> Comma separated list of number of points to skip. For each number given, the TI integration will be repeated.

avgincrement <#> **[avgmax** <#>**]** **[avgskip** <#>**]**
Starting from point 'avgskip' (default 0), repeat the TI integration calculation in increments of <#> up to 'avgmax' (default all points), so 'avgincrement 10' will do points 0-10, 0-20, etc.

bs_samples <**samples**> **[bs_points** <**points**>**]** **[bs_seed**<#>**]** **[bs_fac** <**factor**>**]**
Estimate error via bootstrap analysis, repeating the TI integration <samples> times using <points> points or <factor> times the total number of points. Randomize with given seed.

DataSet Aspects:

[TIcurve] Raw TI curve. If 'nskip' index is number of points skipped. If bootstrapping, index is sample index. If 'avgincrement' the index is the number of points.

[SD] For bootstrap analysis, standard deviation of average free energy over samples.

Calculate free energy using DV/DL energies from thermodynamic integration. The results of integration of the DV/DL curve will be written to <**file**>, while the curves themselves will be written to <**ti curve file**>. Use **nq** to specify number of Gaussian quadrature points; otherwise the lambda values should be specified by **xvals**, where <**x values**> is a comma-separated list.

For example, to perform Gaussian quadrature integration using data sets named 'TIdata', repeating the calculation for various number of skipped data points:

```
ti TIdata nq 9 name Curve out skip.agr curveout curve.agr nskip 0,5,10,15,20,30,40,50
```

12.35 timecorr

```
timecorr vec1 <vecname1> [vec2 <vecname2>] out <filename>
          [order <order>] [tstep <tstep>] [tcorr <tcorr>]
          [dplr] [norm] [drct] [dplrout <dplrfile>] [ptrajformat]
```

vec1 <**vecname1**> **[vec2** <**vecname2**>**]** Vector(s) on which to operate. By default the auto-correlation function will be calculated if one vector is specified, and the cross-correlation function will be calculated if two vectors are specified.

out <**filename**> Name of file to write output to.

[order <**order**>**]** Order of Legendre polynomials to use; default 2.

[tstep <**tstep**>**]** Time between snapshots (default 1.0).

[tcorr <**tcorr**>**]** Maximum time to calculate correlation functions for (default 10000.0).

[dplr] Output correlation functions $C_l \equiv < P_l/(r(0)^3 r(\tau)^3) >$ and $< 1/(r(0)^3 r(\tau)^3) >$ in addition to the P_l correlation function.

[norm] Normalize all correlation functions, i.e., $C_l(t=0) = P_l(t=0) = 1.0$.

[drct] Use the direct method to calculate correlations instead of FFT; this will be much slower.

[dplrout] (dplr only) Write extra information for each vector related to dplr option to <dplrfile>.

[ptrajformat] Write output in ptraj style (prevents use of data formatting options).

DataSet Aspects:

[P] P<order> correlation function.

[C] C<order> correlation function (dplr only).

[R3R3] <1/(r(0)^{3}r(t)^3> correlation function (dplr only).

[R] (_TC_DIPOLAR_) Average magnitude (<R>).

[RRIG] (_TC_DIPOLAR_) Sqrt(<R^2>).

[R3] (_TC_DIPOLAR_) <1/R^3>.

[R6] (_TC_DIPOLAR_) <1/R^6>.

[Name] (_TC_DIPOLAR_) Vector name.

Calculate time auto/cross-correlation functions for vectors using spherical harmonics theory. NOTE: To calculate direct correlation functions for vectors just use the ***corr*** analysis command. The **norm** keyword will normalize the resulting correlation functions. Note that if **dplr** is specified, a new global data set named _TC_DIPOLAR_ will be created, containing extra data for each vector analyzed with a '***timecorr* dplr**' command.

Examples

Vectors between atoms 5 and 6 as well as 7 and 8 are calculated below, for which auto and cross time correlation functions are obtained.

```
vector v0 @5 @6
vector v1 @7 @8
timecorr vec1 v0 tstep 1.0 tcorr 100.0 out v0.out order 2
timecorr vec1 v1 tstep 1.0 tcorr 100.0 out v1.out order 2
timecorr vec1 v0 vec2 v1 tstep 1.0 tcorr 100.0 out v0_v1.out order 2
```

Similarly, a vector perpendicular to the plane through atoms 18, 19, and 20 is obtained and further analyzed.

```
vector v2 @18,@19,@20 corrplane
timecorr vec1 v3 tstep 1.0 tcorr 100.0 out v2.out order 2
```

12.36 vectormath

```
vectormath vec1 <vecname1> vec2 <vecname2> [out <filename>] [norm] [name <setname>]
           [ dotproduct | dotangle | crossproduct ]
```

vec1 <**vecname1**> **vec2** <**vecname2**> Vector(s) on which to operate.

