Amorphous Molecular Dynamics Analysis Toolkit (AMDAT)

Development Version Documentation

# Table of Contents

[I. Table of Contents 1](#_Toc392662130)

[II. Overview 4](#_Toc392662131)

[A. Capabilities 4](#_Toc392662132)

[B. Non-capabilities 4](#_Toc392662133)

[C. General Concepts and Terminology 4](#_Toc392662134)

[D. Running AMDAT 5](#_Toc392662135)

[III. Making AMDAT 6](#_Toc392662136)

[A. Required packages 6](#_Toc392662137)

[B. Makefile Options 6](#_Toc392662138)

[IV. Input File 6](#_Toc392662139)

[A. System block 6](#_Toc392662140)

[1. system types 7](#_Toc392662141)

[2. Trajectory file types 7](#_Toc392662142)

[a) xyz 7](#_Toc392662143)

[b) xyz\_log 8](#_Toc392662144)

[c) custom 8](#_Toc392662145)

[d) xtc 9](#_Toc392662146)

[3. Time Scheme 9](#_Toc392662147)

[B. Analysis Block 10](#_Toc392662148)

[1. General commands, logic structures, and variables 10](#_Toc392662149)

[constant 10](#_Toc392662150)

[user\_input 10](#_Toc392662151)

[print 10](#_Toc392662152)

[wait 10](#_Toc392662153)

[exit 10](#_Toc392662154)

[skip 10](#_Toc392662155)

[limit 10](#_Toc392662156)

[processors 11](#_Toc392662157)

[for 11](#_Toc392662158)

[break 11](#_Toc392662159)

[if...else 11](#_Toc392662160)

[evaluate/eval 11](#_Toc392662161)

[round 11](#_Toc392662162)

[floor 12](#_Toc392662163)

[ceiling/ceil 12](#_Toc392662164)

[2. Selecting trajectories for analysis 12](#_Toc392662165)

[create\_list 12](#_Toc392662166)

[thresholded\_list 13](#_Toc392662167)

[invert\_list 13](#_Toc392662168)

[create\_bin\_list 13](#_Toc392662169)

[remove\_bin\_list 13](#_Toc392662170)

[3. Analyzing trajectories 14](#_Toc392662171)

[a) Trajectory Lists 14](#_Toc392662172)

[b) Bin Lists 14](#_Toc392662173)

[c) Tools for use with trajectory lists and bin\_trajectory\_lists: 14](#_Toc392662174)

[composition 14](#_Toc392662175)

[isfs 14](#_Toc392662176)

[msd 15](#_Toc392662177)

[msd\_2d 15](#_Toc392662178)

[structure\_factor 15](#_Toc392662179)

[d) Tools for use with trajectory lists only: 15](#_Toc392662180)

[clustered\_list 15](#_Toc392662181)

[compare\_gaussian 15](#_Toc392662182)

[displacement\_map 16](#_Toc392662183)

[find\_fast 16](#_Toc392662184)

[gyration\_radius 16](#_Toc392662185)

[isf 16](#_Toc392662186)

[isf\_list 16](#_Toc392662187)

[n\_fold 16](#_Toc392662188)

[ngp 17](#_Toc392662189)

[rgtensor\_stats 17](#_Toc392662190)

[stiffness\_dist 17](#_Toc392662191)

[strings 17](#_Toc392662192)

[u2dist 17](#_Toc392662193)

[vac\_fourier 17](#_Toc392662194)

[vac\_function 17](#_Toc392662195)

[vhd 17](#_Toc392662196)

[vhs 18](#_Toc392662197)

[vht 18](#_Toc392662198)

[write\_list\_trajectory 18](#_Toc392662199)

[write\_list\_trajectory\_full 18](#_Toc392662200)

[write\_starr 18](#_Toc392662201)

[e) Tools for use with value\_lists 18](#_Toc392662202)

[autocorrelate\_value\_list 18](#_Toc392662203)

[crosscorrelate\_value\_lists 18](#_Toc392662204)

[f) Obsolete commands 19](#_Toc392662205)

[radial\_debye\_waller 19](#_Toc392662206)

[isfd 19](#_Toc392662207)

[find\_slow 19](#_Toc392662208)

[find\_average 19](#_Toc392662209)

[vector\_autocorrelation\_function 19](#_Toc392662210)

[V. Developer Documentation 21](#_Toc392662211)

[1. class coordinate(coordinate.h, coordinate.cpp) 21](#_Toc392662212)

[2. main 21](#_Toc392662213)

[3. class trajectory 21](#_Toc392662214)

[4. class atom\_trajectory 21](#_Toc392662215)

[5. class molecule 21](#_Toc392662216)

[6. class system 21](#_Toc392662217)

[7. class analysis 21](#_Toc392662218)

[8. class time\_correlation\_function 22](#_Toc392662219)

[9. class mean\_square\_displacement 22](#_Toc392662220)

[10. class van\_hove\_distinct 22](#_Toc392662221)

[11. class spacial\_decomposition 22](#_Toc392662222)

[12. class control 22](#_Toc392662223)

[13. class wave\_vectors 22](#_Toc392662224)

[14. class spherical\_wave\_vectors 22](#_Toc392662225)

[15. class wave\_density 22](#_Toc392662226)

[16. class intermediate\_scattering\_function 22](#_Toc392662227)

[17. class correlation\_2d 22](#_Toc392662228)

[18. class incoherent\_scattering\_function 23](#_Toc392662229)

[19. class debyewaller\_dist 23](#_Toc392662230)

[20. class stiffness\_dist 23](#_Toc392662231)

[21. class non\_gaussian\_parameter 23](#_Toc392662232)

[22. class gaussian\_comparison 23](#_Toc392662233)

[23. class particle\_list 23](#_Toc392662234)

[24. class particle\_list\_exptime 23](#_Toc392662235)

[25. class fast\_particles 23](#_Toc392662236)

[26. class slow\_particles 23](#_Toc392662237)

[27. class average\_particles 23](#_Toc392662238)

[28. class radial\_debye\_waller 23](#_Toc392662239)

[29. class tokenize 23](#_Toc392662240)

[30. class mean\_square\_displacement\_2d 23](#_Toc392662241)

[31. class velocity\_autocorrelation 23](#_Toc392662242)

[32. class strings 23](#_Toc392662243)

[33. class trajectory\_list 23](#_Toc392662244)

[34. class static\_trajectory\_list 24](#_Toc392662245)

[35. class exptime\_trajectory\_list 24](#_Toc392662246)

[VI. III. Change Log 24](#_Toc392662247)

# Overview

## Capabilities

AMDAT is a toolkit of analysis methods well suited for molecular dynamics simulations of noncrystalline materials. It presently can read multiple file formats produced by the LAMMPS molecular dynamics simulation software, and it also can read .xtc format trajectory produced by Gromacs. The type of trajectory file employed may effect the type of analyses that can be performed in AMDAT (more on this later). The most versatile functionality is presently achievable with LAMMPS custom trajectory files. AMDAT enables versatile selection of sets of particles from within a simulation and contains methods of characterizing both the structure and dynamics of the particles. It also enables user-definition of ‘multibodies’ consisting of multiple particles within a molecule and analysis of either the center of mass motion of these multibodies or of multibody correlations between these particles. AMDAT is currently in a beta testing phase, and functionality is being continuously improved or updated. If you encounter any bugs or have requests for additional functionality, please contact Dr. David S. Simmons at [dsimmon@uakron.edu](mailto:dsimmon@uakron.edu). That being said, AMDAT is written in an object oriented manner in C++, and it is designed to enable straightforward extension with new analysis techniques. We welcome submissions of new additions to this code. Some guidance on how to go about this is provided in the developer section of this manual.

## Non-capabilities

AMDAT includes no built-in tools for visualization of either trajectories or properties of trajectories. All results are output in a text-based form and must be viewed in another software (exe excel). AMDAT does not include the ability to aggregate statistics acquired based on multiple simulations; this requires use of post-analysis software, such as matlab, igor, excel, or similar. AMDAT presently does not include multithreading, although this capability is partially implemented and is expected to be introduced in a future update.

