CPPTRAJ: Trajectory Processing and Analysis

Daniel R. Roe

November 2010

Introduction

A crucial part of research with computational methods involves performing various types of analysis on the significant amount of data that is generated in the form of coordinate trajectories. In the Amber molecular dynamics package, this analysis is current performed with Ptraj. Ptraj is able to perform many types of analyses, and can process multiple trajectories. However, one of the limitations of Ptraj is that all coordinates in a given Ptraj run must correspond to a single topology file. This prevents certain types of analysis, for example calculating the RMSD of a coordinate frame to a reference frame with a different topology.

Cpptraj is a complimentary program to ptraj that can process trajectory files with different topology files in the same run. Although certain parts of the Ptraj code are used in Cpptraj, it is overall a completely new code base written primarily in C++ with an eye towards making future code development and additions as easy as possible. In addition to reading multiple topology files, Cpptraj can read multiple reference structures, write multiple output files (for which specific frames to be written can be specified), stripped topology files (currently useable for visualization only), output multiple data sets to the same data file (e.g. two dihedral calculations like phi and psi can be written to one file), and has native support for compressed files along with many other improvements. The code is at least as fast as ptraj, and in many cases is much faster, particularly when processing NetCDF trajectories.

From a developers standpoint the code is free of memory leaks (checked with valgrind at every stage of development), free of warnings (gnu compilers only, checked with the -Wall compiler flag), and avoids global variables and tangled headers.

Comparison to Ptraj - Important Differences

The overall flow of Cpptraj is similar to Ptraj. First the run is set up via commands read in from an input file; a limited interactive interface (STDIN) similar to ptraj is also supported. Trajectories are then read in one frame at a time. 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.

Some of the most notable differences from Ptraj are as follows:

1. File compression is handled internally rather than external calls to gzip, bzip2, etc, which makes reading and writing compressed files more efficient.

2. Any file read or written by Cpptraj can be compressed (with the exception of Netcdf trajectories). So topology files could be read in as .gz files, and data files can be written as .bz2 files. Compression is detected automatically when reading, and is determined by the filename extension on writing.

3. If two actions specify the same file with the 'out' keyword, data from both actions will be written to that file.

4. Data files specified by the 'out' keyword can be written in xmgrace format if the filename given has a '.agr' extension.

5. Multiple output trajectories can be specified. In addition, output files can be directed to write only specific frames from the input trajectories.

6. Multiple reference structures can be specified. Specific frames from trajectories may be used as a reference structure.

7. The rmsd action allows specification of a separate mask for the reference structure. In addition, per-residue RMSD can be calculated easily.

8. When stripping coordinates with the strip action, a stripped topology file can be written out. Currently this topology is good for visualization only, not simulation (although this support is planned for future releases).

Some notes on multiple topologies

Since Cpptraj supports multiple topology files, 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 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.

Some notes on datasets and datafiles

Datafiles can currently be given in two formats: data file and grace file. Data file simply has data in columns, like ptraj. Grace files can be read in by xmgrace. The format is specified by the file suffix, so that 'filename.agr' will output in grace format, and anything else is a normal data file. The xmgrace output is particularly nice for the secstruct sumout file.

With all action commands that print out a dataset (e.g. distance, angle, dihedral, rmsd, etc) an additional argument can be given optionally to specify the name of the dataset. For example, the command:

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

will write the distance between residues 1 and 2 to d1.dat. The header of d1.dat will be something like

```
#Frame     dataXXXX
```

where XXXX is the internal number of the dataset. In contrast, the command:

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

will write the same distance to d1.dat, but the header of d1.dat will be:

```
#Frame     d1
```

Most actions allow datasets 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 produce in phipsi.dat:

```
#Frame   phi   psi
```

Building Cpptraj

Cpptraj comes with a very simple configure script. Currently only intel (icc, icpc) and gnu (gcc, g++) compilers are supported and tested. The gnu compilers are recommended as they seem to produce the fastest and most compact code. The configure script will perform a basic check to make sure that the compilers and libraries specified actually exist.

```
Usage: ./configure [gnu/intel] OPTIONS
OPTIONS:
--help     : Display this message.
-d, -debug : Turn on compiler debugging info (-g flag)
```

```
-debugon   : Add -DDEBUG flag to activate internal debugging
-mpi       : Use mpicc/mpicxx to compile [NOTE: CURRENTLY NOT FULLY SUPPORTED]
-opt       : Do not use optimized compiler flags (default if -d specified)
-nobzlib   : Do not use Bzip2
-nozlib    : Do not use zlib (gzip/zip)
-nonetcdf  : Do not use netcdf
-nolfs     : Do not enable large file support.
Static linking options:
--with-netcdf=<DIR>
--with-zlib=<DIR>
--with-bzlib=<DIR>
```

To make cpptraj type:

```
make clean
make install
```

This will create the 'cpptraj' binary in the ./bin directory. Alternatively, 'make' by itself will make 'cpptraj' but leave it in the ./src directory.