[out <**filename**>**]** Name of file to write output to.

[dotproduct] (Default) Calculate the dot-product of the two vectors.

[dotangle] Calculate angle from dot-product between the two vectors; vectors will be normalized.

[crossproduct] Calculate cross-product of the two vectors.

[norm] Normalize the vectors; this will affect any subsequent calculations with the vectors. This is turned on automatically if dotangle specified.

Calculate dot product, angle from dot product (degrees), or cross product for specified vectors. Note that **norm** normalizes the vectors themselves; the vectors will remain normalized for subsequent calculations or output. Either **vec1** or **vec2** can be of size 1; in that case each vector in the set with N frames operates on the single vector. For example, if **vec1** is size N and **vec2** is size 1, then each frame of vec1 is operated on the single vector from vec2.

For example, to get the angles between two previously calculated vectors v1 and v2:

```
vectormath vec1 v1 vec2 v2 dotangle out dotproduct.dat name acos(|V1|*|V2|)
```

12.37 wavelet

```
wavelet [crdset <set name>] nb <n scaling vals> [s0 <s0>] [ds <ds>]
        [correction <correction>] [chival <chival>] [type <wavelet>]
        [out <filename>] [name <setname>]
        [cluster [minpoints <#>] [epsilon <value>] [clusterout <file>]
           [clustermapout <file>] [cmapdetail] [kdist] [cprefix <PDB prefix>]
           [overlay <trajfile>] [overlayparm <parmfile>]]
```

[crdset <set name>] COORDS data set to use

nb <n scaling vals> Number of scales. The smaller the number the better resolution, but slower to plot.

[s0 <s0>] The smallest scale of the wavelet function (default 2dt where dt is time between snapshots in ps)

[ds <ds>] Spacing between discrete scales. (Default is 0.25. Smaller value of ds gives finer resolution. The largest values that give adequate sampling in scale for Morlet and Paul are 0.5 and 1.5, respectively)

[correction <correction>] The scale-to-wavelength parameter (1.01 for Morlet, 1.389 for Paul). Automatically set based on wavelet if not otherwise specified.

[chival <chival>] The value of χ^2_2at a particular confidence level

[type <wavelet>] Type of wavelet function to use <morlet> or <paul>

[out <filename>] Write results to file named <filename>

[name <setname>] Store results in data set with name <setname>

[cluster] Perform wavelet clustering i.e. wavelet feature extraction analysis.

- **[minpoints <#>]** Minimum number of points necessary to form a region of interest.
- **[epsilon <value>]** Minimum region of interest size.
- **[clusterout <file>]** Output for clustering (see below).
- **[clustermapout <file>]** Output cluster map (recommended gnuplot format, see below).
- **[cmapdetail]** Instead of the map being smoothed to cluster regions, show full detail.

[kdist] `Can be used to determine minpoints and epsilon - see below.`

[cprefix <PDB prefix>] `Output cluster region PDBs (only containing from minimum to maximum atom and minimum to maximum frame) with given prefix.`

[overlay <trajfile>] `Create a trajectory that can be overlaid with the original trajectory to highlight atoms of interest. Atoms in cluster regions will get their normal coordinates - all others are set to the common center of mass.`

[overlayparm <parmfile>] `Topology that can be used with the overlay trajectory.`

`<wavelet>: morlet, paul`

Perform the wavelet analysis using fast Fourier transform (FFT) algorithm on specified trajectory and write out to a gnuplot-formatted file named <name.gnu>. The created Wavelet map provides a clear picture of the significant motions which are characterized both in time and space. Note that typically the trajectory in question should have rotational and translational movement removed (via e.g. the ***rms*** command); otherwise these will be reflected in the wavelet analysis results.