## General Concepts and Terminology

AMDAT is run from the command line (see section on Running AMDAT). The user provides an input file (see section on Input File), which provides information on the trajectory file and system metadata (see section on System block) and provides a script specifying analyses to be performed (see section on Analysis Block). AMDAT reads in the specified simulation trajectory (see section on Trajectory file types), and based on this constructs a database of particle trajectories (corresponding the the particles in the simulation trajectory). In order to perform analyses on these trajectories, the selects one or more lists of trajectories (trajectory lists) specifying a set of particle trajectories for analysis (see section on Selecting trajectories for analysis). A number of analysis methods themselves generate trajectory lists, and the set of particle trajectories included in these lists may in general vary over the duration of the simulation trajectory (for example, a trajectory list of the most mobile particles – see analysis method find\_fast – will vary over the course of the simulation as particles become more or less mobile with time). The user can also define “multibodies”, which are entities comprised of multiple particle trajectories (for example, one could define a multibody corresponding to a whole molecule or a single sidegroup). In a similar manner to trajectories, the user may define lists of these trajectories for analysis (for example, calculation of the multibody gyration radius).

**Simulation Trajectory**: The set of configurations being analyzed

**Frame**: The index of a given configuration in a trajectory: frame n is the nth configuration in the trajectory being analyzed. In keeping with C++ conventions, indices throughout are understood to begin at 0.

**Time**: refers to the time corresponding to a given frame

**Lag time or displacement time** or **timegap** or **time spacing**: refers to the time in displacement and time correlation calculations, in which time refers to the time from some arbitrary time origin that may be chosen at many points along the trajectory.

## Running AMDAT

AMDAT is run from the command line, specifying the filename of a user input file:

*./AMDAT <FLAGS>*

<FLAGS> refers to a series of options built in to AMDAT. These flags can given in any combination in any order. Below is a list of flags and their respective syntax.

*-i <INPUT FILE>*

<INPUTFILE> is the path to a file directing AMDAT to a trajectory file for analysis, providing some metadata on the trajectory, and specifying the analysis to be performed. A more detailed description can be found in section 1. <INPUTFILE> is required to run the program.

*-n <# OF PROCESSORS>*

Presently, no analysis methods allow for multithreading, so this option does nothing. Eventually, this will permit AMDAT to run some analyses on <# OF PROCESSORS> processing cores. Defaults to 1 if not given..

*-c <CONSTANT NAME> <CONSTANT VALUE>*

This flag passes to AMDAT the value of a constant (aka, variable) with name by <CONSTANT NAME> with an initial value of <CONSTANT VALUE>. AMDAT will replace any text in the input file with the format “${<CONSTANT NAME>}” with “<CONSTANT VALUE>” during execution. This flag is not required. Constants alternatively may be specified in the input file using the *constant* command.

*-h or –help*

This flag returns a help menu detailing the AMDAT verstion and date of release as well as the command line format detailed here.

-v

This flag returns the AMDAT version and date of release.

-w <TIME>

This flag causes the program to wait for <TIME> seconds before beginning.

> <LOGFILE> 2>&1

This is the standard linux command to redirect screen output to <LOGFILE>. This command must be used after all other flags.

# Making AMDAT

To make AMDAT, cd to the main AMDAT directory and run *make*.

## Required packages

Fftw3 – must be installed

Xdrfile-1.1b – must be installed, provided with AMDAT distribution

tnt\_126 – directory must be specified in CPLUS\_INCLUDE\_PATH, provided with AMDAT distribution and available from NIST at http://math.nist.gov/tnt/download.html

jama125 – directory must be specified in CPLUS\_INCLUDE\_PATH, included with AMDAT distribution and available from NIST at http://math.nist.gov/tnt/download.html

## Makefile Options

Debugging flags can be turned on for use with valgrind by changing comment locations

The flag –fopenmp must be present for both the CFLAGS and CFLAGSmain variables in the makefile in order to build with openmp support (multithreading). Even if you do not plan on using multithreading, this flag is required (probably).

To compile for use on TACC’s Ranger SC, set the variable SERVER in *makefile* to TACC. This removes dependencies from the compilation that are not available on this system and disables several functions. This may also potentially help with compatibility on other large servers. See Analysis Block – Analysis Tools to see which functionality is removed.

Note that the default compilation scheme is to leave the SERVER variable blank (all functionality included).

It is necessary to specify paths to qvector files for inverse space calculations. The location of 3d, 2d, and 1d qvector files must be specified by the makefile variables WAVEVECTORS3D, WAVEVECTORS2D, and WAVEVECTORS1D, respectively. The paths must end in the stem of the files (*/qvector* by default) and must be enclosed in *\”* quotation characters. These data folders are supplied with the AMDAT distribution (qvectors.tar.gz) and may be located anywhere in the file system provided that their location is listed here.

# Input File

## System block

The input file always begins with a block of data specifying the trajectory file and metadata describing the trajectory. This block of data must be entered without blank lines. It is structured as follows

*<system type>*

*<trajectory file type>*

*<filename(s)>*

*<time scheme>*

*<Additional lines of data as req'd by trajectory type, described in “Trajectory file types” section>*

### system types

AMDAT divides trajectories into two types, with the type denoted a the *<system\_type>* keyword: constant volume, denoted by keyword *system* or *system\_nv*, and non-constant volume, denoted by keyword *system\_np*. All AMDAT functionalities are available with constant volume systems, whereas the following limitations apply for non-constant volume systems.

1. The only trajectory file type presently supported for non-constant volume trajectories is *custom*.
2. In order to obtain unwrapped coordinates in non-constant volume systems, either unwrapped coordinates must be directly provided by the custom trajectory file or image index values must be provided (i\_x, i\_y, and i\_z in the LAMMPS custom dump file format). Otherwise, any analysis techniques relying on unwrapped coordinates will not work correctly
3. The “distinct van hove” analysis method does not presently work for non-constant volume systems

### Trajectory file types

Currently recognized file types are *xyz, xyz\_log, custom* and *xtc*.

#### xyz

xyz is a common trajectory file format produced by codes such as LAMMPS. It is a text file containing the positions of each particle at each time frame sequentially. Each frame is headed by two lines. The first reads “atoms” and the second contains the number of atoms in that frame. These are followed by a list of all atoms, where the first column is the atom type index, and the second, third, and fourth are the x, y, and z coordinates of that atom at that time step. Note that AMDAT assumes that the atoms are in the same order in each frame; if the trajectory does not conform to this rule results will generally be incorrect. Since xyz files employ ‘wrapped’ coordinates, they generally face the challenge that they do not always facilitate unambiguous reconstruction of unwrapped particle trajectories. AMDAT will attempt to infer unwrapped trajectories by assuming that the shortest distance travelled by a given particle between two timesteps, considering all possible boundary crossings, is the correct one. If AMDAT encounters trajectories in which displacements approach half the box size in any dimension, it will output a warning indicating that unambiguous particle unwrapping is not possible. If you encounter this warning, it is strongly suggested that you not rely upon any dynamical analysis resulting from this trajectory. This problem can often be solved by employing more closely spaced frames or more generally employing a lammps custom trajectory with either unwrapped coordinates or image flags provided.

Minimally, *<filenames>* for xyz must include the path to the xyz trajectory file. If this is the only file provided, AMDAT will assume that the trajectory is ordered such that all molecules of a given species are grouped together, with the species being listed in the order given in the “additional lines” section below. However, the user may include a second ‘template’ file, which provides the order of molecules by species type. The format is:

*<species\_k\_name> <# of continuous molecules of species k>*

*<species\_j\_name> <# of continuous molecules of species j>*

…

Species in the template file may generally be repeated (ie, molecules of a given species need not be continuous).

