Aqua-Duct Documentation

Release 0.3.1

Tomasz Magdziarz
Karolina Mitusińska
Sandra Gołdowska
Alicja Płuciennik
Michał Stolarczyk
Magdalena Ługowska
Artur Góra

Feb 04, 2017

CONTENTS

CHAPTER

ONE

AQUA-DUCT INSTALLATION GUIDE

1.1 Overview

Aqua-Duct software is software written in Python (CPython) and comprises of two elements:

1. aquaduct - a Python package,
2. valve - a script that uses `aquaduct` to perform calculations.

Download

You can download Aqua-Duct packages directly from Aqua-Duct homepage. This page includes older versions of Aqua-Duct as well as development version.

If you follow this installation guide you will install current release.

1.2 Troubleshooting

If you encounter any problems with installation do not hesitate to contact us at info@aquaduct.pl. We are **REALLY** willing to help!

Please, provide us with us much info as you can. In particular try to include following information:

- Operating system's name and version, and CPU architecture (if relevant).
- Python version.
- Command(s) you have used for installation.
- Any error/warning/info message(s) that emerged during or after installation.

1.3 Requirements

1.3.1 Software-wise requirements

- **Python 2.7 (CPython implementation)**
  - numpy >= 1.7.0
  - scipy >= 0.14.0
  - scikit-learn >= 0.16.0
  - MDAnalysis[amber] >= 0.15.0
  - roman >= 2.0.0

1.3.2 Hardware-wise requirements

Aqua-Duct should work on every machine on which you can install the above mentioned software. On computers older then 10 years it may work very slow though. We recommend 64bit SMP architecture, with at least 4GB RAM (32 GB RAM is recommended).

1.4 Installation

1.4.1 Generic Python installation

The easiest way to install Aqua-Duct is to install Python 2.7 and use following command:

```
pip install aquaduct
```

If *pip* is not available try to install it by typing:

```
easy_install pip
```

Depending on the settings of your system you can prepend the above command with *sudo* or *doas* or do *user* installation:

```
# sudo
sudo pip install aquaduct

# doas
doas pip install aquaduct

# 'user' installation
pip install aquaduct --user
```

It is also good idea to try to install Aqua-Duct using virtualenv:

```
virtualenv aquaduct_installation
cd aquaduct_installation
. bin/activate
pip install aquaduct
```

Installation of PyMOL

Under most modern GNU/Linux distributions PyMOL is available as a package in repositories. For example if you are under Ubuntu/Debian you can install it by following command:

```
sudo apt-get install pymol
```

Under Windows there are several ways to install PyMOL, for more details see PyMOL web site.

Instructions for macOS and OpenBSD are in appropriate sections below.

1.4.2 GNU/Linux

Installation was tested on limited number of GNU/Linux systems. On the most of modern installations you can simply follow generic instructions, for example under Ubuntu 16.04 you can type:

```
sudo pip install aquaduct
```

NetCDF4 & MDAnalysis installation Ubuntu 14.04

Other systems may require additional work, in particular installation of NetCDF4 is sometimes cumbersome. Following is an example how to install all required packages under Ubuntu 14.04:

```
# install required python packages
sudo apt-get install python-dev python-pip python-numpy python-scipy python-
↪matplotlib python-scikits-learn

# install necessary libraries and git -  all required to compile netCDF4
sudo apt-get -y install libnetcdf-dev libhdf5-dev git

# clone netcdf4 python repository
git clone https://github.com/Unidata/netcdf4-python.git
# cd to cloned repository
cd netcdf4-python
# modify setup.cfg to add paths of hdf5 and netcdf4 libraries
sed -i '/\[directories\]/a \
HDF5_dir = /usr/lib \
HDF5_libdir = /usr/lib \
HDF5_incdir = /usr/include \
netCDF4_dir = /usr/lib \
netCDF4_libdir = /usr/lib \
netCDF4_incdir = /usr/include' setup.cfg
# run setup.py
sudo python setup.py install

# install MDAnalysis
sudo pip install "MDAnalysis[amber]>=0.15"
```

If everything went fine you can follow generic instructions.

SciPy update and Ubuntu/Debian

Debian (and Ubuntu) uses strange approach to Python installation. To install newer version of SciPy (if required) try following procedure:

```
# install libraries required for SciPy compilation
apt-get build-dep python-scipy

# install SciPy
easy_install-2.7 --upgrade scipy
```

Warning: The above procedure will remove current SciPy from *easy-install.pth* file.

1.4.3 MacOS

Aqua-Duct installation was tested on MacOS Sierra and is quite straightforward. It can be installed either with existing system Python or with custom Python installation. In both cases one have to install Xcode for the App Store.

System native Python

```
sudo easy_install pip
sudo pip install aquaduct
```

The drawback of using system Python installation is a lack of PyMOL. It should be, however, relatively easy to compile PyMOL on your own. Try to follow compilation instruction under BSD systems.

Custom Python

This is recommended way of Aqua-Duct installation. If you do not have custom Python installation you can get it by using one of package managers available for macOS, for example homebrew. With this package manager you can do following:

```
brew install python
sudo easy_install pip
sudo pip install aquaduct
```

Next, you can install PyMOL:

```
brew install pymol
brew cask install xquartz
```

Once XQuartz is installed you should reboot. The above procedure installs PyMOL, however, PyMOL Python modules are not visible. To fix it you can issue following commands:

```
cd /usr/local/lib/python2.7/site-packages
sudo ln -s /usr/local/Cellar/pymol/*/libexec/lib/python2.7/site-packages/* ./
```

The above instruction assumes that you are using *brew* and you have only one PyMOL installation.

1.4.4 Windows

Installation under Windows is also possible. The limiting factor is MDAnalysis which is not officially available under Windows yet. You can, however, install Cygwin and perform Aqua-Duct installation in Cygwin.

First, start with Cygwin installation. During the setup select following packages:

- python (2.7)
- python-devel (2.7)
- python-cython
- libnetcdf-devel
- libhdf5-devel
- liblapack-devel
- libopenblas
- python-numpy
- python-six

Another key component that have to be installed is C, C++ and Fortran compilers. You can simply install **gcc-g++** and **gcc-fortran** packages as a first choice, select following packages:

- gcc-g++
- gcc-fortran

Once Cygwin is installed with all required libraries you can perform following steps:

```
# install pip
easy_install-2.7 pip
```

First, try to install SciPy:

```
# install SciPy
pip install scipy
```

If you encounter any problems related to missing **xlocale.h** header file try the following workaround:

```
# prepare fake xlocale.h
ln -s /usr/include/locale.h xlocale.h
export CFLAGS="I"$( pwd )

# install SciPy
pip install scipy
```

Note: The above procedure for SciPy installation might not be optimal. For more information please got to SciPy web page.