Wavelet analysis contains two main steps which performs continues wavelet transform (CWT) and statistical significance testing as proposed by Torrence and Compo[34]. Analysis is executed on one dimensional (1-D) coordinate which is defined as the displacement from the starting position. For each atom, CWT is calculated over a specified range of scales from S_0up to $S_0 2^{(nb-1)ds}$. To obtain the CWT of the trajectory the Fourier transform of atom's displacement and wavelets which scaled by S (S is calculated from: $S = S_0 2^{jds}$; $j = 0, 1, 2, \ldots, nb-1$) is computed and then the inverse Fourier transform of the product of Fourier transforms will be calculated as the CWT. After calculating the wavelet coordinates for all atoms, a significance testing is performed to determine the significance of each wavelet coordinate. For doing this test we need to have an appropriate background spectrum to consider as a mean or expected spectrum and compare our wavelet coordinates against this background. In order to calculate the background spectrum since wavelet spectrum (according to the convolution theorem) follows the Fourier spectrum, the Fourier coefficients over every atom's displacement is calculated using the following formula and a model (μ_k) is constructed on average which Fourier coefficients fit (X_n) is the time series which is the atom's displacement and k is the frequency index[35].

$$f_{k=\frac{1}{N}} \sum_{n=0}^{N-1} \exp\left(\frac{-2\pi i k n}{N}\right) X_n$$

This test is implemented based on the null hypothesis that the assumption is that Fourier coordinates normally distributed around the expected value, then

the wavelet coordinates should also be normally distributed. Assuming the expected background spectrum and since the square of a normally distributed variable is chi-square distributed, then the distribution for the square of the absolute values of wavelet coordinates ($|W_{i,k}|^2$ is as follows ($\sigma^2$is the variance of the atom's displacement).

$$\sigma^2 \mu_k \chi_2^2/2$$

Then choosing a confidence level we can determine the minimum acceptable value for $|W_{i,k}|^2$to be considered as a significant coordinates at that certain confidence level. In the final map the scales of only those wavelet coordinates which are significantly above the expected distribution are stored.

For example, to perform wavelet analysis on residues 1 to 17 with 40 scaling values starting from scaling of 0.2 with a spacing of 0.25 using the Morlet wavelet:

```
parm nowat.withions.parm7
trajin nowat.image.nc
rms :1-17@C*,N*,O*,P* first mass
wavelet nb 40 s0 0.2 ds 0.25 correction 1.01 chival 1.6094 type morlet \
        :1-17 out wavelet.gnu usemap
```

Wavelet Analysis Feature Extraction

Wavelet analysis feature extraction (WAFEX)[36] uses a density-based clustering algorithm (a modified version of the DBSCAN algorithm) to highlight physical and temporal regions that have significant motions from wavelet mapsand can extract the specific atoms and frames involved in these motions for further analysis. Cluster regions shown in the map will be smoother by default for easier visualization (unless **cmapdetail** is specified). Details of the clustering are provided via the **clusterout** keyword with format:

```
#Cluster [points] [minatm] [maxatm] [minfrm] [maxfrm] [avgval]
```

#**Cluster** Cluster region number.

points Number of points in the cluster.

minatm Starting atom of the region.

maxatm End atom of the region.

minfrm Starting frame of the region.

maxfrm End frame of the region.

avgval Average value of points in the region.

For example, to create a 2D gnuplot map highlight regions of interest called 'cluster.gnu' one could use the following input.

```
parm ../DPDP.parm7
trajin ../DPDP.nc
rms @C,CA,N first
wavelet nb 10 s0 2 ds 0.25 type morlet correction 1.01 chival 0.25 \
        :1-22 name DPDP \
        cluster clustermapout cluster.gnu clusterout cluster.dat \
          minpoints 66 epsilon 10.0
datafile cluster.gnu usemap palette kbvyw
```

Some experimentation with **kdist** may be required to obtain reasonable values for **minpoints** and **epsilon**. See 12.4 on page 177 as well as the Heidari et al paper for further discussion.

13 Analysis Examples

Please note that typically for principal component analysis (PCA) the trajectory needs to be aligned against a reference structure to remove overall global and translation motion. Use the ***rms*** command for this.