Additional lines in system block for xyz:

*<first species name> <number of molecules> <second species name> <number of molecules> …*

*<first atom type> <second atom type> …*

*<# of first type in first species> <# of second type in first species> …*

*<# of first type in second species> <# of second type in second species>…*

*...*

*...*

*<# of first type in last species>…<# of last type in last species>*

*<Lx> <Ly> <Lz>*

*<xlo> <xhi> <ylo> <yhi> <zlo> <zhi>*

Li is the total length of the box in the i’th dimension; ilo is the position of the lower-valued boundary in the i’th direction, and ihi is the position of the upper-valued boundary in the i’th direction.

#### xyz\_log

This format is the same as xyz, but it attempts to employ a LAMMPS log file to determine the box size rather than requiring the user to input it. Accordingly, this requires paths to at least two files to be provided in the *<filenames>* line: the xyz trajectory file and a LAMMPs log file. A third path specifying template file may optionally be provided, with the same effect and formatting as for type xyz.

Additional lines in system block for xyz\_log:

*<first species name> <number of molecules> <second species name> <number of molecules> …*

*<first atom type> <second atom type> …*

*<# of first type in first species> <# of second type in first species> …*

*<# of first type in second species> <# of second type in second species>…*

*...*

*...*

*<# of first type in last species>…<# of last type in last species>*

#### custom

Custom files are a type of trajectory file created by LAMMPS. Currently, AMDAT can only read custom trajectory files that are sorted. Requirements for ordering of the atoms are presently the same as those for xyz, and a template file may also be specified.

Coordinate types understood by AMDAT included wrapped (x, y, and z), scaled wrapped (xs, ys, and zs), unwrapped (xu, yu, and zu), and scaled unwrapped (xsu, ysu, and zsu). At least one complete set of these coordinate types must be present in order for AMDAT to proceed. AMDAT handles this information in the following way.

1. If both scaled and unscaled coordinates of either wrapped or unwrapped type are provided, AMDAT defaults to reading the unscaled coordinates and ignores the scaled coordinates.
2. If only wrapped coordinates are provided, they are read, and the code later attempts to infer unwrapped coordinates from the particle displacements between frames. However, if time between frames sufficiently large that particle displacements are exceeding 40% of the box size in any direction, this may lead to incorrect results due to unwrapping ambiguity.
3. If only unwrapped coordinates are provided, they are read, and AMDAT determines the corresponding wrapped coordinates.
4. If wrapped and unwrapped coordinates are provided, AMDAT reads both.
5. If wrapped coordinates are provided, unwrapped coordinates are not provided, and the ‘box image index’ is provided, then AMDAT reads the wrapped coordinates and determines the unwrapped coordinates by adding to x, y, and z the box image index times the box size.
6. If the box image index is provided by unwrapped coordinates are also provided, the box image index is ignored.

AMDAT also understands velocities (vx, vy, and vz), and masses (m), although these categories are not required and are only used for certain analysis techniques. An additional data category that *must be included* in the custom trajectory file is “type”. These columns may be in any order in the trajectory file. Custom files include the box bounds information in the header of each frame, so this information need not be provided by the user in the input file.

Additional lines in system block for custom:

*<first species name> <number of molecules> <second species name> <number of molecules> …*

*<first atom type> <second atom type> …*

*<# of first type in first species> <# of second type in first species> …*

*<# of first type in second species> <# of second type in second species>…*

*...*

*...*

*<# of first type in last species>…<# of last type in last species>*

#### xtc

This is a binary format produced by the GROMACS molecular dynamics package.

### Time Scheme

The *<time scheme>* line is formatted as follows

*<type> <arguments>*

Formatting for each type is:

type args

*linear <number of timesteps> <time unit>*

Here *<number of timesteps>* is the number of frames in the trajectory and <time unit> is the time between each frame.

*Exponential <# of exponential blocks> <timesteps per block> <exp base> <frt> <first exp> <time unit>*

In an exponential timescheme, frame times are given by the following formula:

where m is the block number (m = 0,1,2,3,…,M) and k is the frame (k=0,1,2,3… if *frt* = 0 and k = 1,2,3,… if *frt* = 1), N is the number of timesteps per block. If *frt* is zero, then an extra initial time frame at time zero is used at the beginning of the trajectory. If *frt* is one, then the first time frame is at time (dt)b^*<first exponent>*. This has significant implications for how time blocks are handled. First exponent is usually zero or one and is the first value of the exponent for the power law progression in time. Deltat is the time unit. For clarity, below is a segment of pseudocode describing the list of simulation times corresponding to this scheme. Also packaged with AMDAT is an example LAMMPS input file yielding a trajectory file corresponding to this time scheme.

block\_starttime=0;

for(blockii=0;blockii<#\_of\_exponentials;blockii++)

{

for(expii=1;expii<=timesteps\_per\_block;expii++)

{

timeii++;

if(pow(exp\_base,expii-1+first\_exponent) <= expii)

{

time[timeii] = block\_starttime+expii\*time\_unit;

}

else

{

time[timeii] = block\_starttime+floor(pow(exp\_base,expii-1+first\_exponent))\*time\_unit;

}

}

block\_starttime = timelist[timeii];

}

}

*Snapshot* (no args)

Snapshot is used for a single frame trajectory.

## Analysis Block

The remainder of the file provides a script specifying the analyses to be performed on the trajectory by AMDAT. AMDAT includes a modest set of logic structures including loops, if structures and variables, described below. It also provides a number of ways of selecting particles for analysis and defining multibodies – entities consisting of multiple particles – for analysis. AMDAT incorporates a considerable number of analysis methods applicable to these selections, also described below.

### General commands, logic structures, and variables

##### constant

*constant <name> <value>*

Sets the constant with name *<name>*  to value *<value>*

##### user\_input

*user\_input*

Pauses script execution and allows the user to manually input commands as though they were part of the input script. This allows the user to run additional analysis, view or change the value of a constant (using **print** or **constant**, respectively), or use any other command available. When prompted, enter the commands line-by-line, pressing “enter” when finished with a line. When completely finished enetering commands, type “done” (without quotes) to execute the commands or “cancel” to cancel execution. When the commands are finished executing (or cancelled), the user will be prompted to enter more commands. This will repeat until the user enters the “done” command with no other commands entered. After exiting, execution will resume where it left off.

##### print

*print ARG1 ARG2…*

Prints everything after the command. Useful for printing the value of a constant: *print ${CONSTANT}*

##### wait

*wait <T>*

Pauses execution for *T* seconds

##### exit

*exit*

Terminates script execution

##### skip

*skip*

*…*

*endskip*

Skips input lines until command “endskip”

##### limit

Sets maximum number of frame pairings to use per time gap. Note that this method is ‘dumb’: it simply cuts off execution of the relevant algorithm at the specified number of data points and does not choose time separations that are statistically independent. We highly recommend not using this command – it should be reserved for situations in which analysis is taking unacceptably long.

##### processors

*processors <N>*

Sets the number of processing cores to be used in multithreaded analysis methods to <N>, where 0 < N ≤ MAXTHREADS, where MAXTHREADS is the number given by the –n flag when executing AMDAT (default 1).

##### for

*for <constant\_name> <initial\_value> <final\_value> <(optional)step>*

*…*

*end*

Loops over the region between *for* and *end*, changing the value of of *<constant\_name>* by *<step>* (defaults to 1) from *<initial\_value>* to *<final\_value>* (exclusive, it never actually equals *final\_value*, identical to *for (int const=init;const<final;const=const+step)* in C++).

If *<initial\_value>* is larger than *<final\_value>* it will automatically decrement *<constant>* by *<step>* until it reaches *<final\_value>.* Changing the value of *<constant\_name>* will not affect the loop, *<constant\_name>* will have its value changed to the next value as if you never changed it.

##### break

*break*

Breaks out of a loop, returning to the code immediately after the *end* command that corresponds to the loop. Only breaks out of one layer of loops. NOTE: old functionality of stopping AMDAT execution is now moved to *exit* command! (V0.410)

##### if...else

*if <val1> <compare> <val2>*

*…*

*else*

*…*

*end*

Compares *<val1>* and *<val2>* using *<compare>*. *<compare>* can be (without quotes): ‘==’, ‘!=’, ‘<’, ‘>’, ‘<=’, ‘>=’

If the comparison evaluates to true, the code between the *if* and *else* (if present) or *end*(if no else present) is executed. If it evaluates to false, then the code between *else* and *end* is executed, if present.