Now, install **scikit-learn** and then Aqua-Duct:

```
# install scikit-learn
pip install scikit-learn

# finally, install aquaduct
pip install aquaduct
```

1.4.5 OpenBSD

Aqua-Duct can be also installed under OpenBSD (5.9 and 6.0 amd64). NetCDF-c version 4 has to be installed as OpenBSD ships only netCDF in version 3. First, install hdf5 library and GNU make:

```
# install hdf5 and GNU make
pkg_add hdf5 gmake
```

Next, download netCDF sources. Version 4.2.1.1 works out of the box but is a bit outdated. Visit NetCDF web page and select version of your choice. Older versions are available in the FTP archive. Once netCDF is downloaded and extracted go to the source directory and try following procedure:

```
# set LD and CPP flags
export LDFLAGS=-L/usr/local/lib
export CPPFLAGS=-I/usr/local/include

# configure project
./configure --enable-shared --enable-dap --disable-doxygen --enable-netcdf-4 --
↪prefix=/path/to/netCDF4/lib

# make and install
gmake
gmake install
```

You may now install py-scipy package:

```
pkg_add py-scipy
```

Install pip if it is missing:

```
pkg_add py-pip
```

Install netCDF4 Python:

```
# define netcdf-4 installation directory
export NETCDF4_DIR=/path/to/netCDF4/lib
pip2.7 install netCDF4
```

At this point you can follow generic Python instructions, type:

```
pip2.7 install aquaduct
```

PyMOL at OpenBSD

According to our knowledge it is possible to install PyMOL 1.4.1 and it is sufficient to work with Aqua-Duct. Go to SourceForge PyMOL download page and download, save, and extract sources.

PyMOL requires Python Mega Widgets. Download, for example Pmw 1.3.3b from SourceForge Pmw download page. Extract it and install by:

```
python2.7 setup.py install
```

TKinter (2.7) and several other packages are also required:

```
pkg_add python-tkinter freeglut glew png
```

Next, go to the extracted PyMOL sources open setup.py and modify inc_dirs variable at line 129 by adding following paths:

```
"/usr/X11R6/include/freetype2",
"/usr/X11R6/include",
"/usr/local/include",
```

Now, you can build and install PyMOL by typing following commands:

```
python2.7 setup.py build
python2.7 setup.py install
python2.7 setup2.py install
cp pymol /usr/local/bin
```

PyMOL can be run by typing *pymol* or can be used as Python module.

Other BSDs

Installation on other BSDs might be easier. For example, Python netCDF4 is available in ports of FreeBSD and DragonFlyBSD. Try to install it and SciPy, then proceed to generic Python installation instructions.

If you are using NetBSD or other BSD try to follow OpenBSD instructions.

CHAPTER

TWO

VALVE MANUAL

Valve application is a driver that uses `aquaduct` module to perform analysis of trajectories of selected residues in MD simulation.

2.1 *Valve* invocation

Once `aquaduct` module is installed (see *Aqua-Duct installation guide*) properly on the machine *Valve* is available as `valve.py` command line tool.

2.1.1 Usage

Basic help of *Valve* usage can be displayed by following command:

```
valve.py --help
```

It should display following information:

```
usage: valve.py [-h] [--debug] [--debug-file DEBUG_FILE]
                [--dump-template-config] [-t THREADS] [-c CONFIG_FILE] [--sps]
                [--max-frame MAX_FRAME] [--version] [--license]
Valve, Aquaduct driver
optional arguments:
  -h, --help            show this help message and exit
  --debug               Prints debug info. (default: False)
  --debug-file DEBUG_FILE
                        Debug log file. (default: None)
  --dump-template-config
                        Dumps template config file. Suppress all other output
                        or actions. (default: False)
  -t THREADS            Limit Aqua-Duct calculations to given number of
                        threads. (default: None)
  -c CONFIG_FILE        Config file filename. (default: None)
  --sps                 Use single precision to store data. (default: False)
  --max-frame MAX_FRAME
                        Limit number of frames. (default: None)
  --version             Prints versions and exits. (default: False)
  --license             Prints short license info and exits. (default: False)
```

2.1.2 Configuration file template

Configuration file used by *Valve* is of moderate length and complexity. It can be easily prepared with a template file that can be printed by *Valve*. Use following command to print configuration file template on the screen:

```
valve.py --dump-template-config
```

Configuration file template can also be easily saved in to a file with:

```
valve.py --dump-template-config > config.txt
```

Where config.txt is a configuration file template.

For detailed description of configuration file and available options see *Configuration file options*

2.1.3 *Valve* calculation run

Once configuration file is ready *Valve* calculations can be run with a following simple command:

```
valve.py -c config.txt
```

Some of *Valve* calculations can be run in parallel. By default all available CPU cores are used. This is not always desired - limitation of used CPU cores can be done with `-t` option which limits number of concurrent threads used by *Valve*. If it equals 1 no parallelism is used.

Note: Specifying number of threads greater then available CPU cores is generally not optimal.

However, in order to maximize usage of available CPU power it is recommended to set it as number of cores + 1. The reason is that *Valve* uses one thread for the main process and the excess over one for processes for parallel calculations. When parallel calculations are executed the main threads waits for results.

Note: Option `--max-frame` can be used for testing or debugging purposes. It allows to limit number of frames processed by *Valve*. If it set, for example, to `999` only first 1000 frames will be processed making all calculations very fast.

Single precision storage

Most of the calculation is *Valve* is performed by NumPy. By default, NumPy uses double precision floats. *Valve* does not change this behavior but has special option `--sps` which forces to store all data (both internal data stored in RAM and on the disk) in single precision. This spare a lot of RAM and is recommended what you perform calculation for long trajectories and you have limited amount of RAM.

Debuging

Valve can output some debug information. Use `--debug` to see all debug information on the screen or use `--debug-file` with some file name to dump all debug messages to the given file. Beside debug messages standard messages will be saved in the file as well.

2.2 How does *Valve* work

Application starts with parsing input options. If `--help` or `--dump-template-config` options are provided appropriate messages are printed on the screen and *Valve* quits with signal `0`.

Note: In current version *Valve* does not check the validity of the config file.

If config file is provided *Valve* parse it quickly and regular calculations starts according to its content. Calculations performed by *Valve* are done in several stages described in the next sections.

2.2.1 Traceable residues

In the first stage of calculation Valve finds all residues that should be traced and appends them to the list of *traceable residues*. It is done in a loop over all frames. In each frame residues of interest are searched and appended to the list but only if they are not already present on the list.

The search of the residues is done according to user provided definitions. Two requirements have to be met to append residue to the list:

1. The residue has to be found according to the *Object* definition.
2. The residue has to be within the *Scope* of interest.

The *Object* definition encompasses usually the active site of the protein. The *Scope* of interest defines, on the other hand, the boundaries in which residues are traced and is usually defined as protein.

Since `aquaduct` in its current version uses MDAnalysis Python module for reading, parsing and searching of MD trajectory data, definitions of *Object* and *Scope* have to be given as its *Selection Commands*.

Object definition

Object definition has to comprise of two elements:

1. It has to define residues to trace.
2. It has to define spatial boundaries of the *Object* site.

For example, proper Object definition could be following:

```
(resname WAT) and (sphzone 6.0 (resnum 99 or resnum 147))
```

It defines `WAT` as residues that should be traced and defines spatial constrains of the *Object* site as spherical zone within 6 Angstroms of the center of masses of residues with number 99 and 147.

Scope definition

Scope can be defined in two ways: as *Object* but with broader boundaries or as the convex hull of selected molecular object.

