MDASH Molecular Dynamics Analysis

December 2017

Matt Ellis, Brian Hudson, David Salt and David Whitley

This manual is for MDASH, version 3.1.0.

Table of Contents

1 Overview

DASH is a technique for analysing molecular dynamics (MD) simulations based on the torsion angles of rotatable bonds. It extracts the major features for each torsion angle and combines these into *states* that describe the conformation of the whole molecule. By ignoring high-frequency oscillations within the DASH states, a compressed representation of an MD trajectory is obtained as the sequence of DASH states visited during the simulation, together with the length of time spent in each state. The latest version of the DASH algorithm is implemented here as a C++ program `mdash`.

The DASH algorithm takes care to treat the circular nature of the torsion angle data correctly and uses circular statistics to describe the DASH states. The main steps in the DASH algorithm are as follows.

- To remove some of the high-frequency noise, the initial torsion angles are replaced by moving average values. These are calculated over a window of user-specified length, typically 11 time steps.
- A discrete set of states for the individual torsion angles is identified by examining their histograms. Local maxima in the distribution of each torsion angle are located and states are defined by placing cut-points midway between adjacent maxima. The trajectory of each torsion angle is then represented by the sequence of states visited at each time step.
- A smoothing process is applied to eliminate short-lived states. The state sequence for each torsion angle is considered as a series of *bouts* in which the trajectory remains in a single state. Bouts of short duration (less than the *bout_length* parameter) are removed by dividing them in half and assigning the two halves to the preceding and following bouts.
- The torsion angle state sequences are then combined into state sequences for the whole molecule. For example, if torsion T1 has state sequence 1, 1, 2, 1, 1, 1, 2, . . . and torsion T2 has state sequence 1, 1, 3, 2, 1, 1, 1, . . . then the combined sequence is (1,1), (1,1), (2,3), (1,2), (1,1), (1,1), (2,1),. . . Trajectories typically remain in the same state for long periods, leading to long constant runs, so the sequences are compressed by writing them in the form of bouts $(s_1, b_1), (s_2, b_2), ...$, where the *ith* bout consists of b_i time steps in state s_i.
- The combined sequences are smoothed to remove short-lived states in the same way as the individual torsion sequences. This determines a final set of DASH states, but tends to remove too much dynamic detail as it removes all bouts of length less than the smoothing level σ. Therefore a final *roughening* process is applied, using a bout length $\rho < \sigma$, in which bouts of the DASH states with length between ρ and σ are reinstated.

For further details see:

- *D. W. Salt, B. D. Hudson, L. Banting, M. J. Ellis and M. G. Ford, DASH: A novel analysis method for molecular dynamics simulation data. Analysis of ligands of PPAR-gamma, J. Med. Chem., 48, 3214-3220, 2005.*

2 Running the program

The program was originally called `dash`, but since its first release the Debian Almquist shell, also named `dash`, has become the default shell on Debian based distributions such as Ubuntu. To avoid a name clash, releases will henceforth be named *mdash-x.y.z* and the main program is now `mdash`. The main algorithm, however, will continue to be known as the DASH algorithm.

2.1 Command line interface

The syntax for the `mdash` command is:

`mdash` Launch the graphical interface.

`mdash [options] -i` *`input_file`* `-o` *`output_file`*
Run `mdash` from the command line with the specified options.

`mdash [options] -P` *`parameter_file`* `-T` *`trajectory_file`* `-o` *`output_file`*
Run `mdash` from the command line on an Amber trajectory specified by *seed*.

`mdash -S|--similarity` *`file1 file2`*
Calculate similarities between states in two `mdash` output files.

Options:

`-h [--help]`
Display a help message.

`-v [--version]`
Display the program version number.

`-q [--quiet]`
Run quietly with no progress messages.

`-D [ --debug ]`
Write internal details to stderr.

`-i [--input]` *`file`*
Specify input file.

`-o [--output]` *`file`*
Specify output file.

`-w [--window]` *`winsize`*
Set the moving average window size (an odd integer) [default 11].

`-b` *`binsize`*
Set the bin size for single variable distributions [default 4].

`-u` *`runlen`* Set the run length defining single-variable maxima [default 3]. A local maximum is preceded by at least *runlen* increasing bin values and followed by at least *runlen* decreasing values.