*if ${const1} == 5*

*print It is five!*

*else*

*print It is not five!*

*end*

##### evaluate/eval

*evaluate <constant\_name>* = *<num> <operator> <num> <operator> …*

Evaluates the mathematical expression to the right of the equals sign and saves its value into the constant *<constant\_name>*. Evaluation is done from left to right with parenthesis evaluated first. Currently supported operations are: + - \* / ^ %

*eval C=((${A}^2)+(${B}^2))^(1/2)*

##### round

*round <constant\_name>*

Rounds the value of constant *<constant\_name>* to the closest integer.

##### floor

*floor <constant\_name>*

Rounds the value of constant *<constant\_name>* down to the closest integer.

##### ceiling/ceil

*ceiling <constant\_name>*

Rounds the value of constant *<constant\_name>* up to the closest integer. Can be shortened from *ceiling* to *ceil*.

### Selecting trajectories for analysis

There are generally two ways of storing trajectories for analysis. “Trajectory\_lists” store a list of particle trajectories that may be fixed or may change from frame to frame. “Trajectory\_bin\_lists” divide all particle trajectories into the system at each frame into spatially defined bins for analysis of locally varying properties. Subsets of particles may then be treated on a local basis by analyzing the intersection of a trajectory\_list with a trajectory\_bin\_list (where intersection denotes the set of particle trajectories present in both lists. Following are a list of commands used in the selection of particle trajectories for analysis. In addition to these general commands, certain analysis objects also yield trajectory\_lists. For example, find\_fast returns a trajectory\_list containing the most mobile particles in the system.

##### create\_list

Creates a static list of particle trajectories for analysis based upon set features such as the particle’s type and location within molecules of a given species.

*create\_list <listname>*

*<keyword> <arguments>*

Available keywords and associated arguments are listed below. It is possible to combine multiple keywords and associated arguments to enable selection of complex sets, as follows:

*create\_list <listname>*

*<keyword1 > <arguments1> <keyword2 > <arguments2>*

keyword args

*all none*

Selects all atoms in the system

*type\_system <atom\_type>*

Selects atoms of type *atom\_type*

*type\_species <species\_name> <atom\_type>*

Selects atoms of type *atom\_type* in species *species\_name*

*species <species\_name>*

Selects all atoms within species *species\_name*

*molecule <species\_name> <molecule\_index>*

Selects atoms within a single molecule of species *<species\_name>*, where the molecule is specified by *molecule\_index*, which corresponds to the position of the molecule in the trajectory file within the set of molecules of that species (with the initial molecule of a species indexed as 0.

*type\_molecule <species\_name> <molecule\_index> <atom\_type>*

Selects atoms of type *<atom\_type>* within a single molecule of species *<species\_name>*, where the molecule is specified by *molecule\_index*, which corresponds to the position of the molecule in the trajectory file within the set of molecules of that species

*atom\_species <species\_name> <atom\_type> <atom\_index>*

Selects a single atom of type *<atom\_type>* within every molecule species *<species\_name>*, with the atom being specified by *atom\_index*, which corresponds to the order in which the atoms of that type were first read from the trajectory file for that species (again, indexed from 0).

##### thresholded\_list

Creates a thresholded list of trajectories based on a specified analysis\_value\_list.

*thresholded\_list <analysis\_value\_list name> <thresholded\_list name><threshold type><first threshold><optional:second threshold>*

An analysis\_value\_list is an object that specifies a value (for example, a float) for some or all of the particle trajectories in the system. For example, this could be used to store some measure of local order around each particle. Thresholded list could then be employed to construct a trajectory\_list containing the most ordered particles at each time.

Allowable values for *threshold type* are

*greater* – selects trajectories with values greater than the first threshold

*less* – selects trajectories with values less than the first threshold

*between* – selects trajectories with values between and including the first and second threshold

##### invert\_list

Creates a new trajectory list by inverting an existing trajectory list (meaning it unincludes all included trajectories and includes all unincluded trajectories) and then intersecting it with a second list (meaning it includes only those trajectories in both lists). The net effect of this is to include only trajectories that are not in trajectory\_list 1 but are in trajectory\_list 2.

*clustered\_list <linitial list> < second list><name of inverted list>*

##### create\_bin\_list

Creates a list of all trajectories based on a binning scheme (specific subsets are called in analysis tools by taking the intersection of this list with a trajectory list defined with the *create\_list* command).

*create\_bin\_list <listname>*

*<type> <args>*

Formatting of second line is:

type args

*all* *<n\_xbins> <n\_ybins> <n\_zbins>*

*All trajectories are binned into a (n\_xbins,n\_ybins,n\_zbins) structure where the bins are evenly distributed across the dimensions of the entire system*

*region <n\_xbins> <n\_ybins> <n\_zbins> <xlo> <xhi> <ylo> <yhi> <zlo> <zhi>*

All trajectories are binned similarly to <all>, but the bins are evenly split across the specified region.

*distance trajectory <list\_to\_bin> <list\_to\_take\_distance\_from> <bin\_thickness> <n\_bins>*

All trajectories are binned as a function of distance from the nearest particle in a second list.

*distance point <list\_to\_bin> <x\_coordinate> <y\_coordinate> <z\_coordinate> <bin\_thickness> <n\_bins>*

All trajectories are binned as a function of radial distance from the specified point.

*distance plane <direction> <list\_to\_bin> <axes perpendicular to plane> <position of plane> <bin\_thickness> <n\_bins>*

All trajectories are binned as a function of perpendicular distance from the specified plane. <direction> can be “above” or “below”, which will only include particles above or below the plane, or “average” which will include particles on both sides of the plane.

##### remove\_bin\_list

Removes a bin list from memory, making it no longer accessible for analysis but freeing up any RAM it was occupying.

*remove\_bin\_list <listname>*

### Analyzing trajectories

Most trajectory analysis methods consist of two lines, where the first line (the “type” line) defines the type of analysis to be performed and the output file, and the second line (the “target” line) specifies the atoms or molecules on which the analysis is to be performed. Analysis methods generally have two types of output: output files and trajectory lists. Output files are text files containing results of the analysis. Trajectory lists are lists of atoms and/or molecules identified by the analysis methods that are given a custom name by the user and are stored internally as objects of future analysis.

Currently, target trajectories may be specifies in two ways (denoted by *<target>* herein). The first (and preferred) method is to specify a trajectory list that was defined earlier in the input script. The second is to specify a set of trajectories by name (these do not work with a spatially binned system).

#### Trajectory Lists

The syntax for targetline with this method is

*list <listname>*

*listname* is the name of a trajectory list previously specified by the user. Lists may be created by the command create\_list (described below), which creates a trajectory list corresponding to a trajectory set. They also may be created by various analysis methods.

#### Bin Lists

*bin\_list <bin\_list\_ID> <list\_ID> <persistence check? 0 or 1 (optional)>*

Takes the union of the list specified and the binning structuredefined in *create\_bin\_list* and performs the calculation on each bin individually*.* It should be noted that currently analysis can only be performed on those trajectories that remain in the bin for the duration of the specific timestep (i.e. only persistent trajectories are used for calculations). Also a file extention will be appended to the *<output file>* with the bin indicies (e.g. *<output file>.2.1.2.bindata*)

<Persistence check> specifies whether or not, for dynamic analyses, the algorithm retains only particles that are in the bin at start and end times of a given time gap. The default is zero, meaning this check is not performed.