13.1 Cartesian covariance matrix calculation and projection (PCA)

After calculating modes, snapshots can be projected onto these in an additional pass through the trajectory. It is very important that the snapshots used when projecting are exactly the same as those used to generate the original covariance matrix. This example takes advantage of the COORDS data set functionality in cpptraj to save snapshots for the purposes of projection.

```
# Step one. Generate average structure.
# RMS-Fit to first frame to remove global translation/rotation.
parm myparm.parm7
trajin mytraj.nc
rms first !@H=
average crdset AVG
run
# Step two. RMS-Fit to average structure. Calculate covariance matrix.
# Save the fit coordinates.
rms ref AVG !@H=
matrix covar name MyMatrix !@H=
createcrd CRD1
run
# Step three. Diagonalize matrix.
runanalysis diagmatrix MyMatrix vecs 2 name MyEvecs
# Step four. Project saved fit coordinates along eigenvectors 1 and 2
crdaction CRD1 projection evecs MyEvecs !@H= out project.dat beg 1 end 2
```

13.2 Dihedral covariance matrix calculation and projection for backbone phi/psi (PCA)

```
parm ../1rrb_vac.prmtop
trajin ../1rrb_vac.mdcrd
# Generation of phi/psi dihedral data
multidihedral BB phi psi resrange 2
run
# Calculate dihedral covariance matrix and obtain eigenvectors
matrix dihcovar dihedrals BB[*] out dihcovar.dat name DIH
diagmatrix DIH vecs 4 out modes.dihcovar.dat name DIHMODES
run
# Project along eigenvectors
projection evecs DIHMODES out dih.project.dat beg 1 end 4 dihedrals BB[*]
run
```

References

[1] Daniel R. Roe and T.E. Cheatham, III. PTRAJ and CPPTRAJ: Software for Processing and Analysis of Molecular Dynamics Trajectory Data. *J. Chem. Theory Comput.*, 9:3084–3095, 2013.

[2] Daniel R. Roe and Thomas E. Cheatham III. Parallelization of CPPTRAJ enables large scale analysis of molecular dynamics trajectory data. *Journal of Computational Chemistry*, 39(25):2110–2117, 2018.

[3] S. Chatterjee, P. G. Debenedetti, F. H. Stillinger, and R. M. Lynden-Bell. A Computational Investigation of Thermodynamics, Structure, Dynamics and Solvation Behavior in Modified Water Models. *J. Chem. Phys.*, 128:124511, 2008.

[4] T. Lazaridis. Inhomogeneous Fluid Approach to Solvation Thermodynamics. 1 Theory. *J. Phys. Chem. B*, 102:3531–3541, 1998.

[5] C. N. Nguyen, T. Kurtzman Young, and M. K. Gilson. Grid Inhomogeneous Solvation Theory: Hydration Structure and Thermodynamics of the Miniature Receptor cucurbit[7]uril. *J. Chem. Phys.*, 137:044101, 2012.

[6] W. Humphrey, A. Dalke, and K. Schulten. VMD Visual Molecular Dynamics. *J. Molec. Graph.*, 14:33–38, 1996.

[7] D. J. Sindhikara, N. Yoshida, and F. Hirata. Placevent: An Algorithm for Prediction of Explicit Solvent Atom Distribution-Application to HIV-1 Protease and F-ATP Synthase. *J. Comput. Chem.*, 33:1536–1543, 2012.

[8] J.J. Chou, D.A. Case, and A. Bax. Insights into the mobility of methyl-bearing side chains in proteins. *J. Am. Chem. Soc.*, 125:8959–8966, 2003.

[9] C. Perez, F. Lohr, H. Ruterjans, and J.M. Schmidt. Self-Consistent Karplus Parameterization of (3)J couplings depending on the polypeptide side-chain torsion chi(1). *J. Am. Chem. Soc.*, 123:7081–7093, 2001.

[10] P.H. Hunenberger, A.E. Mark, and W.F. van Gunsteren. Fluctuation and Cross-correlation Analysis of Protein Motions Observed in Nanosecond Molecular Dynamics Simulations. *J. Mol. Biol.*, 252:492–503, 1995.

[11] J.J. Prompers and R. Brüschweiler. Dynamic and structural analysis of isotropically distributed molecular ensembles. *Proteins*, 46:177–189, 2002.

[12] J.J. Prompers and R. Brüschweiler. General framework for studying the dynamics of folded and nonfolded proteins by NMR relaxation spectroscopy and MD simulation. *J. Am. Chem. Soc.*, 124:4522–4534, 2002.

[13] M.L. Connolly. Analytical molecular surface calculation. *J. Appl. Cryst.*, 16:548–558, 1983.

[14] M. A. El Hassan and C. R. Calladine. Two distinct modes of protein-induced bending in dna. *J. Mol. Biol.*, 282:331–343, 1998.

[15] XJ Lu and WK Olson. 3dna: a software package for the analysis, rebuilding and visualization of three-dimensional nucleic acid structures. *NUCLEIC ACIDS RESEARCH*, 31:5108–5121, 2003.