`-f` *`fmax`* Set the minimum value for a single variable maximum, specified as a percentage of the number of frames in the trajectory [default 2.4].

`-m` *`smin`* Set the minimum distance allowed between single-variable maxima [default 48]. Local maxima closer than this are merged into a single pseudo-maximum midway between them.

`-l` *`boutlen`*
Set the minimum bout length defining a single variable state [default 20].

`-s` *smooth* Set the smoothing level [default 40].

`-r` *rough* Set the roughening level [default 20].

`-t [ --timestep ]` *step*
: Specify the inter-frame timestep in picoseconds [default 1].

`-d [ --distances ]`
: Process distance, rather than angular, data. This is an experimental feature, not intended for general use at present.

`-p [ --pca ]`
: Calculate principal components.

`-a [ --pca-autoscale ] [=`*[0|1]*`]`
: Use autoscaled variables for pca [default 1].

`-x [ --repex ]`
: Process a replica exchange trajectory.

`-y [ --repex-fraction ] arg`
: The fraction of frames defining replica exchange states [default 0.01].

`-H [ --write-histogram ]`
: Write histograms of the single variable distributions.

`-C [ --write-combined ]`
: Write the combined states and trajectory.

`-S [ --write-smoothed ]`
: Write the smoothed states and trajectory.

`-F [ --write-sequence ]`
: Write the full state sequence. This produces copious output and is intended only for debugging.

`-L [ --write-labelled-data ]` *filename*
: Write input data with state labels appended in final column to *filename*.

`-S [ --similarity ] arg (=`*file1 file2*`) write similarities between`
: states in two `mdash` output files to stdout

Amber Interface Options:

`-P [ --amber-parm ]` *arg*
: AMBER parameter (topology) file

`-T [ --amber-traj ]` *arg*
: AMBER trajectory (netcdf) file(s). Accepts a single file or a comma-separated list of files.

`-R [ --residues ]` *arg*
: Range of residues to analyse. A comma-separated list of individual residue numbers or ranges N-M. If omitted, all residues are analysed.

`-D [ --dihedrals ]` *arg (=phi,psi)*
: Dihedral types to analyse. Accepts all types recognized by cpptraj.

`-N [ --snap ]`
: Write PDB files for frames representing DASH states

`-k [ --keep-mdash-input ]`
: Retain the mdash input file

`-j [ --keep-cpptraj ]`

Retain intermediate cpptraj files

2.2 Graphical interface

The `mdash` command without any options launches the graphical interface. The workflow consists of loading a DASH input file or an AMBER parameter and trajectory file using the `File` menu, setting the options for the DASH algorithm by selecting `Dash|Options` and running the algorithm with `Dash|Run`. The main window displays progress messages and information about trajectories loaded into the program. The results of each run are displayed in a separate MDASH Viewer window. Multiple viewer windows may be open simultaneously to compare different trajectories or the effect of different options.

The Viewer window contains the following panels.

- Summary

  A brief listing of the input trajectory, the program options and the number of states found.
- Trajectory

  A plot of the DASH states visited during the trajectory against time.
- Frequencies

  A histogram of the DASH state frequencies. Each bout is shown in a separate colour.
- Bouts

  A grid showing the number of frames spent in each bout for the DASH states. This is the numerical data used to plot the Frequencies panel.
- Dash States

  A grid with a row for each state, showing:
  - the mean torsion angles for the state
  - the number and percentage of frames spent in the state
  - the representative frame for the state
  - the RMSD between the mean torsion angles and the angles of the representative frame
- Similarity

  A grid showing the similarities between the Dash states. Select the appropriate tab to display the either the circular or cosine similarities, which are calculated as follows.

  Each dash state is represented by the vector of mean torsion angles $x = (x_1, \ldots, x_n)$. The distance between two states x and y is

  $D(x, y) = \sqrt{d(x_1, y_1)^2 + \ldots + d(x_n, y_n)^2}$

  where the distance between two angles (in degrees) is

  $d(x_i, y_i) = min(|x_i - y_i|, 360 - |x_i - y_i|)$.

  As this angular distance lies in $[0, 180]$, the distance between two states lies in $[0, 180\sqrt{n}]$. So we define the circular similarity between two states as

  $S(x, y) = 1 - D(x, y)/180\sqrt{n}$

  which lies in $[0, 1]$.