In the first case definition is very similar to *Object* and it has to follow the same limitations. For example, proper *Scope* definition could be following:

```
resname WAT around 2.0 protein
```

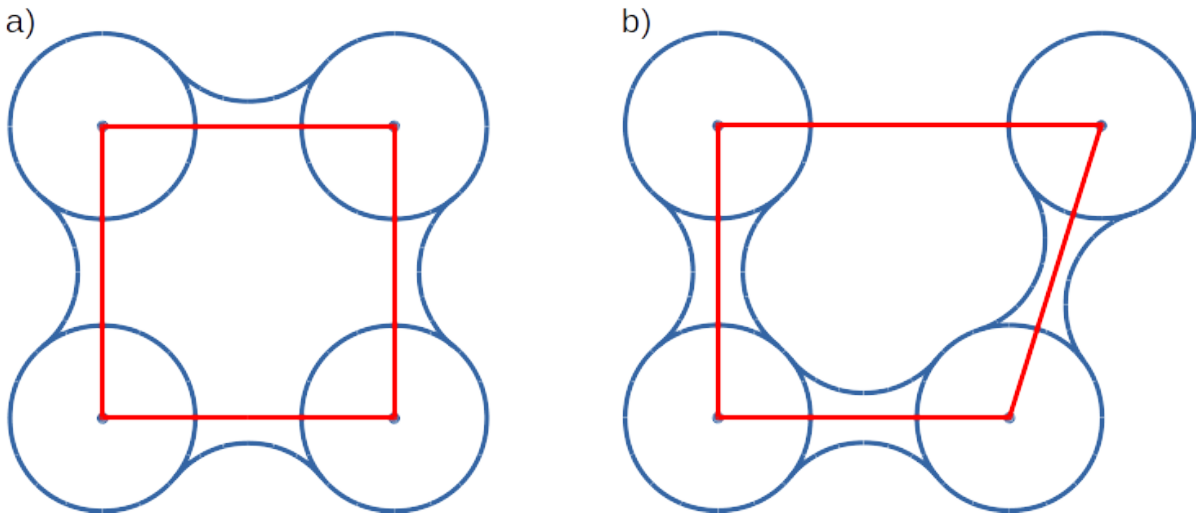
It consequently has to define `WAT` as residues of interest and defines spatial constrains: all `WAT` residues that are within 2 Angstroms of the protein.

If the *Scope* is defined as the convex hull of selected molecular object (which is recommended), the definition itself have to comprise of this molecular object only, for example `protein`. In that case the scope is interpreted as the interior of the convex hull of atoms from the definition. Therefore, *traceable residues* would be in the scope only if they are within the convex hull of atoms of `protein`.

Convex hulls of macromolecule atoms

AQ uses quickhull algorithm for convex hulls calculations (via SciPy class `scipy.spatial.ConvexHull`, see also http://www.qhull.org/ and original publication The quickhull algorithm for convex hulls).

Convex hull concept is used to check if traced molecules are inside of the macromolecule. Convex hull can be considered as rough approximation of molecular surface. Following picture shows schematic comparison of convex hull and solvent excluded surface:



Convex hull (red shape) of atoms (blue dots with VdW spheres) and SES (blue line): a) Convex hull and SES cover roughly the same area, Convex hull approximates SES; b) movement of one atom dramatically changes SES, however, interior of the molecule as approximated by Convex hull remains stable.

No doubts, Convex hull is a very rough approximation of SES. It has, however, one very important property when it is used to approximate interior of molecules: its interior does not considerably depend on the molecular conformation of a molecule (or molecular entity) in question.

2.2.2 Raw paths

The second stage of calculations uses the list of all traceable residues from the first stage and finds coordinates of center of masses for each residue in each frame. As in the first stage, it is done in a loop over all frames. For each residue in each frame *Valve* calculates or checks two things:

1. Is the residue in the *Scope* (this is always calculated according to the Scope definition).
2. Is the residue in the *Object*. This information is partially calculated in the first stage and can be reused in the second. However, it is also possible to recalculate this data according to the new *Object* definition.

For each of the *traceable residues* a special *Path* object is created. If the residue is in the *Scope* its center of mass is added to the appropriate *Path* object together with the information if it is in the *Object* or not.

2.2.3 Separate paths

The third stage uses collection of *Path* objects to create *Separate Path* objects. Each *Path* comprise data for one residue. It may happen that the residue enters and leaves the *Scope* and the *Object* many times over the entire MD. Each such an event is considered by *Valve* as a separate path.

Each *separate path* comprises of three parts:

1. *Incoming* - Defined as a path that leads from the point in which residue enters the *Scope* and enters the object for the firs time.
2. *Object* - Defined as a path that leads from the point in which residue enters the *Object* for the first time and leaves it for the last time.
3. *Outgoing* - Defined as a path that leads from the point in which residue leaves the *Object* for the last lime and leaves the *Scope*.

It is also possible that incoming and/or outgoing part of the separate path is empty.

Auto Barber

After the initial search of *Separate Path* objects it is possible to run procedure, Auto Barber, which trims paths down to the approximated surface of the macromolecule or other molecular entity defined by the user. This trimming is done by creating collection of spheres that have centers at the ends of paths and radii equal to the distance for the center to the nearest atom of user defined molecular entity. Next, parts of raw paths that are inside these spheres are removed and separate paths are recreated.

Auto Barber procedure has several options:

- **auto_barber** allows to define molecular entity which is used to calculate radii of spheres used for trimming raw paths.
- **auto_barber_mincut** allows to define minimal radius of spheres. Spheres of radius smaller then this value are not used in trimming.
- **auto_barber_maxcut** allows to define maximal radius of spheres. Spheres of radius greater then this value are not used in trimming.
- **auto_barber_tovdw** if set to *True* radii of spheres are corrected (decreased) by Van der Waals radius of the closest atom.

See also *options of separate_paths* stage.

Smoothing

Separate paths can be optionally smoothed. This can be done in two modes: *soft* and *hard*. In the former mode smoothed paths are used only for visualization purposes. In the latter, raw paths are replaced by smoothed.

Note: If *hard* mode is used all further calculations are performed for smoothed paths.

Available methods

Aqua-Duct implements several smoothing methods:

1. Savitzky-Golay filter - `SavgolSmooth` - see also original publication Smoothing and Differentiation of Data by Simplified Least Squares Procedures (doi:10.1021/ac60214a047).
2. Window smoothing - `WindowSmooth`
3. Distance Window smoothing - `DistanceWindowSmooth`
4. Active Window smoothing - `ActiveWindowSmooth`
5. Max Step smoothing - `MaxStepSmooth`
6. Window over Max Step smoothing - `WindowOverMaxStepSmooth`
7. Distance Window over Max Step smoothing - `DistanceWindowOverMaxStepSmooth`
8. Active Window over Max Step smoothing - `ActiveWindowOverMaxStepSmooth`

For detailed information on available configuration options see configuration file *smooth section* description.

2.2.4 Clusterization of inlets

Each of the separate paths has beginning and end. If they are at the boundaries of the *Scope* they are considered as *Inlets*, i.e. points that mark where the *traceable residues* enters or leaves the *Scope*. Clusters of inlets, on the other hand, mark endings of tunnels or ways in the system which was simulated in the MD.