#### Tools for use with trajectory lists and bin\_trajectory\_lists:

##### composition

Calculates the composition and number density

*composition <output file>*

*<target>*

##### isfs

Calculates self-part of the intermediate scattering function. MC

***isfs*** *<output file> ><first wavevector index> <last wavevector index> <geometry> <max\_length\_scale> <(optional) fullblock>*

*<target>*

*<first wavevector index> and <lastwavevector index>* give the limits wave vectors to be calculated. Options for *<geometry>* are *xyz, xy, xz, yz, x, y,* and *z*. This chooses which dimensions in k-space to include in the calculation of the intermediate scattering function. *xyz* computes the full radial three dimensional isf, *xy, yz, and xz* calculate two-dimensional in-plane radial isf’s, and *x, y, and z* compute one-dimensional isf’s. *<max\_length\_scale>* determines the longest distance which will be decomposed into inverse space. If a distance of 0 is given, the full box size is used*. <(optional) fullblock>* is an optional argument that can be either 0 or 1. The default is 0, in which case time spacings spanning multiple blocks use only the first time of each block. A setting of 1 specifies that it should use all block times for times spanning blocks. This may result in substantial computational time increases but offers the possibility of modestly increased data quality at very long times.

##### msd

Calculates mean square displacement. MC

*msd <output file>*

<target>

##### msd\_2d

Calculates mean square displacement in two dimensions

*msd <output file> <plane:”xy”,”yz”,”xz”>*

*<target>*

##### structure\_factor

Calculates structure factor.

*structure\_factor <output file> <symmetry> <geometry> <max\_length\_scale> <(optional) fullblock>*

*<target>*

*<target 2(optional, if symmetry = asymmetric only)*

Options for *<geometry>* are *xyz, xy, xz, yz, x, y,* and *z*. This chooses which dimensions in k-space to include in the calculation of the intermediate scattering function. *xyz* computes the full radial three dimensional isf, *xy, yz, and xz* calculate two-dimensional in-plane radial isf’s, and *x, y, and z* compute one-dimensional isf’s. *<symmetry>* is either *symmetric* or *asymmetric*. If *symmetric*, the analysis calculates the structure factor between the set of particles specified in target and itself. If *asymmetric*, then a second target must be specified, and the partial structure factor describing correlations of *<target 1>* with only *<target 2>* is calculated. *<first frame>* and *<last frame>* are the indices of the limits on time spacings to be calculated.

*<max\_length\_scale>* determines the longest distance which will be decomposed into inverse space, if a distance of 0 is given, the full box size is used*. <(optional) fullblock>*is 0 if structure factor should only be calculated at the start of each block, 1 if it should be calculated at all times

**trajectory\_list\_decay**

Calculates how a trajectory list has decayed over a period of time. Currently works by averaging the decay between each block

*Trajectory\_list\_decay <output file>*

*<target>*

#### Tools for use with trajectory lists only:

##### clustered\_list

Generates a clustered trajectory list.

*clustered\_list <linitial list> < name of clustered list><sigma matrix file><plane><primary><secondary>*

*<primary> is the number of neighbors required to be in the initial list to be in the clustered list. <secondary> is the number of neighbors required to be in the clustered list to be in the clustered list.*

##### compare\_gaussian

Find points where the self Van Hove crosses the Gaussian approximation to the self Van Hove.

*compare\_gaussian <output file> < displacement time index (optional)>*

*<target>*

Requires prior calculation of self van hove in the same run. If only two arguments are given, this analysis first calculates the non-Gaussian parameter and used the time spacing corresponding to the peak in the nGP. If a third argument is given, self Van Hove will be compared to the Gaussian approximation at the time index given.

##### displacement\_map

Generates maps of particle displacement magnitude and writes to pdb files with value beta specifying displacement.

*displacement\_map <output file name> <index of time spacing> <first block> <last block> <optional: max displacement to write to pdb>*

##### find\_fast

Find particles that are ‘fast’ based on the above comparison between the self Van Hove and Gaussian approximation.

*find\_fast <list name> <output file>*

*<target>*

*<list name>* designates the name of a list of fast particles to be created by this analysis that can be used as targets for other analyses (*note: this tool cannot be used with lists, trajectory sets must be used*).

##### gyration\_radius

Very limited method only applicable to simulations of a single molecule centered at the origin of the box. Under this circumstance, calculates gyration radius of this molecule.

*gyration\_radius <output file> <species index> <molecule index>*

*<species index>* and *<molecule index>* are species and molecule indices of the target molecule.

##### isf

Calculates the intermediate scattering function. This analysis only permits the use of trajectory descriptors (not lists) in the target.

*isf <output file> <method> <symmetry> <geometry> <(optional) fullblock> <additional args>*

*<target>*

*<target 2(optional, if symmetry = asymmetric only)*

*<method>* is either *manual* or *automatic*. If *automatic*, include additional args *<first wavenumber index> <last wavenumber index>,* specifying by index the wavenumber range to be calculated. If manual, include additional arg *<structure factor output filename>*, which the user can view before being prompted to manually enter a wavevector range. *<symmetry>* is either *symmetric* or *asymmetric*. If *symmetric*, the analysis calculates the structure factor between the set of particles specified in target and itself. If *asymmetric*, then a second target must be specified, and the partial structure factor describing correlations of *<target 1>* with only *<target 2>* is calculated. Options for *<geometry>* are *xyz, xy, xz, yz, x, y,* and *z*. This chooses which dimensions in k-space to include in the calculation of the intermediate scattering function. *xyz* computes the full radial three dimensional isf, *xy, yz, and xz* calculate two-dimensional in-plane radial isf’s, and *x, y, and z* compute one-dimensional isf’s. *<(optional) fullblock>*is either 0 or 1: if 0, time displacements across multiple blocks use only the first frame in each block, if 1, they are average over all frames. . *<first frame>* and *<last frame>* are the indices of the limits on time spacings to be calculated.

##### isf\_list

Calculate intermediate scattering function, using a trajectory list.

*Isf\_list <output file> <list name> <plane> <first wavenumber index> <last wavenumber index>*

Options for *<geometry>* are *xyz, xy, xz, yz, x, y,* and *z*. This chooses which dimensions in k-space to include in the calculation of the intermediate scattering function. *xyz* computes the full radial three dimensional isf, *xy, yz, and xz* calculate two-dimensional in-plane radial isf’s, and *x, y, and z* compute one-dimensional isf’s.

#### n\_fold

Calculates mean and time dependence of n-fold order parameter and writes maps to files.

*nfold <output file name> <n-fold order> <plane:”xy”,”yz”,”xz”> <neighbor cutoff dist.> <sigma matrix file> <analysis\_value\_list name> <optional 2: map file name stem> <optional 2: first map time> <optional 2: last map time>*

*<target>*

*<sigma matrix file>* refers to a file that describes the cross interaction σ values. It is formatted as described below:

*<arbitraty name for species 1> <> <> <> … <>*

*<arbitraty name for species 2> <> <> <> … <>*

*.*

*.*

*.*

*<arbitrary name for species n> <> <> <> … <>*

##### ngp

Calculates non-Gaussian parameter of the mean squared displacement.

*ngp <output file>*

*<target>*

##### rgtensor\_stats

Calculates statistics describing the radius of gyration tensor characterizing particle trajectories as a function of time.

*rgtensor\_stats <eigenvalue output file> <relative asphericity distribution output file> <# asphericity bins> <radius of gyration distribution output file> <# of rg bins> <max rg binned>*

*<target>*

##### stiffness\_dist

Calculates distribution of inverse Debye-Waller factor values 1/u2.

*stiffness\_dist <output file> <# of bins> <maximum stiffness value> <displacement time index>*

*<target>*

*<displacement time index>* is the time spacing at which the Debye-Waller factor is to be defined.

##### strings

*strings <output file> < timestep index> <threshold> <sigma matrix file> <trajectory\_list\_name>*

*<target>*

Calculates mean string length, mean number of strings, string order parameter, and distribution of string lengths. The target should be a trajectory list of mobile particles. This can be produced via the “find\_fast” command. Threshold sets the threshold (as a multiple of sigmaij for each particle pair) within which one particle is considered to have replaced another. A standard value for this is 0.6. Also creates a trajectory list called <*trajectory\_list\_name>* containing all particles in strings for further analysis.

Sigma matrix file specifies the sigma­­ij for all interaction types as described in the **n\_fold** command.

##### u2dist

Calculates distribution of square displacements at a specified time.