[16] M.S. Babcock, E.P.D. Pednault, and W.K. Olson. Nucleic Acid Structure Analysis. *J. Mol. Biol.*, 237:125–156, 1994.

[17] Wilma K. Olson, Manju Bansal, Stephen K. Burley, Richard E. Dickerson, Mark Gerstein, Stephen C. Harvey, Udo Heinemann, Xiang-Jun Lu, Stephen Neidle, Zippora Shakked, Heinz Sklenar, Masashi Suzuki, Chang-Shung Tung, Eric Westhof, Cynthia Wolberger, and Helen M. Berman. A standard reference frame for the description of nucleic acid base-pair geometry. *J. Mol. Biol.*, 313:229–237, 2001.

[18] C Altona and M Sundaralingam. Conformational analysis of the sugar ring in nucleosides and nucleotides. a new description using the concept of pseudorotation. *J Am Chem Soc*, 94:8205–8212, 1972.

[19] SC Harvey and M Prabhakaran. Ribose puckering - structure, dynamics, energetics, and the pseudorotation cycle. *J Am Chem Soc*, 108:6128–6136, 1986.

[20] D Cremer and JA Pople. A general definition of ring puckering coordinates. *J Am Chem Soc*, 97:1354–1358, 1975.

[21] W. Kabsch and C. Sander. Dictionary of protein secondary structure: pattern recognition of hydrogen-bonded and geometrical features. *Biopolymers*, 22:2577–2637, 1983.

[22] G. Cui, J. M. Swails, and E. S. Manas. SPAM: A Simple Approach for Profiling Bound Water Molecules. *J. Chem. Theory Comput.*, 9:5539–5549, 2013.

[23] J. Weiser, P.S. Shenkin, and W.C. Still. Approximate Atomic Surfaces from Linear Combinations of Pairwise Overlaps (LCPO). *J. Comput. Chem.*, 20:217–230, 1999.

[24] M. Ester, H. Kriegel, J. Sander, and X. Xu. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. *Proc. Second Int. Conf. Knowledge Disc. Data Mining (KDD-96)*, pages 226–231, 1996.

[25] Alex Rodriguez and Alessandro Laio. Clustering by fast search and find of density peaks. *Science*, 344:1492–1496, 2014.

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

[27] Donald A. McQuarrie. *Statistical Thermodynamics*. Harper and Row, New York, 1973.

[28] Chia-En Chang, Wei Chen, and Michael K. Gilson. Evaluating the Accuracy of the Quasiharmonic Approximation. *Journal of Chemical Theory and Computation*, 1(5):1017–1028, 2005. PMID: 26641917.

[29] Daniel R. Roe, Christina Bergonzo, and Thomas E. Cheatham III. Evaluation of enhanced sampling provided by accelerated molecular dynamics with hamiltonian replica exchange methods. *J. Phys. Chem. B*, 118(13):3543–3552, 2014.

[30] Rodrigo Galindo-Murillo, Daniel R. Roe, and T.E. Cheatham, III. Convergence and reproducibility in molecular dynamics simulations of the DNA duplex d(GCACGAACGAACGAACGC). *Biochim. Biophys. Acta*, 1850:1041–1058, 2015.

[31] V. Wong and D.A. Case. Evaluating rotational diffusion from protein md simulations. *J. Phys. Chem. B*, 112:6013–6024, 2008.

[32] B. Schneider, S. Neidle, and H. M. Berman. Conformations of the sugar-phosphate backbone in helical dna crystal structures. *Biopolymers*, 42(1):113–124, 1997.

[33] B. Schneider, Z. Moravek, and H. M. Berman. Rna conformational classes. *Nucleic Acids Res.*, 32(5):1666–1677, 2004.

[34] C. Torrence and G. P. Compo. A practical guide to wavelet analysis. *Bull. Am. Meteorol. Soc.*, 79(1):61–78, 1998.

[35] N. C. Benson and V. Dagget. Wavelet analysis of protein motion. *Int. J. Wavelets Multi.*, 10(4), 2012.

[36] Z. Heidari, D. R. Roe, R. Galindo-Murillo, J. B. Ghasemi, and T. E. Cheatham, III. Using Wavelet Analysis To Assist in Identification of Significant Events in Molecular Dynamics Simulations. *J. Chem. Inf. Model.*, 56:1282–1291, 2016.