Testing Cpptraj

The ./test directory contains numerous tests that check almost all of the functionality of Cpptraj. By default the tests will run on the binary in the ./bin directory. To run the tests:

```
cd test
make test
```

A summary of the test results will be written to 'Test_Results.dat' in the ./test directory and every individual test directory. Stdout from each test is recorded to 'test.out' in every individual test directory.

Running Cpptraj

Command Line Syntax

The commands are very similar in format to ptraj commands. To run:

```
./cpptraj -i <input file> [-p <parm file1> ...]
```

or for compatibility with Ptraj:

```
./cpptraj <parm file> <input file>
```

For the first syntax case zero or more topology files may be specified with '-p' on the command line; however if no topology file is specified on the command line one must be specified in the input file. If run with no arguments:

```
./cpptraj
```

this brings up a (somewhat) interactive command-line style interface (STDIN), similar to that in Ptraj.

Input Command Syntax

The following is a list of commands that are currently recognized by Cpptraj from the input file or STDIN. Lines beginning with '#' are ignored as comments. Lines can also be continued through use of the '\' character.

The mask syntax is the same as for Ptraj. It is important to note that for several commands (notably trajout and the rmsd action) some arguments are ranges - THESE ARE NOT 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.

General Commands

noprogress

Do not display progress bars during trajectory processing.

debug <#>

Set the level of debug information to print. In general the higher the <#> the more information that is printed.

File Commands

parm <filename>

Read in parameter file specified by <filename>. Currently can read Amber topology and PDB files.

parminfo [<#>] [<mask>]

Print information about atoms in <mask> for parameter file specified by <#> (numbers start from 0).

trajin <filename> [start] [stop] [offset] [parm <parmfile> | parmindex <#>] [remdtraj remdtrajtemp <Temperature>]

Read in trajectory specified by filename with specified start, stop, and offset (1, # frames, and 1 if not specified). Associate with parmfile specified by filename or parm number specified by parmindex. If no parm/parmindex argument is specified the first parm read in is used. Example:

```
# parmindex 0
parm top0
# parmindex 1
```

```
parm top1
# parmindex 2
parm top2
trajin Test1.crd parm top1
trajin Check1.crd parmindex 1
trajin Test0.crd
```

Test1.crd and Check1.crd are associated with top1, which is the second parm read in. Test0.crd is associated with top0; since no parm or parmindex keyword was specified Cpptraj defaulted to the first parm read in.

If the **remdtraj** keyword is specified the trajectory is treated as belonging to the lowest # replica of a group of REMD trajectories following a naming convention of <REMDFILENAME>.X, where X is the replica number. All files matching this convention will be searched for, and during processing only frames with a temperature matching the one specified by **remdtrajtemp** will be processed.

trajout <filename> [<fileformat>] [append] [nobox] [parm <parmfile> | parmindex <#>] [<range>]

Write trajectory specified by filename in specified file format (Amber trajectory if none specified). Other currently recognized formats are "pdb", "netcdf", and "restart". The file will be appended to if **append** is specified. Box coordinates will not be written if **nobox** specified (only matters when input topology has box coordinates). Associate with parmfile or parmindex (first parm read in if not specified).

Multiple output trajectories of any format can be specified. Currently only frames that match the parameter file will be written. So given the input:

```
parm top0
parm 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) will be written to output.crd. The trajectory input0.crd is associated with top0; since no parameter file was specified cpptraj defaults to the first parm file read in.

If <range> is given, only input frames matching the range will be written out. For example, given the input:

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

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

reference <filename> [frame#] [parm <parmfile> | parmindex <#>]

Read specified trajectory frame (1 if not specified) as reference coordinates. Associate with parmfile or parmindex (first parm read in if not specified).

Action Commands

Most actions in Cpptraj function exactly the way they do in ptraj and are backwards-compatible. Some commands have extra functionality (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).

angle <mask1> <mask2> <mask3> [out <filename>]

Calculate angle between atoms in mask1, mask2, and mask3.

center [<mask>] [origin] [mass]

Center all atoms to the center of the box. If origin is specified, center to the origin instead. If mass is specified, move using the center of mass of atoms instead of center of geometry. If mask is specified center all atoms using only the atoms in mask.

dihedral <mask1> <mask2> <mask3> <mask4> [out <filename>]

Calculate dihedral angle between atoms in mask1-4.

distance <mask1> <mask2> [out <filename>] [geom] [noimage]

Calculate distance between the center of mass of atoms in mask1 to atoms in mask2. If geom is specified use the geometric center instead. For periodic systems imaging is turned on by default; the noimage keyword disables imaging.

hbond [out <filename>] <mask> [angle <cut>] [dist <cut>] [avgout <avg-filename>]

Search for hydrogen bond donor and acceptor atoms in the region specified by <mask> (currently following the simplistic criterion that "hydrogen bonds are FON", i.e. hydrogens bonded to F, O, and N atoms are considered) and calculate the number of hydrogen bonds formed for each frame, writing the results to the file specified by "out". Hydrogen bonds are considered to have the form:

```
Acceptor ... Hydrogen-Donor
```

are determined via the distance between the heavy atoms using a cutoff of 3.0 Å (or the value specified by dist) and the angle between the acceptor, hydrogen, and donor atoms using a cutoff of 135° (or the value specified by angle).

If avgout is specified the average of each hydrogen bond (sorted by population) formed over the course of the trajectory is printed to <avgfilename>. The output file has 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, and AvgAng is the average angle of the bond.

image [origin] [center] [triclinic | familiar [com <commask>]] [<mask>]

For periodic systems only, image any atoms in <mask> (or all atoms if no mask specified) outside of the box back into the box. Currently both orthorhombic (ifbox=1, rectangular/cubic) and non-orthorhombic (ifbox=2, truncated octahedron) boxes are supported. Right now only imaging by molecule is supported.

If origin is specified center at the origin, otherwise use the box center. If center is specified the center of mass of atoms will be used to image, otherwise the position of the first atom will be used. If triclinic is specified force imaging with triclinic code (this is the default for non-orthorhombic boxes). If familiar is specified image with the triclinic code and put the box into the more familiar truncated octahedral shape. If com is specified with familiar, center based on the center of mass of atoms in <commask> (otherwise center to origin or box center).

rmsd <mask> [first | reference | ref <reffilename> | refindex <#>] [<refmask>] [out <filename>] [nofit] [mass]
[perres perresout <perresfile> [range <resRange>] [refrange <refRange>] [perresmask <pmask>] [perrescenter] [perresinvert]]

Calculate the RMSD between atoms in Frame defined by <mask> to atoms in Reference defined by <refmask>. Both <mask> and <refmask> must specify the same number of atoms. If no <refmask> is specified, <mask> is used for Reference. The Reference structure is defined by one of the following keywords:

- first: Use the first trajectory frame processed as reference.
- reference: Use the first previously read in reference structure (refindex 0).
- ref: Use previously read in reference structure specified by <reffilename>.
- refindex: Use previously read in reference structure specified by <#> (based on order read in).

If nofit is specified the Frame coordinates will not be best-fit to reference coords prior to RMSD calculation. If mass keyword is specified the center of mass of atoms in <mask> will be used, otherwise geometric center will be used. Example:

```
reference StructX.crd
reference Struct-begin.rst7
rmsd :1-20@C,CA,N reference :3-23@C,CA,N out rmsd1.dat
```

This will calculate the 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 (the first reference structure read in) and write the results to rmsd1.dat. If instead of the reference keyword 'refindex 1' was used, Struct-begin.rst7 would be the reference structure instead. Note that if the number of atoms in <mask> does not equal the number of atoms in <refmask> an error will be generated.

If the **perres** keyword is specified, after the initial RMSD calculation the no-fit RMSD of each residue in each Frame specified by <resRange> (or all solute residues if refrange not specified) will be calculated to each residue in Reference specified by <refRange> (or each residue in <resRange> if refrange not specified), the results of which will be written to the file speicfied by perresout. By default all atoms of each residue are used - an additional mask can be specified by perresmask - this mask is appended to the default mask of each residue (:X, where X is residue number). If perrescenter is specified residues will be centered to a common point of reference before no-fit RMSD is calculated (this will emphasize changes in the local structure of the residue). So for example:

```
rmsd :10-260 refindex 0 perres perresout PRMS.dat range 190-211 perresmask &!(@H=)
```

will first perform an rms-fit calculation on residues 10-260, then
calculate the per-residue no-fit rmsd of residues 190 to 211
(excluding any hydrogen atoms).
If perresinvert is specified, data for each residue in <perresfile>
will be written in rows instead of columns, i.e.:

```
RES1   RmsdFrame1 RmsdFrame2 RmsdFrame3 ...
RES2   RmsdFrame1 RmsdFrame2 RmsdFrame3 ...
```

secstruct [out <filename>] [<mask>] [sumout <sumfilename>]

Calculate secondary structural propensities for residues in mask (or all solute residues if no mask given) using the DSSP method of Kabsch and Sander [REF]. Results will be written to filename specified by out in format:

```
#Frame    STRING
```

where #Frame is the frame number and STRING is a string of characters (one for each residue) where each character represents a different structural type:

Character	**SS type**
0	**None**
b	**Parallel Beta-sheet**
B	**Anti-parallel Beta-sheet**
G	**3-10 helix**
H	**Alpha helix**
I	**Pi (3-14) helix**
T	**Turn**

Average structural propensities over all residues for each frame will be written to the file specified by sumout (or ".sum" will be appended to <filename> if sumout is not specified).

strip <mask1> [outprefix <name>]

Strip all atoms specified by mask from the frame and modify the topology to match. If outprefix is specified, for every topology modified in this way a file <name>.<parmFilename> will be written that matches the stripped system. Currently these stripped topologies are for visualization purposes only.

Caveat Emptor

Currently this code is still in the Alpha stage and so is not guaranteed to work flawlessly.

One important thing to keep in mind is that although for the purposes of specifying frames in the trajin, trajout, and reference command frames start from 1 (for backwards-compatibility with Ptraj), internally frames start from zero and when datafiles are output they still start from 0.