*u2dist <output file> <# of bins> <max u2 binned> <displacement time index>*

*<target>*

*<displacement time index> is the time spacing at which the Debye-Waller factor is to be defined.*

##### vac\_fourier

Calculates fourier transform of velocity autocorrelation. This works only for trajectories that are linearly spaced in time.

*vac\_fourier <output file>*

*<target>*

##### vac\_function

Calculates velocity autocorrelation function by taking two derivatives of the mean-square-displacement. This works only for trajectories that are linearly spaced in time (not for exponential timeschemes). It also requires that frames be closely spaced in time in order to give high quality results. *Note that this method presently does not employ velocity data directly!*

*vac\_function <output file>*

*<target>*

##### vhd

Calculates distinct part of the Van Hove function.

*vhd <output file> <min cell size> <maximum range binned> <# of bins>*

*<target>*

This algorithm uses a spatial decomposition of the box into cells, with cells guaranteed to be of size *<min cell size>* or larger (the algorithm will select the minimum cell size that is an integer divisor of the box size and is larger than *<min cell size>*.

##### vhs

Calculates self-part of the Van Hove function.

*vhs <output file> <maximum range binned> <# of bins>*

*<target>*

##### vht

If a self-Van Hove and distinct Van Hove have previously been calculated, this adds them together to obtain the total van hove.

*vht <output file>*

##### write\_list\_trajectory

Writes to file an xyz trajectory from a trajectory list.

*write\_list\_trajectory <list name> <trajectory output filename>*

##### write\_list\_trajectory\_full

Writes to file an xyz trajectory from a trajectory list where all particles not included are written arbitrarily at position (1000,1000,1000). This may be useful for visualization software that requires a contant number of particles (exe. VMD)

*write\_list\_trajectory <list name> <trajectory output filename>*

##### write\_starr

**write\_bin\_xyz**

Writes an xyz trajectory file from a binned trajectory list.

*write\_bin\_xyz <bin\_list\_ID> <filename>*

*<type> <args>*

type args

*all n/a*

*single <x\_index> <y\_index> <z\_index>\**

\*Note that indicies start at 0.

#### Tools for use with value\_lists

##### autocorrelate\_value\_list

Calculates the time autocorrelation of a value\_list.

*autocorrelate\_value\_list <output file> <value\_list 1>*



where f is the value listed, i is the particle index, N is the number of particles in the list, and  is the average value of f over all particles listed and all times . Also, note that this method does not restrict time zero to the first frame of the trajectory, but uses all available times (in accordance with the choice of either linear or exponential time scheme) as the time zero. This expression assumes the system to be at steady state.

##### crosscorrelate\_value\_lists

Determines the mean time cross-correlation of two value lists

*crosscorrelate\_value\_lists <correlation type> <output file> <value\_list 1> <value\_list 2>*

<correlation type> is either *static* or *dynamic*. If static, it will output a single number providing the static correlation between the two value lists:



If dynamic, it will write to file the time crosscorrelation function of the two.



where f is the value stored in *value\_list 1* and g is the value stored in *value\_list 2*, and , i is the particle index, N is the number of particles in the list, and  and  are the average value of and g, respectively over all particles listed and all times . Also, note that this method does not restrict time zero to the first frame of the trajectory, but uses all available times (in accordance with the choice of either linear or exponential time scheme) as the time zero. This expression assumes the system to be at steady state.

#### Obsolete commands

Several analysis methods are outdated and intended for removal from AMDAT but are technically still available. Use them at your own risk.

##### radial\_debye\_waller

Calculates debye-waller factor as a function of distance from a given point.

*radial\_debye\_waller <output file> <time spacing index> <# of bins> <maximum range> <xorigin> <yorigin> <zorigin>*

*<target>*

*Note that this analysis does not do any box unwrapping – if one of your bins includes space outside the simulation boundaries, it will just not include the space in its calculation. <time spacing index> gives the time spacing at which the debye-waller factor is defined.*

##### isfd

Calculates distinct part of the intermediate scattering function. This method is obsolete and may give incorrect results.

##### find\_slow

This doesn’t really do anything physically meaningful – it is obsolete.

*find\_slow <list name> <output file>*

*<target>*

##### find\_average

This doesn’t really do anything physically meaningful – it is obsolete.

*find\_average <list name> <output file>*

*<target>*

##### vector\_autocorrelation\_function

Calculates total and orientational autocorrelation function of vectors (bonds) connecting pairs of atoms, specified by the user.

*vector\_autocorrelation\_function <output file> <vector list file>*

This analysis method does not use a <target> line, the analysis target is total specified in the vector list file, which has the following format.

First line: number of vectors

Remaining lines: vectors, specified by the format

*<species name> <1st atom type name> <1st atom index in molecule> <2nd atom type name> <2nd atom index in molecule>*

Vectors can only be specified between atoms within the same species, and the method will average over this vector in every molecules of the specified speices. Atom index is numbered from 0 to n (where n is the number of atoms of a given type in a molecule of a given species) and is determined by the order in which the atoms of that type were listed in the trajectory file. The quantities calculated are as follows.

Total vector autocorrelation function:



Orientational vector autocorrelation function



Where N is the number of vectors, i is an index over vectors, r denotes the vector between the two specified atoms, bars denote vector magnitude, and brackets denote an ensemble average.

# Developer Documentation

This developer documentation is somewhat out of date, but it may still provide some help.

Broadly speaking, there are two parts of the AMDAT code: 1) the data structure and 2) analysis classes.

The purpose of the data structure is to store *trajectories*. A trajectory is a sequence of coordinates in time, with some additional metadata. Common trajectories include trajectories of atoms (or particles), and potentially also molecular centers of mass. The core data structure includes the classes System, Trajectory, Molecule, Atom\_Trajectory, and Coordinate. Molecule and Atom\_Trajectory are both daughter classes of Trajectory, and any Trajectory contains an array of Coordinates.

The analysis classes comprise a set of classes which operate on this data structure to yield output data. There are generally two ways by which an analysis class may interact with the data structure:

1. It may use preexisting loops in the System class to loop over rational subsets of trajectories.
2. It may use an intermediary set of classes, Trajectory\_List and its daughters, to loop over a subset of trajectories previously defined by the user.

Presently, all analysis classes should preferably be made compatible with both methods; however, method 2) should be given priority as method 1) may be phased out at a later time.

Classes shown in blue are those that inherit the functionality of parent class analysis. Classes shown in green are part of the

### class coordinate(coordinate.h, coordinate.cpp)

Class to store a coordinate in three dimensions, with some associated methods and operator definitions.

### main

(amdat.cpp)

### class trajectory

(trajectory.h, trajectory.cpp)

Stores information about a particle, molecule, or point and its trajectory through time (stores an array of coordinates)

### class atom\_trajectory

(atom\_trajectory.h, atom\_trajectory.cpp)

Stores information about an atom (particle) and its trajectory through time (stores an array of coordinates)

### class molecule

(molecule.h, molecule.cpp)

Stores information about a molecule, its substituent atoms (stored in arrays) and in some cases the center-of-mass trajectory of the molecule

### class system

(system.h, system.cpp)

This is the master class that reads trajectory files and stores information about their contents, including: molecules, atoms, time scheme, box size, and so on. Also contains methods to run loops over subsets of atoms and molecules as well as over times for use by analysis objects.

### class analysis

(analysis.h, analysis.cpp)

Parent class for many analysis techniques, with methods for interfacing with the atom, molecule, and time loops provided by the system class.

Basic public methods for running this

analyze(string)

analyze(Trajectory\_List\*)

### class time\_correlation\_function

Parent class for several analysis classes that calculate a time correlation function.

(time\_correlation\_function.h, time\_correlation\_function.cpp)

class van\_hove\_self

Analysis class to calculate self van hove.

(van\_hove\_self.h, van\_hove\_self.cpp)

class progress

(progress.h, progress.cpp)

Class to track progress of a calculation.

### class mean\_square\_displacement

(mean\_square\_displacement.h, mean\_square\_displacement.cpp)

Class to calculate mean square displacement as a function of time.

### class van\_hove\_distinct

(van\_hove\_distinct.h, van\_hove\_distinct.cpp)

Class to calculate distinct van hove.