Clusterization of inlets is performed in following steps:

1. *Initial clusterization*: All inlets are submitted to selected clusterization method and depending on the method and settings, some of the inlets might not be arranged to any cluster and are considered as outliers.
2. [Optional] *Outliers detection*: Arrangement of inlets to clusters is sometimes far from optimal. In this step, *inlets* that do not fit to cluster are detected and annotated as outliers. This step can be executed in two modes:
   (a) *Automatic mode*: Inlet is considered to be an outlier if its distance from the centroid is greater then mean distance + 4 * standard deviation of all distances within the cluster.
   (b) *Defined threshold*: Inlet is considered to be an outlier if its minimal distance from any other point in the cluster is greater then the threshold.
3. [Optional] *Reclusterization of outliers*: It may happen that the outliers form actually clusters but it was not recognized in initial clusterization. In this step clusterization is executed for outliers only and found clusters are appended to the clusters identified in the first step. Rest of the inlets are marked as outliers.

Potentially recursive clusterization

Both *Initial clusterization* and *Reclustarization* can be run in a recursive manner. If in the appropriate sections defining clusterization methods option *recursive_clusterization* is used appropriate method is run for each cluster separately. Clusters of specific size can be excluded from recursive clusterization (option *recursive_threshold*). It is also possible to limit maximal number of recursive levels - option *max_level*.

For additional information see *clusterization sections* options.

Available methods

Aqua-Duct implements several clustering methods with `sklearn.cluster` module:

1. `MeanShift` - default for *Initial clusterization*, see also original publication Mean shift: a robust approach toward feature space analysis (doi:10.1109/34.1000236).
2. `DBSCAN` - default for *Reclusterization of outliers*, see also original publication A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise
3. `AffinityPropagation` - see also original publication Clustering by Passing Messages Between Data Points (doi:10.1126/science.1136800)
4. `KMeans` - see also k-means++: The advantages of careful seeding, Arthur, David, and Sergei Vassilvitskii in Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms, Society for Industrial and Applied Mathematics (2007), pages 1027-1035.
5. `Birch` - see also Tian Zhang, Raghu Ramakrishnan, Maron Livny BIRCH: An efficient data clustering method for large databases and Roberto Perdisci JBirch - Java implementation of BIRCH clustering algorithm.

For additional information see *clusterization sections* options.

Master paths

At the end of clusterization stage it is possible to run procedure for *master path* generation. First, separate paths are grouped according to clusters. Paths that begin and end in particular clusters are grouped together. Next, for each group a *master path* (i.e., average path) is generated in following steps:

1. First, length of *master path* is determined. Lengths of each parts (incoming, object, outgoing) for each separate paths are normalized with bias towards longest paths. These normalized lengths are then used for as weights in averaging not normalized lengths. Values for all parts are summed and resulting value is the desired length of *master path*.
2. All separate paths are divided into chunks. Number of chunks is equal to the desired length of *master path* calculated in the previous step. Lengths of separate paths can be quite diverse, therefore, for different paths chunks are of different lengths.

3. For each chunk averaging procedure is run:
   (a) Coordinates for all separate paths for given chunk are collected.
   (b) Normalized lengths with bias toward longest paths for all separate paths for given chunk are collected.
   (c) New coordinates are calculated as weighted average of collected coordinates. As weights collected normalized lengths are used.
   (d) In addition width of chunk is calculated as a mean value of collected coordinates mutual distances.
   (e) Type of chunk is calculated as probability (frequency) of being in the *scope*.
4. Results for all chunks are collected, types probability are changed to types. All data is then used to create Master Path. If this fails no path is created.

More technical details on master path generation can be found in `aquaduct.geom.master.CTypeSpathsCollection.get_master_path()` method documentation.

2.2.5 Analysis

Fifth stage of *Valve* calculations analyses results calculated in stages 1 to 4. Results of the analysis is displayed on the screen or can be save to text file and comprise of following parts:

- Tile and data stamp.
- [Optional] Dump of configuration options.
- **Basic information on traceable residues and separate paths.**
  - Number of traceable residues.
  - Number of separate paths.
- **Basic information on inlets.**
  - Number of inlets.
  - Number of clusters.
  - Are outliers detected.
- **Summary of inlets clusters. Table with 5 columns:**
  1. **Nr**: Row number, starting from 0.
  2. **Cluster**: ID of the cluster. Outliers have 0.
  3. **Size**: Size of the cluster.
  4. **INCOMING**: Number of inlets corresponding to separate paths that enter the scope.
  5. **OUTGOING**: Number of inlets corresponding to separate paths that leave the scope.
- **Summary of separate paths clusters types. Table with 9 columns.**
  1. **Nr**: Row number, starting from 0.
  2. **CType**: Separate path Cluster Type.
  3. **Size**: Number of separate paths belonging to Cluster type.
  4. **Inp**: Average length of incoming part of the path. If no incoming part is available it is nan.
  5. **InpStd**: Standard deviation of length Inp.
  6. **Obj**: Average length of object part of the path. If no incoming part is available it is nan.
  7. **ObjStd**: Standard deviation of length Inp.
  8. **Out**: Average length of outgoing part of the path. If no incoming part is available it is nan.
  9. **OutStd**: Standard deviation of length Inp.

- **List of separate paths and their properties. Table with 17 columns.**
    1. **Nr**: - Row number, starting from 0.
    2. **ID**: - Separate path ID.
    3. **BeginF**: Number of frame in which the path begins.
    4. **InpF**: Number of frame in which path begins Incoming part.
    5. **ObjF**: Number of frame in which path begins Object part.
    6. **OutF**: Number of frame in which path begins Outgoing part.
    7. **EndF**: Number of frame in which the path ends.
    8. **InpL**: Length of Incoming part. If no incoming part nan is given.
    9. **ObjL**: Length of Object part.
    10. **OutL**: Length of Outgoing part. If no outgoing part nan is given.
    11. **InpS**: Average step of Incoming part. If no incoming part nan is given.
    12. **InpStdS**: Standard deviation of InpS.
    13. **ObjS**: Average step of Object part.
    14. **ObjStdS**: Standard deviation of ObjS.
    15. **OutS**: Average step of Outgoing part. If no outgoing part nan is given.
    16. **OutStdS**: Standard deviation of OutS.
    17. **CType**: Cluster type of separate path.

Separate path ID

Separate Paths IDs are composed of two numbers separated by colon. First number is the residue number. Second number is consecutive number of the separate path made by the residue. Numeration starts with 0.

Cluster Type of separate path

Each separate paths has two ends: beginning and end. Both of them either belong to one of the inlets clusters, or are among outliers, or are inside the scope. If an end belongs to one of the clusters (including outliers) it has ID of the cluster. If it is inside the scope it has special ID of `N`. Cluster type is an ID composed of IDs of both ends of separate path separated by colon charter.

2.2.6 Visualization

Sixth stage of *Valve* calculations visualizes results calculated in stages 1 to 4. Visualization is done with PyMOL. *Valve* automatically starts PyMOL and loads visualizations in to it. Molecule is loaded as PDB file. Other objects like Inlets clusters or paths are loaded as CGO objects.

Following is a list of objects created in PyMOL (all of them are optional). PyMOL object names given in **bold** text or short explanation is given.

- Selected frame of the simulated system. Object name: *molecule*.
- Inlets clusters, each cluster is a separate object. Object name: **cluster_** followed by cluster annotation: otliers are annotated as Out; regular clusters by ID.
- List of cluster types, raw paths. Each cluster type is a separate object. Object name composed of cluster type (colon replaced by underline) plus **_raw**.
- List of cluster types, smooth paths. Each cluster type is a separate object. Object name composed of cluster type (colon replaced by underline) plus **_smooth**.