  The cosine similarity between states x and y is defined as $S(x, y) = (x.y)/\sqrt{|x||y|}$, which lies in $[-1, 1]$.
- Torsion Histograms

  Plots of the histograms of each torsion angle. The `Next` and `Previous` buttons navigate through the torsion angles. The number of plots on each page is controlled by selecting `Options|Torsion Histogram Grid` from the menu.

- Torsion States

  A list of the states found for each torsion angle. These should be compared with the Torsion Histograms to check that a reasonable set of torsion states has been found.
- Scree Plot

  The principal component eigenvalues. The variance explained by each principal component is shown in red, the cumulative variance explained in green.
- PCA

  A 3D plot of the principal components, with states drawn in separate colours. The left mouse button rotates the plot and the `R` key resets the view. To change the components displayed select `Options|PCA Plot`.

The `File|Save` menu in the Viewer window allows the results displayed in the panels to be saved, either as text files or PNG images as appropriate. Some of the images can be saved only when they are visible, otherwise their selection is disabled in the menu.

The manual can be viewed by selecting `Help|Manual` in the main window.

2.3 Amberdash

The Perl script `amberdash` runs `mdash` on trajectories generated by Amber (`http://ambermd.org`). Documentation in POD format is included in the file. To read it use the command

```
perldoc amberdash
```

The script calls `mdash` and `cpptraj`. If they are not on the PATH, specify their full pathnames at the top of the script.

Much of the functionality of `amberdash` is now integrated into the main `mdash` program.

2.4 Replica exchange trajectories

For pseudo-trajectories formed by concatenating sections of replica exchange molecular dynamics, initial tests showed the steps in the DASH algorithm that assume the input is from a single time series (i.e. the smoothing) are inappropriate. The combined states can be calculated as usual, but replica exchange trajectories typically produce large numbers of states, many of which are sparsely populated. Therefore, when the `-x` option is used, a set of replica exchange states is defined as the subset of the combined states that contain at least a minimum fraction of frames, specified by the `-y` option. In this case the replica exchange state trajectory in the output file is essentially the combined state trajectory, with sparsely populated states marked as state 0.

3 File Formats

Some sample input and output files are provided in the `examples` directory.

3.1 Input Files

The input files for `mdash` are whitespace-separated ASCII text files containing the values of *ntor* torsion angles, in degrees, at *nstep* time steps, with the values for each torsion angle in a single column and the values for each time step in a single row.

Input files may be compressed with `gzip` or `bzip2`. In this case the file type is inferred from the filename extension, which must be `.gz` or `.bz2`, respectively.

3.2 Output Files

Following two header lines containing the version number and a timestamp, the DASH output file is divided into blocks separated by tags in square brackets. The first two sections contain details of the input trajectory and the DASH options.

```
DASH, version 2.12
Wed Nov  4 08:57:25 2015

[TRAJECTORY]
file      : /home/dw/projects/dash/svn/trunk/examples/tzd.in
variables : 8
frames    : 25000

[OPTIONS]
data     : angles
timestep : 1 ps
window   : 11
binsize  : 4
runlen   : 3
fmax     : 2.4
smin     : 48
boutlen  : 20
smooth   : 40
rough    : 20
```

The next section describes the states for the individual variables. For each variable, DASH prints the list of local maxima in the torsion angle distribution; the single-variable states, numbered sequentially from 1; the number of state transitions during the trajectory; and the number of frames where the single-variable states are reassigned by smoothing the trajectories, with the corresponding proportion of the total number of frames in parentheses. For example, the following extract shows a single state for torsion ANGLE_1 and three states for ANGLE_2, with one state wrapping around +/-180.

```
[ANGLE_1]
maxima      : -62
states      : 1 = [-180, 180)
transitions : 1
reassigned  : 0 (0.00%)

[ANGLE_2]
maxima      : -178, -58, 66
states      : 1 = [-180, -118), 2 = [-118, 4), 3 = [4, 124), 1 = [124, 180)
transitions : 27
reassigned  : 1 (0.01%)
```

If the -H flag is used, a histogram of the distribution of angles is printed within each torsion angle block. The following example shows a case where the torsion angle has frequencies 2330 in the [-180,-176) bin, 1466 in the [-176,-172) bin, etc.

```
[ANGLE_1_HISTOGRAM]
Bin     Count
-180    2330
-176    1466
-172    408
...
 168    54
 172    287
 176    1284
```