### class spacial\_decomposition

(spacial\_decomposition.h, spacial\_decomposition.cpp)

Class to spatially decompose the system into smaller cells by sorting atoms at each time into these cells.

### class control

(control.h, control.cpp)

Master class that reads input file and calls other objects accordingly.

### class wave\_vectors

(wave\_vectors.h, wave\_vectors.cpp)

### class spherical\_wave\_vectors

(spherical\_wave\_vectors.h, spherical\_wave\_vectors.cpp)

### class wave\_density

(wave\_density.o, wave\_density.cpp)

### class intermediate\_scattering\_function

(intermediate\_scattering\_function.h, intermediate\_scattering\_function.cpp)

### class correlation\_2d

(correlation\_2d.h, correlation\_2d.cpp)

### class incoherent\_scattering\_function

(incoherent\_scattering\_function.h, incoherent\_scattering\_function.cpp)

### class debyewaller\_dist

(debyewaller\_dist.h, debyewaller\_dist.cpp)

### class stiffness\_dist

(stiffness\_dist.h, stiffness\_dist.cpp)

### class non\_gaussian\_parameter

(non\_gaussian\_parameter.h, non\_gaussian\_parameter.cpp)

### class gaussian\_comparison

### class particle\_list

### class particle\_list\_exptime

### class fast\_particles

### class slow\_particles

### class average\_particles

### class radial\_debye\_waller

### class tokenize

### class mean\_square\_displacement\_2d

### class velocity\_autocorrelation

### class strings

### class trajectory\_list

### class static\_trajectory\_list

### class exptime\_trajectory\_list

# III. Change Log

AMDAT 0.500

* Enabled custom definition of “multibodies” – sets of multiple particle trajectories within the same molecule on which analyes relevant to multibody correlations and so on and be performed
* Coded “gyration radius” multibody\_analysis method that calculates the mean gyration radius of a list of multibodies
* Enabled custom defition of new trajectories corresponding to the center of mass or centroid of above trajectories
* Eliminated all old functionality associated with molecule center of mass calculation and the old method for calculating molecule gyration radius (which worked only for a single molecule centered on the box center). This functionality has been subsumed by multibody methods.
* Made extensive updates to storage of trajectory\_lists, etc to employ hash tables for more efficient lookup
* Eliminated capability for analysis methods to use system loops – now they must use trajectory lists only
* Removed a number of deprecated functionalities such as particle lists and updated code to work without them
* Disabled multithreading to to concerns over how data from separate processors was being merged

AMDAT 0.430

* Added initial multithreading capability to the following analysis methods: msd, isfs, structure\_factor
* Added commands: processors, floor, ceiling.
* Added support for parenthesis and modulus (%) in *evaluate* command. Removed support for the word ‘equals’ and requires using ‘=’.
* AMDAT version is now displayed in the screen output at the start of execution.
* To update an analysis method to use multithreading you must remove all references to global variables that are changed in the analysis method and add “bool isThreadSafe(){return true;};” to the header file of the analysis method. In addition, you must actually cause the multithreading to happen, look at the system.cpp -> displacement\_list method for an example. MSD/ISFS multithread in system.cpp, SF multithreads in trajectory\_list.cpp

AMDAT 0.421

* Added new analysis method mean\_displacement, which calculated the mean displacement as a function of time. Its primary purpose is to look for momentum buildup in simulations.

AMDAT 0.420

* Corrected time weighting out of exponential blocks – it was previously bugged . It is likely still bugged for frt = 1, but it should work properly for frt = 0.
* Changed meaning of number of timesteps for linear time scheme. It is now number of frames minus 1.
* Corrected bug in creation of trajectory list bins based on a plane, in the ‘above’ direction. It was previously not handling box wrapping correctly.

AMDAT 0.410

* Nearly complete rewrite of input file handling. No longer creates temporary input file and instead uses a vector of strings with each line as a seperate element. No obvious difference to end user except that there are no temporary files, but allows for most of the new functions this version.
* Initial implementation of commands: remove\_bin\_list, print, wait, constant, round, for, evaluate/eval, exit, if/else, user\_input.
* Constants are now able to have their value changed during execution. This is primarily done using the commands: constant, evaluate, for, and round.
* Changed *break* command functionality to *exit* and repurposed *break* for breaking out of loops.
* Added ability to remove bin lists after creation to free up memory. Uncommented several “delete” commands in trajectory\_list\_bins which has reduced the amount of memory needed to create additional bin lists.
* Lines in the input script can be commented out using the “#” symbol (without quotes) in the same way that it is used in BASH. This includes using in the middle/end of lines (proper commenting of AMDAT scripts can now happen)
* Comparison of box sizes between initial and current fatime now uses a float comparison that includes a small margin of error for rounding.

AMDAT 0.400

* **Critical fix -**  Unwrapping for isfs was not being handled correctly previously and has been fixed. Previous version worked correctly for cubic boxes if the box size size was used to determine wavenumbers. In other systems, typically very small errors were usually introduced by this problem.
* Changes to strings
  + Strings analysis method now only accepts one timegap (multiple timegaps did not work properly before)
  + Strings analysis method now creates a trajectory list of the particles in strings; an additional argument for the name of the trajectory list has been added at the end.

AMDAT 0.392

* Implemented trajectory\_list\_decay.

AMDAT 0.391

* Fixed issues with certain directions in binning as by distance from a plane

AMDAT 0.390

* updated binning as a function of distance to include number of bins and to include binning only above or below the plane.
* qvector file paths are now defined in the makefile at compile time rather than in the input file at runtime
* new qvector files have been freshly generated for 3d, 2d, and 1d
* All analysis methods that use qvectors should now also work with 1d options. The list of options is now xyz, xy, xz, yz, x, y, and z.

AMDAT 0.381

* Implemented binning as a function of distance from a plane or point.
* Added –t flag

AMDAT 0.380

* Implemented binning with structure factor, only works for symmetric.
* Implemented binning as a function of distance from another trajectory list.
* Corrected msd,composition,structure factor and isfs to return a value of 0 when no particles are in the trajectory list.
* Added –w flag
* Added error message if constants are not defined.

AMDAT 0.372

* Fixed an error in new binning code that caused segmentation faults when particles were on the bin boundary.

AMDAT 0.371

* New results have revealed that there is still a bug in center-of-mass determination. Center-of-mass analysis should not be trusted.
* Fixed an error in new binning code that caused segmentation faults when particles were outside of binned region.

AMDAT 0.370

* Modified and fixed analysis with bins. Bin analysis now defaults to using the non-persistence checked version. Adding a 4th argument to the target line of “1” now turns on persistence checking for dynamic analyses. Also made changes to improve memory management so that binning and binned analysis should not encounter serious memory problems.

AMDAT 0.360

* Updated command line syntax for running AMDAT
  + Constants now taken from command line with –c flag. Constants can no longer be defined within the input file.
  + –i flag now required before input file
  + –h updated
  + –l flag no longer works. Use standard linux command to redirect screen file
  + –v flag updated to include release date
* ‘Constants’ parsing improved to prevent an eror that disallowed consecutive constants in some cases. For real this time.
* Various manual updates.

AMDAT 0.351

* Can no longer create list from a single bin, this is due to improved functionality for binning (higher density of bins now possible)
* Changes to how bins are stored:
  + trajectory lists for bins are calculated on the fly rather than stored internally
  + For reference: on a systemwith 12GB RAM, no more than 100 bins are currently possible for a system of 4000 particles when calculating composition
  + Future changes will allow >1000 bins for all analysis techniques, stay tuned.
* Fixed error in wave\_vectors.cpp that caused incorrect decomposition of system into shells
* Fixed elapsed time output for isfs.
* ‘Constants’ parsing improved to prevent an eror that disallowed consecutive constants in some cases.
* Various manual updates for clarity.