- All raw paths. They can be displayed as one object or separated in to Incoming, Object and Outgoing part. Object name: **all_raw**, or **all_raw_in**, **all_raw_obj**, and **all_raw_out**.
- All raw paths inlets arrows. Object name: **all_raw_paths_io**.
- All smooth paths. They can be displayed as one object or separated in to Incoming, Object and Outgoing part. Object name: **all_smooth**, or **all_smooth_in**, **all_smooth_obj**, and **all_smooth_out**.
- All raw paths inlets arrows. Object name: **all_raw_paths_io**.
- Raw paths displayed as separate objects or as one object with several states. Object name: **raw_paths_** plus number of path or **raw_paths** if displayed as one object.
- Smooth paths displayed as separate objects or as one object with several states. Object name: **smooth_paths_** plus number of path or **smooth_paths** if displayed as one object.
- Raw paths arrows displayed as separate objects or as one object with several states. Object name: **raw_paths_io_** plus number of path or **raw_paths_io** if displayed as one object.
- Smooth paths arrows displayed as separate objects or as one object with several states. Object name: **smooth_paths_io_** plus number of path or **smooth_paths_io** if displayed as one object.

Color schemes

Inlets clusters are colored automatically. Outliers are gray.

Incoming parts of paths are red, Outgoing parts are blue. Object parts in case of smooth paths are green and in case of raw paths are green if residue is precisely in the object area or yellow if is leaved object area but it is not in the Outgoing part yet.

Arrows are colored in accordance to paths colors.

CHAPTER

THREE

CONFIGURATION FILE OPTIONS

Valve Configuration file is a simple and plain text file. It has similar structure as INI files commonly used in one of the popular operating systems and is compliant with Python module `ConfigParser`.

Configuration file comprises of several *sections*. They can be grouped in to three categories. Names of sections given in **bold** text.

1. **Global settings:**
    - **global**
2. **Stages options:**
    (a) **traceable_residues**
    (b) **raw_paths**
    (c) **separate_paths**
    (d) **inlets_clusterization**
    (e) **analysis**
    (f) **visualize**
3. **Methods options:**
    - **smooth**
    - **clusterization**
    - **reclusteriation**

3.1 Section global

This section allows settings of trajectory data and some other future global options.

Option	Default value	Description
top	None	Path to topology file. Aqua-Duct supports PDB, PRMTOP, PFS topology files.
trj	None	Path to trajectory file. Aqua-Duct supports NC and DCD trajectory files.

Note: Options **top** and **trj** are mandatory.

3.2 Common settings of stage sections

Stages 1-4 which perform calsulations have some common options allowig for execution control and saving/loading data.

Option	Default value	Description
execute	runonce	Option controls stage execution. It can have one of three possible values: `run`, `runonce`, and `skip`. If it is set to `run` calculations are always performed and if **dump** is set dump file is saved. If it is set to `runonce` calculations are performed if there is no dump file specified by **dump** option. If it is present calculations are skiped and data is loaded from the file. If it is set to `skip` calculations are skip and if **dump** is set data is loaded from the file.
dump	[dump file name]	File name of dump data. It is used to save results of calculations or to load previously calculated data - this depends on **execute** option. Default value of this option depends on the stage and for stages 1 to 4 is one of the following (listed in order): • 1_traceable_residues_data.dump • 2_raw_paths_data.dump • 3_separate_paths_data.dump • 4_inlets_clusterization_data.dump

Stages 5-6 also uses **execute** option, however, since they do not perform calculations *per se* in stead of **dump** option they use **save**.

Option	Default value	Description
execute	run	Option controls stage execution. It can have one of three possible values: `run`, `runonce`, and `skip`. If it is set to `run` or `runonce` stage is executed and results is saved according to **save** option. If it is set to `skip` stage is skipped.
save	[save file name]	File name for saving results. Default value of this option depends on the stage and for stages 1 to 4 is one of the following (listed in order): • 5_analysis_results.txt • 6_visualize_results.py Stage 6 can save results in two file types: 1. As Python script - extension `.py` plus companion archive `.tar.gz`, 2. As PyMOL session - extension `.pse`.

3.3 Stage traceable_residues

Option	Default value	Description
scope	None	Definition of *Scope* of interest. See also *Scope definition*.
scope_convexhull	True	Flag to set if the *Scope* is direct or convex hull definition.
object	None	Definition of *Object* of interest. See also *Object definition*.

Note: Options **scope** and **object** are mandatory.

3.4 Stage raw_paths

This stage also requires definition of the *Scope* and *Object*. If appropriate settings are not given, settings from the previous stage are used.

Option	Default value	Description
scope	None	Definition of *Scope* of interest. See also *Scope definition*. If `None` value form previous stage is used.
scope_convexhull	None	Flag to set if the *Scope* is direct or convex hull definition. If `None` value form previous stage is used.
object	None	Definition of *Object* of interest. See also *Object definition*. If `None`, value form the previous stage is used
clear_in_object_info	False	If it is set to `True` information on occupation of *Object* site by traceable residues calculated in the previous stage is cleared and have to be recalculated. This is useful if definition of *Object* was changed.

3.5 Stage separate_paths

Option	Default value	Description
discard_empty_paths	True	If set to `True` empty paths are discarded.
sort_by_id	True	If set to `True` separate paths are sorted by ID. Otherwise they are sorted in order of apparance.
apply_smoothing	False	If set to `True` smooth paths are precalculated according to **smooth** setting. This speed up access to smooth paths in later stages but makes dump data much bigger.
apply_soft_smoothing	True	If set to `True` raw paths are replaced by smooth paths calculated according to **smooth** section.
discard_short_paths	1	This option allows to discard paths that are shorter then the threshold.
auto_barber	None	This option allows to select molecular entity used in Auto Barber procedure. See also *Auto Barber* and `barber_with_spheres()`.
auto_barber_mincut	None	Minimal radius of spheres used in Auto Barber. If a sphere has radius smaller then this value it is not used to cut. This option can be switched off by setting it to *None*.
auto_barber_maxcut	2.8	Maximal radius of spheres used in Auto Barber. If a sphere has radius greater then this value it is not used to cut. This option can be switched off by setting it to *None*.
auto_barber_tovdw	True	Correct cutting sphere by decreasing its radius by VdW radius of the closest atom.

3.6 Stage inlets_clusterization

Option	Default value	Description
recluster_outliers	False	If set to `True` reclusterization of outliers is executed according to the method defined in **reclusterization** section.
detect_outliers	False	If set detection of outliers is executed. It could be set as a floating point distance threshold or set tu `Auto`. See *Clusterization of inlets* for more details.
singletons_outliers	False	Maximal size of cluster to be considered as outliers. If set to number > 0 clusters of that size are removed and their objects are moved to outliers. See *Clusterization of inlets* for more details.
max_level	5	Maximal number of recursive clusterization levels.
create_master_paths	False	If set to `True` master paths are created (fast CPU and big RAM recommended; 50k frames long simulation may need ca 20GB of memory)

3.7 Stage analysis

Option	Default value	Description
dump_config	True	If set to `True` configuration options, as seen by Valve, are added to the head of results.