The next section summarizes the result of the DASH algorithm, giving the number of combined states, the number of final states after the smoothing and roughening procedures, and the number of frames where the combined state is reassigned during this process, with the corresponding percentage of the total number of frames in parentheses.

```
[SUMMARY]
combined states : 82
final states    : 55
transitions     : 269
reassignments   : 1602 (6.41%)
```

The DASH states and their frequencies in the trajectory are listed next. Each state is defined as a sequence of individual torsion angle states. In the following example, the first DASH state corresponds to state 2 in T6 and to state 1 in each of the other torsion angles.

```
[DASH_STATES]
1     1 1 1 1 1 2 1 1
2     1 1 1 1 1 3 1 1
3     1 1 1 1 1 3 2 1
...
53    1 3 2 1 2 1 2 1
54    1 3 2 1 2 3 2 1
55    1 3 2 1 3 3 2 1
```

If the `-R` flag is used, the trajectory frames that best represent each DASH state are identified. These are the rows of the *input_file*, numbered from 1, with the smallest RMSD from the mean angles for the DASH state.

This is followed by two blocks of summary statistics: the circular means and standard deviations of the torsion angles in each Dash state. We emphasize that these are both *circular* statistics.

```
[DASH_STATE_DISTRIBUTION]
State     Frames   %Frames   Rep.Frame      RMSD
1           265      1.06        9700      5.25
2           243      0.97        9914      7.15
3            46      0.18        9891      6.48
...
53          396      1.58       11422      4.49
54          360      1.44        1514      4.76
55          116      0.46       13050      7.56

[DASH_STATE_MEANS]
1        -60.41 -179.58  -81.36  116.31  -64.51   76.34 -118.17   -9.24
2        -60.15 -176.83  -76.57  113.39  -64.85  178.57  -81.61    2.67
3        -62.07 -177.93  -75.38  110.89  -61.97 -176.81   87.02  -17.91
...
53       -67.23   67.36   79.28   64.36   60.81  -80.63  114.09   22.25
54       -66.61   67.07   78.13   64.40   63.21  176.05   79.07    1.98
55       -66.66   66.04   81.55   87.70  178.57 -175.36   72.41   -2.04

[DASH_STATE_STANDARD_DEVIATIONS]
1         14.04   11.17   23.73   15.04    9.99   11.15   12.33   22.90
2         14.19   13.56   35.19   19.73   19.76   27.02   48.91   49.62
3         12.52   10.61   14.07   14.88   12.13   13.45   44.36   65.29
...
53        13.98   15.57   31.09   14.04    9.97   12.49   12.08   14.19
54        14.13   19.23   16.70   20.83   17.38   13.32   28.62   46.80
55        12.16   11.07   55.95   31.46   12.50   33.58   56.68   55.79
```

The DASH-compressed trajectory is defined by two columns containing DASH state numbers and their corresponding bout lengths. This example shows a trajectory spending 21 frames in state 21, followed by 44 frames in state 10, 50 frames in state 11, etc.

```
[DASH_STATE_TRAJECTORY]
State    Frames  Cumulative
```

```
21          21           21
10          44           65
11          50          115
...
30          23        24848
29          54        24902
30          98        25000

[DASH_STATE_TRANSITIONS]
269

[DASH_STATE_REASSIGNMENTS]
1602 (6.41%)
```

The final section collects the bouts spent in each DASH state. Here there are two bouts, of length 190 and 75, in state 1, three bouts, of length 69, 105 and 69, in state 2, etc.

```
[DASH_STATE_BOUTS_(FRAMES)]
1      190 75
2       69 105 69
3       46
...
52      140 33 150 389
53      69 95 232
54      33 77 86 81 83
55      38 78
```

If the inter-frame timestep differs from the default value (1 ps) then a similar block is printed listing the times (ps) spent in each bout.

If the -C flag is used, the details of the initial combined states before the smoothing and roughening procedures take place are printed. This output is placed in blocks with the same format as the corresponding blocks for the Dash states.

The combined states with zero frequency after smoothing are removed to leave a final set of states, which are renumbered at this point. If the -S flag is used, the details of these smoothed states are printed. This output is placed in blocks with the same format as the corresponding blocks for the Dash states.