AMDAT 0.350

* Added bin\_static\_analysis and implemented w/ composition; added big 3 to composition
* Added density composition to composition analysis method
* Eliminated analysis\_value\_list (merged into value\_list) and made extensive changes corresponding to this
* Added autocorrelate\_value\_list
* Added crosscorrelate\_value\_lists
* Added vector\_autocorrelation\_function
* Fixes to Strings (still needs a bit more additional verification, but looks like it probably is working right now)
  + now uses matrix of particle sizes to determine threshold
  + corrected algorithm error that was causing severe undercounting of strings
  + corrected algorithm error that was leading to strings of length zero
* Incoherent scattering function (isfs) now works with bin trajectory lists
* Altered incoherent scattering function to default to *not* using multiple parallel times for time spacings across blocks, but to instead just use the first time of each block. The user can now include an additional, optional, argument, of 0 or 1, with 1 specifying to use all times across blocks and 0 simply explicitly using the default setting.
* Changed doubles in incoherent\_scattering\_function to floats for reduced memory usage, but retained double math in some of the core algorithm to prevent substantial rounding error.
* ngp (non\_gaussian\_parameter) now works with trajectory\_bin\_lists and has big 3

AMDAT 0.340

* Added clustered\_list
* Added invert\_list
* Fixed asymmetric structure factor
* Added write\_trajectory\_list\_full() method to write out xyz with dummy values for particles not in the trajectory list

AMDAT 0.330

* Corrected output of thresholds from compare\_gaussian
* String method now writes image file to a file named by taking the filename given by user and appending “\_img.xyz”
* Created new Structure\_Factor class that calculates structure factor with greatly reduced memory use. Modified control and input syntax accordingly.
* Made corrections to molecule center of mass calculation. Eliminated valgrind memcheck errors. Some additional warnings regarding ambigious unwrapping are possible now.

AMDAT 0.321

* Fixed Bug in system preventing the use of xyz files.

AMDAT 0.320

* Changed functionality of structure\_factor() in control and wave\_density to reduce calculation time of wave densities.
* Added copy constructor, default constructor, assignment operator, and destructors to
  + intermediate\_scattering\_function
  + wave\_density
  + wave\_vectors
* Changes to remove unsupported dependencies for usage on TACC’s Ranger SC cluster. Makefile and various methods changed.
* Added ability to create a list from an individual bin from a previously created bin list.
* Bug fixes
  + define\_constant: multiple uses of a variable in a line
  + create\_bin\_list: type *region* now functions properly (added conditional statement to trajectory\_list\_bins::bin\_boolean\_calculation() ).
  + custom: boolean definitions in system updated.

AMDAT 0.311

* Made corrections to xyz\_log for reading box size

AMDAT 0.310

* Made some corrections to read-in of custom dump files and template files

AMDAT 0.300

* Added ability to read non-constant volume files.
* Modified multiple analysis methods to work with new non-constant box dimensions.
* Corrected problems with read-in of custom trajectory with template file
* Improved algorithm for calculating molecule center of mass trajectories. The algorithm now searches for the first time at which center of mass calculation is unambiguous and then uses unwrapped particle displacements to determine the COM position at all other times. As a consequence, instead of requiring COM calculation to be unambiguous at *all* times, the algorithm only requires it to be unambiguous at *one time*. In the case that a molecule has no unambiguous time, the algorithm will print a single warning for that molecule COM, a considerable reduction from the large number of warnings frequently generated previously.

AMDAT 0.260

* Can now read LAMMPS custom dump files (this feature is still in beta and should be tested further)
* Added command “define\_constant”, which allows user to define a constant in input file which will be replaced any time it is found. Constant can be used with the syntax *${constant\_name}*.
* Can now determine box bound information from a LAMMPS log file rather than from input file, if log file is provided
* Corrected an error in the way trajectory type is determined for center of mass trajectories
* Added ability to read in particle velocity information from a custom LAMMPs dump file and store it in trajectory objects; also added methods to class trajectory to interact with this velocity data.

AMDAT 0.250

* Structure factor updated to work in 2D with plane options.
* msd\_2d compatible with binning.
* Unused variables removed to eliminate warning messages on compilation.
* Corrected center-of-mass calculations.

AMDAT 0.240

* Binning scheme added with limited functionality, see documentation for capabilities.
  + New classes: trajectory\_list\_bins and bin\_dynamics\_analysis
  + Significant changes to control including new run\_analysis template to be used with any method that uses binning
  + write\_bins\_xyz capability included, see documentation
  + Any analysis method must have the following changes to utilize binning (see msd for example)
    - Copy constructor
    - Destructor
    - Assignment operator
    - bin\_hook method to allow bin\_dyanmics\_analysis’s displacement kernel to call an analysis object’s displacement kernel.
    - Must use run\_analysis template in control
* Assignment operator, copy constructor, and destructor added for trajectory\_list, boolean\_list, and mean\_square\_displacement.
* bin\_hook method added to msd.
* hexatic\_order\_parameter replaced with n\_fold\_order\_parameter
* Added composition class, which allows computation of the time dependent and average compositions
* added analysis\_value\_list, which stores an array of value\_lists
  + update to control to be able to call stored analysis\_value\_lists
* added skip command to control

AMDAT 0.230

* Corrections and assignment operators for trajectory\_list and Boolean\_list
* Made some OOP and const-correctness improvements to the core code. This may give some minor performance improvements and should help in error prevention.
* Use of passing by const reference where appropriate to improve efficiency
* Use of const class methods where appropriate
  + Returning constant pointers from class methods when appropriate to prevent unintended/unauthorized modification of internal data members
* In trajectory list public ‘set’ command taking a Boolean\_List pointer argument, switched from copying pointer over to doing a deep copy of the Boolean\_List objects referred to by the pointer – this should prevent unintentional modification of the included trajectory list by external code
* Began adding copy constructors and assignment operators to all classes where the defaults are inadequate due to use of pointers.

AMDAT 0.220

* Eliminated spurious warning associated with molecule center of mass trajectory calculation
* Updated value\_list.h class with new write\_pdb method that allows the class to write a pdb file that VMD will read and color according to the values stored in value\_list.h
* Corrected ngp calculation via trajectory list method – it was incorrect in prior versions
* Slightly modified the way in which ‘ambiguous unwrapping’ warnings are generated in a way that should catch more ambiguous unwrappings
* Added hexatic\_order\_parameter class, which allows computation of average and time dependent hexatic order parameter and also enables creation of hexatic order parameter maps. This class works only for 2-D and pseudo-2-D systems, and presently is not implemented for center of mass trajectories, only for single atom trajectories. It also presently excludes particles within a cutoff distance of the edge of the box.
* Added displacement\_map class, which generates a map (or maps) of particle displacements at user-specified times.

AMDAT 0.210

* Added option for additional command line parameter specifying file to redirect log output to.
* Corrected errors related to unwrapping and molecule COM calculation warning generation

AMDAT 0.200

* Created new timetype snapshot, which loads a single frame (the first) from a trajectory
* Internally reworked system loops to pass all trajectory objects by reference rather than passing their species, molecule, type, and atom indices
* Corrected calculation of molecule center of mass trajectories and made the calculation automatic during system construction
* Most analysis techniques should now work with molecule center of mass trajectories
* Created new trajectory set (analysis target) commands moleculecom\_all and moleculecom\_species, which perform analysis on molecule centers of mass of all molecules and molecules of a given species, respectively.
* Created new analysis.write(string) command which serves as a prototype for data writing commands for all analysis classes. All data writing by analysis classes must now be handled by this command.
* Fixed structure factor and intermediate scattering function calculation from lists
* Created new command *write\_list\_trajectory*, which writes an xyz trajectory file from a trajectory list.

AMDAT 0.130

* Created class Boolean\_List, which stores a 0 or 1 for every trajectory in the system. Objects of this class are used as an inclusion list by Trajectory\_List class to allow for Boolean comparisons between trajectory lists.
* Created templated class Value\_List, which stores a value of variable type for every trajectory in the system, and also stores a Boolean\_List, signifying which trajectories are included
* Enhanced Trajectory\_List classes to support Boolean operations between Trajectory\_List objects and between times of a given object
* Expanded rgtensor\_stats command and class to output additional data about rgtensor distributions.