3.8 Stage visualize

Option	Default value	Description
simply_smooths	RecursiveVector	Option indicates linear simplification method to be used in plotting smooth paths. Simplification removes points which do not (or almost do not) change the shape of smooth path. Possible choices are: • `RecursiveVector` (see *LinearizeRecursiveVector*), • `HobbitVector` (see *LinearizeHobbitVector*), • `OneWayVector` (see *LinearizeOneWayVector*), • `RecursiveTriangle` (see *LinearizeRecursiveTriangle*), • `HobbitTriangle` (see *LinearizeHobbitTriangle*), • `OneWayTriangle` (see *LinearizeOneWayTriangle*). Optionally name of the method can be followed by a threshold value in parentheses, ie `RecursiveVector(0.05)`. For sane values of thresholds see appropriate documentation of each method. Default values works well. This option is not case sensitive. It is recommended to use default method or `HobbitVector` method.
all_paths_raw	False	If True produces one object in PyMOL that holds all paths visualized by raw coordinates.
all_paths_smooth	False	If True produces one object in PyMOL that holds all paths visualized by smooth coordinates.
all_paths_split	False	If is set True objects produced by **all_paths_raw** and **all_paths_smooth** are split into Incoming, Object, and Outgoing parts and visualized as three different objects.
all_paths_raw_io	False	If set True arrows pointing beginning and end of paths are displayed oriented accordingly to raw paths orientation.
all_paths_smooth_io	False	If set True arrows pointing beginning and end of paths are displayed oriented accordingly to smooth paths orientation.
paths_raw	False	If set True raw paths are displayed as separate objects or as one object with states corresponding to number of path.
paths_smooth	False	If set True smooth paths are displayed as separate objects or as one object with states corresponding to number of path.
paths_raw_io	False	If set True arrows indicating beginning and and of paths, oriented accordingly to raw paths, are displayed as separate objects or as one object with states corresponding to number of paths.
paths_smooth_io	False	If set True arrows indicating beginning and and of paths, oriented accordingly to smooth paths, are displayed as separate objects or as one object with states corresponding to number of paths.
paths_states	False	If True objects displayed by **paths_raw**, **paths_smooth**, **paths_raw_io**, and **paths_smooth_io** are displayed as one object with with states corresponding to number of paths. Otherwise they are displayed as separate objects.
ctypes_raw	False	Displays raw paths in a similar manner as non split **all_paths_raw** but each cluster type is displayed in separate object.
ctypes_smooth	False	Displays smooth paths in a similar manner as non split **all_paths_smooth** but each cluster type is displayed in separate object.
show_molecule	False	If is set to selection of some molecular object in the system, for example to `protein`, this object is displayed.
show_molecule_frames	0	Allows to indicate which frames of object defined by **show_molecule** should be displayed. It is possible to set several frames. In that case frames would be displayed as states.
show_chull	False	If is set to selection of some molecular object in the system, for example to `protein`, convex hull of this object is displayed.
show_chull_frames	0	Allows to indicate for which frames of object defined by **show_chull** convex hull should be displayed. It is possible to

Note: Possibly due to limitations of `MDAnalysis` only whole molecules can be displayed. If **show_molecule** is set to `backbone` complete protein will be displayed any way. This may change in future version of `MDAnalysis` and or *aquaduct*.

Note: If several frames are selected they are displayed as states which may interfere with other PyMOL objects displayed with several states.

Note: If several states are displayed protein tertiary structure data might be lost. This seems to be limitation of either `MDAnalysis` or PyMOL.

3.9 Clusterization sections

Default section for definition of clusterization method is named **clusterization** and default section for reclusterization method definition is named **reclusterization**. All clusterization sections shares some common options. Other options depends on the method.

Option	Default value	Description
method	meanshift or dbscan	Name of clasteriation method. It have to be one of the following: dbscan, affprop, meanshift, birch, kmeans. Default value depends if it is **clusteriation** section (meanshift) or **reclusterization** section (dbscan).
recursive_clusterization	clusterization or None	If it is set to name of some section that holds clusterization method settings this method will be called in the next recursion of clusteriation. Default value for **reclusterization** is None.
recursive_threshold	None	Allows to set threshold of that excludes clusters of certain size from reclusterization. Value of this option comprises of *operator* and *value*. Operator can be one of the following: >, >=, <=, <. Value have to be expressed as floating number and it have to be in the range of 0 to 1.

3.9.1 dbscan

For detailed description look at `sklearn.cluster.DBSCAN` documentation. Following table summarized options available in *Valve* and is a copy of original documentation.

Option	Value type	Description
eps	float	The maximum distance between two samples for them to be considered as in the same neighborhood.
min_samples	int	The number of samples (or total weight) in a neighborhood for a point to be considered as a core point. This includes the point itself.
metric	str	The metric to use when calculating distance between instances in a feature array. Can be one of the following: • `cityblock`, • `cosine`, • `euclidean`, • `manhattan`.
algorithm	str	The algorithm to be used by the NearestNeighbors module to compute pointwise distances and find nearest neighbors. Can be one of the following: • `auto`, • `ball_tree`, • `kd_tree`, • `brute`.
leaf_size	int	Leaf size passed to BallTree or cKDTree.

3.9.2 affprop

For detailed description look at `AffinityPropagation` documentation. Following table summarized options available in *Valve* and is a copy of original documentation.

Option	Value type	Description
damping	float	Damping factor between 0.5 and 1.
convergence_iter	int	Number of iterations with no change in the number of estimated clusters that stops the convergence.
max_iter	int	Maximum number of iterations.
preference	float	Points with larger values of preferences are more likely to be chosen as exemplars.

3.9.3 meanshift

For detailed description look at `MeanShift` documentation. Following table summarized options available in *Valve* and is a copy of original documentation.

Option	Value type	Description
bandwidth	Auto or float	Bandwidth used in the RBF kernel. If `Auto` or `None` automatic method for bandwidth estimation is used. See `estimate_bandwidth()`.
cluster_all	bool	If true, then all points are clustered, even those orphans that are not within any kernel.
bin_seeding	bool	If true, initial kernel locations are not locations of all points, but rather the location of the discretized version of points, where points are binned onto a grid whose coarseness corresponds to the bandwidth.
min_bin_freq	int	To speed up the algorithm, accept only those bins with at least min_bin_freq points as seeds. If not defined, set to 1.

3.9.4 birch

For detailed description look at `Birch` documentation. Following table summarized options available in *Valve* and is a copy of original documentation.

Option	Value type	Description
threshold	float	The radius of the subcluster obtained by merging a new sample and the closest subcluster should be lesser than the threshold. Otherwise a new subcluster is started.
branching_factor	int	Maximum number of CF subclusters in each node.
n_clusters	int	Number of clusters after the final clustering step, which treats the subclusters from the leaves as new samples. By default, this final clustering step is not performed and the subclusters are returned as they are.

3.9.5 kmeans

For detailed description look at `KMeans` documentation. Following table summarized options available in *Valve* and is a copy of original documentation.

Option	Value type	Description
n_clusters	int	The number of clusters to form as well as the number of centroids to generate.
max_iter	int	Maximum number of iterations of the k-means algorithm for a single run.
n_init	int	Number of time the k-means algorithm will be run with different centroid seeds. The final results will be the best output of n_init consecutive runs in terms of inertia.
init	str	Method for initialization, defaults to `k-means++`. Can be one of following: `k-means++` or `random`.
tol	float	Relative tolerance with regards to inertia to declare convergence.