The circular and cosine similarity matrices between the dash states are printed in two blocks.

```
[DASH_STATE_CIRCULAR_SIMILARITY]
     1      2      3      4      5      6      7      8      9     10     11     12
1   1.00   0.41   0.51   0.54   0.56   0.47   0.51   0.52   0.56   0.47   0.35   0.39
2   0.41   1.00   0.72   0.61   0.49   0.58   0.59   0.48   0.41   0.39   0.53   0.50
...
11    0.35   0.53   0.58   0.54   0.39   0.63   0.60   0.53   0.46   0.42   1.00   0.75
12    0.39   0.50   0.62   0.64   0.37   0.56   0.58   0.60   0.42   0.44   0.75   1.00

[DASH_STATE_COSINE_SIMILARITY]
     1      2      3      4      5      6      7      8      9     10     11     12
1   1.00   0.22   0.55   0.59   0.63   0.52   0.57   0.63   0.71   0.09   0.27   0.40
```

```
2   0.22  1.00  0.78  0.62  0.04  0.61  0.64  0.54  0.30  0.20  0.54  0.54
...
11   0.27  0.54  0.65  0.54  0.19  0.75  0.72  0.68  0.53  0.43  1.00  0.91
12   0.40  0.54  0.73  0.72  0.16  0.71  0.74  0.79  0.48  0.13  0.91  1.00
```

If a principal components analysis is performed, the results are printed in four blocks as follows.

```
[PCA_SUMMARY]
                   PC1         PC2  ...      PC7         PC8
Variance        1.5849      1.0910  ...   0.9035      0.5034
Explained       0.1981      0.1364  ...   0.1129      0.0629
Cumulative      0.1981      0.3345  ...   0.9371      1.0000

[PCA_COEFFICIENTS]
         PC1         PC2  ...      PC7         PC8
      0.0353      0.1326  ...   0.0684      0.0213
      0.1342      0.4136  ...  -0.1826      0.0210
      0.0010      0.7001  ...   0.6221     -0.0225
     -0.6583      0.0435  ...   0.0101      0.7116
      0.6490     -0.0788  ...   0.0521      0.6991
     -0.2247     -0.4481  ...   0.5789     -0.0529
      0.2744     -0.2829  ...   0.4654     -0.0005
     -0.0230      0.1798  ...   0.1429      0.0254

[PCA_WEIGHTS]
         PC1         PC2  ...      PC7         PC8
      0.0012      0.0176  ...   0.0047      0.0005
      0.0180      0.1711  ...   0.0333      0.0004
      0.0000      0.4901  ...   0.3870      0.0005
      0.4333      0.0019  ...   0.0001      0.5064
      0.4212      0.0062  ...   0.0027      0.4888
      0.0505      0.2008  ...   0.3352      0.0028
      0.0753      0.0800  ...   0.2166      0.0000
      0.0005      0.0323  ...   0.0204      0.0006

[PCA_CENTROID]
      ANGLE1      ANGLE2  ...   ANGLE7      ANGLE8
    -64.7678    -38.4875  ...  -4.3546     -2.0389
```

4 Installation

4.1 Obtaining the software

The software is distributed under the terms of the GNU General Public License, version 3 (see the `LICENSE` file for details), and can be downloaded from

```
https://github.com/uop-ibbs/mdash
```

Binary packages are provided for GNU/Linux (RPM and DEB) and for Windows. The source code is provided in two formats:

```
mdash-x.y.z.tar.bz2
mdash-x.y.z.zip
```

Use one of the commands

```
tar xjf dash-x.y.z.tar.bz2
unzip dash-x.y.z.zip
```

as appropriate to unpack the archive into a directory `mdash-x.y.z`.

4.2 Compiling and installing the programs

Instructions for building the programs are given in `mdash-x.y.z/INSTALL.md`.

4.3 Documentation

The files `mdash.html`, `mdash.pdf` and `mdash.info` in the directory `mdash-x.y.z/src/dash` contain documentation in the indicated formats.

4.4 Authors

The DASH algorithm was conceived by Dave Salt, Brian Hudson and Martyn Ford. Matt Ellis produced the first C++ implementation during his PhD, and several Erasmus students (Paul Bonnel, Sylvain Lambert, Renaud Belloncle and Megane Bige) from IUT Belfort, France, have contributed to the program.

4.5 Contact

Any comments, suggestions or bug reports should be addressed to the current maintainer:

Dr. David Whitley (dashsoftware@protonmail.com)