3.10 Smooth section

Section **smooth** supports following options:

Option	Value type	Description
method	str	Smoothing method. Can be one of the following: • `window`, (see *WindowSmooth*) • `mss`, (see *MaxStepSmooth*) • `window_mss`, (see *WindowOverMaxStepSmooth*) • `awin`, (see *ActiveWindowSmooth*) • `awin_mss`, (see *ActiveWindowOverMaxStepSmooth*) • `dwin`, (see *DistanceWindowSmooth*) • `dwin_mss`, (see *DistanceWindowOverMaxStepSmooth*) • `savgol`. (see *SavgolSmooth*)
recursive	int	Number of recursive runs of smoothing method.
window	int or float	In window based method defines window size. In plain `window` it has to be int number. In `savgol` it has to be odd integer.
step	int	In step based method defines size of the step.
function	str	In window based methods defines averaging function. Can be `mean` or `median`.
polyorder	int	In `savgol` is polynomial order.

CHAPTER

FOUR

VALVE TUTORIAL

This tutorial assumes `aquaduct` and *Valve* is already installed - see *Aqua-Duct installation guide*. It is also assumed that user is acquainted with *Valve manual* and *Valve Configuration file options*.

4.1 *Valve* invocation

Usually *Valve* is run by:

```
valve.py
```

To check if *Valve* is installed and works properly try to issue following commands:

```
valve.py --help
valve.py --version
```

4.2 Test data

Mouse!

We will use 1ns MD simulation data of sEH protein (PDBID **1cqz**). This simulation was performed in Amber 14. Necessary files can be found at Aqua-Duct home page in section download. Required data is in the *sample data* file.

4.3 Inspect your system

Before we start any calculations lets have a look at the protein of interest. Start *PyMOL* and get `1cqz` PDB structure (for example by typing in *PyMOL* command prompt `fetch 1cqz`).

To setup *Valve* calculations we need to know the active site of the protein. More precisely we need to know IDs of residues that are in the active site. This would allow us to create *Object definition*.

But wait. Is it really the correct structure? How many chains there are? What is the numeration of residues? How does it compare with the topology file from *sample data*?

4.3.1 Create *Object definition*

Lets load another structure. Open file `1cqz_sample_topology.pdb` (see *Test data*). It is a first frame of the MD simulation and it is en example of how the frame of MD looks like. In order to create *Object definition* you have to discover following things:

1. What is the name of water molecules?
2. What are numbers of residues in the active site?

3. What size the active site is of?

Note: It is also a good idea to open `.pdb` file in your favorite text editor and look at residue numbers and names.

4.3.2 Create *Scope definition*

Scope definition is easy to create. We will use *Convex hull* version so the scope definition could be simply `backbone`.

4.4 Prepare config file

Valve performs calculations according to the configuration (aka *config*) file.

Lets start from dumping config file template to `config.txt` file. Open it in your favorite editor and fill all options. If you have troubles look at *Configuration file options* (and *Valve manual*).

Things to remember:

1. Provide correct paths to topology and trajectory data.
2. Enter correct *Object* and *Scope* definitions.
3. Make sure visualization is switched on.

4.5 Run *Valve*

Make sure all necessary data is in place. Open terminal, go to your working directory and type in:

```
valve.py -c config.txt
```

Depending on your machine and current load it may take a while (matter of minutes) to complete all calculations.

4.5.1 Visual inspection

In the last stage *PyMOL* should pop up and *Valve* should start to feed it with visualization data. This would take a moment and if you set up `save` option a *PyMOL* session would be saved. Once it is done *Valve* quits and switches off *PyMOL*. Now, you can restart it and read saved session.

4.5.2 Clusterization

Improve clusterization of Inlets. See *Configuration file options* for more hints on available clusterization options.

4.5.3 Analysis tables

Open `5_analysys_results.txt` file and look at summaries and tables. See also *Valve manual*.

4.6 Feedback

Give us your opinion. Send your questions, inquires, anything to developer(s): mailto:info@aquaduct.pl. This are couple of questions that might be useful to form your opinion.

1. What do you like in *Valve* and *Aqua-Duct*?
2. What do you do not like in *Valve* or *Aqua-Duct*?
3. What is missing?
4. Do you find it useful?

CHAPTER

FIVE

AQUADUCT

5.1 aquaduct package

5.1.1 Subpackages

aquaduct.apps package

Submodules

aquaduct.apps.valvecore module

version_nice()

class **ValveConfig**

Bases: object, aquaduct.apps.valvecore.ConfigSpecialNames

__init__()

static **common_config_names**()

static **common_traj_data_config_names**()

static **global_name**()

static **cluster_name**()

static **recluster_name**()

static **recursive_clusterization_name**()

static **recursive_threshold_name**()

static **smooth_name**()

stage_names(*nr=None*)

get_common_traj_data(*stage*)

get_global_options()

get_stage_options(*stage*)

get_cluster_options(*section_name=None*)

get_recluster_options()

get_smooth_options()

get_default_config()

load_config(*filename*)

save_config_stream(*fs*)

save_config(*filename*)

dump_config()

_ValveConfig__make_options_nt(*input_options*)

valve_begin()

valve_end()

valve_load_config(*filename*, *config*)

valve_read_trajectory(*top*, *traj*)

valve_exec_stage(*stage*, *config*, *stage_run*, *reader=None*, *no_io=False*, *run_status=None*, ***kwargs*)

stage_I_run(*config*, *options*, *reader=None*, *max_frame=None*, ***kwargs*)

stage_II_run(*config*, *options*, *reader=None*, *all_res=None*, *res_ids_in_object_over_frames=None*, *max_frame=None*, ***kwargs*)

stage_III_run(*config*, *options*, *paths=None*, *reader=None*, ***kwargs*)

stage_IV_run(*config*, *options*, *spaths=None*, *center_of_system=None*, ***kwargs*)

stage_V_run(*config*, *options*, *spaths=None*, *paths=None*, *inls=None*, *ctypes=None*, ***kwargs*)

stage_VI_run(*config*, *options*, *reader=None*, *spaths=None*, *inls=None*, *ctypes=None*, *master_paths=None*, *master_paths_smooth=None*, ***kwargs*)

aquaduct_version_nice()
Returns `aquaduct` version number as nicely formatted string.

Returns string composed on the basis of the number returned by `version()`.

Return type str

Module contents

aquaduct.geom package

Submodules

aquaduct.geom.cluster module

This module provides functions for clusterization. Clusterization is done by `scikit-learn` module.

MeanShiftBandwidth(*X*, ***kwargs*)

class **PerformClustering**(*method*, ***kwargs*)
Bases: `object`

__init__(*method*, ***kwargs*)

__call__(*coords*)

_get_noclusters(*n*)

fit(*coords*)

centers()

aquaduct.geom.convexhull module

_vertices_ids(*convexhull*)

_vertices_points(*convexhull*)

_point_within_convexhull(*convexhull*, *point*)

_facets (*convexhull*)

_edges (**args*, ***kwargs*)

is_point_within_convexhull (*point_chull*)

aquaduct.geom.master module

part2type_dict = {0: 's', 1: 'c', 2: 's'}
Part number to *GenericPathTypeCodes* dictionary.

parts = (0, 1, 2)
Parts enumerate.

class CTypeSpathsCollectionWorker (*spaths=None*, *ctype=None*, *bias_long=5*, *smooth=None*)
Bases: object

Worker class for averaging spaths in points of master path.

__init__ (*spaths=None*, *ctype=None*, *bias_long=5*, *smooth=None*)
Core method for averaging spaths in to master path.

Averaging is done in chunks.

Parameters

- **spaths** (*list*) – List of separate paths to average.
- **ctype** (InletClusterGenericType) – CType of spaths.
- **bias_long** (*int*) – Bias towards long paths used in `lens_norm()`.
- **smooth** (Smooth) – Smoothing method.

coords_types_prob_widths (*sp_slices_*)
Calculates average coordinates, type and width in given chunk.

Parameter `sp_slices_` is tuple of length equal to number of spaths. It contains slices for all spaths respectively. With these slices spaths are cut and **only** resulting chunks are used for calculations.

Therefore, this method average spaths in one point of master math. This point is defined by slices submitted as `sp_lices_` parameter.

Algorithm of averaging (within current chunks of spaths):

1. Coordinates for all spaths are collected.
2. Lengths of all spaths are collected (from cached variables) and kept as lists of lengths equal to chunks' sizes.

   Note: Lengths of collected lengths of spaths are of the same size as coordinates

3. New coordinates are calculated as weighted average of collected coordintates with `numpy.average()`. As weights collected lengths are used.

   Note: Function `numpy.average()` is called with flatten coordinates and lengths.

4. Width of average path is calculated as mean value of flatten coordinates mutual distances.
5. Type of average paths is calculated as probability (frequency) of *scope_name*.

Parameters **sp_slices** (*tuple*) – Slices that cut chunks from all paths.

Return type 3 element tuple

Returns coordinates, type (frequency), and width of averaged spaths in current point

__call__(*nr_sp_slices_*)
Callable interface.

Parameters **nr_sp_slices** (*tuple*) – Two element tuple: nr and sp_slice

class CTypeSpathsCollection(*spaths=None*, *ctype=None*, *bias_long=5*, *pbar=None*, *threads=1*)
Bases: object

Object for grouping separate paths that belong to the same CType. Method *get_master_path()* allows for calculation of average path.

parts = (0, 1, 2)
Enumeration of spath parts.

__init__(*spaths=None*, *ctype=None*, *bias_long=5*, *pbar=None*, *threads=1*)

Parameters

- **spaths** (*list*) – List of separate paths.
- **ctype** (InletClusterGenericType) – CType of spaths.
- **bias_long** (*int*) – Bias towards long paths used in *lens_norm()*.
- **pbar** – Progress bar object.
- **threads** (*int*) – Number of available threads.

beat()
Touch progress bar, if any.

update()
Update progres bar by one, if any.

lens()
Returns total lengths of all paths.

If ctype in #:# and not 0 and not None then take length of *object* part only.

Returns Total (or *object* part) lengths of all paths.

Return type numpy.ndarray

lens_norm()
Returns normalized lengths calculated by *lens()*.

Applied normalization is twofold:

1. All lengths are divided by maximal length, and
2. All lengths are subjected to pow() function with p = bias_long.

Returns Normalized total (or *object* part) lengths of all paths.

Return type numpy.ndarray

lens_real()
Returns real lengths of all paths.

Returns Sizes of all paths.

Return type list

full_size()
Returns desired size of master path.

Returns Size of master path.

Return type int

static simple_types_distribution(*types*)
Calculates normalized sizes of incoming, object, and outgoing parts of spath using generic types.

It is assumed that spath has object part.

Parameters **types** (*list*) – List of generic types.

Return type 3 element list

Returns Normalized sizes of incomin, object, and outgoing parts.

types_distribution()

Return type numpy.matrix

Returns median values of *simple_types_distribution()* for all spaths.

types_prob_to_types(*types_prob*)
Changes types probabilities as returned by *CTypeSpathsCollectionWorker.coords_types_prob_widths()* to types.

Parameters **types_prob** (*list*) – List of types probabilities.

Return type list

Returns List of *GenericPathTypeCodes*.

get_master_path(*smooth=None*, *resid=0*)
Averages spaths into one master path.

This is done in steps:

1. Master path is an average of bunch of spaths. Its length is determined by *full_size()* method.
2. All spaths are then divided in to chunks according to *xzip_xzip()* function with `N` set to lenght of master path. This results in list of length equal to the length of master path. Elements of this lists are slice objects that can be used to slice spaths in appropriate chunks.
3. Next, for each element of this list *CTypeSpathsCollectionWorker.coords_types_prob_widths()* method is called. Types probabilities are changed to types wiht *types_prob_to_types()*.
4. Finally, all data are used to create appropriate `MasterPath`. If this fails *None* is returned.

Parameters

- **smooth** (`Smooth`) – Smoothing method.
- **resid** (*int*) – Residue ID of master path.

Return type *MasterPath*

Returns Average path as *MasterPath* object or *None* if creation of master path failed.

aquaduct.geom.pca module

class Center(*X*)
Bases: `object`

__init__(*X*)

__call__(*X*)

undo(*X*)

class Normalize(*X*)
Bases: `object`

__init__(*X*)

__call__(*X*)

undo(*X*)

class **Standartize**(*X*)
Bases: *aquaduct.geom.pca.Center*, *aquaduct.geom.pca.Normalize*

__init__(*X*)

__call__(*X*)

undo(*X*)

class **PCA**(*X*, *prepro=None*)
Bases: object

__init__(*X*, *prepro=None*)

P

preprocess(*X*)

preprocess_undo(*X*)

__call__(*X*)

undo(*T*)

aquaduct.geom.smooth module

Smooth module defines methods for smoothing of trajectories.

Available methods:

SavgolSmooth	Savitzky-Golay based smoothing.
WindowSmooth	Defined size window smoothing.
DistanceWindowSmooth	Distance defined size window smoothing.
ActiveWindowSmooth	Active size window smoothing.
MaxStepSmooth	Maximal step smoothing.
WindowOverMaxStepSmooth	Window smoothing over maximal step smoothing.
DistanceWindowOverMaxStepSmooth	Distance window smoothing over maximal step smoothing.
ActiveWindowOverMaxStepSmooth	Active window smoothing over maximal step smoothing.

class **Smooth**(*recursive=None*, ***kwargs*)
Bases: object

Base class for all smoothing methods.

__init__(*recursive=None*, ***kwargs*)

Parameters **recursive** (*int*) – Number of recursions of the method, everything evaluated to `False` is equivalent to 1.

smooth(*coords*)
Abstract method for smoothing method implementation.

Parameters **coords** (*Iterable*) – Input coordinates to be smoothed.

__call__(*coords*)
Call method for all smoothing methods.

Input coordinates should be iterable and each element should be numpy.ndarray. If length of `coords` is less then 3 smoothing method is not run and coordinates are returned unchanged.

If `recursive` is set smoothing method is applied appropriate number of times.

Parameters **coords** (*Iterable*) – Input coordinates to be smoothed.

Return type numpy.ndarray

Returns Smoothed coordinates.

class **GeneralWindow** (*function=<function mean>*, ***kwargs*)
Bases: object

Base class for window based smoothing methods.

__init__ (*function=<function mean>*, ***kwargs*)

Parameters **function** (*function*) – Function to be used for averaging coordinates within a window.

static **max_window_at_pos** (*pos*, *size*)
Method returns maximal possible window at given position of the list with given size of the list. Returned window fits in to the list of given size and is symmetrical.

Parameters

- **pos** (*int*) – Position in question.
- **size** (*int*) – Length of the list.

Return type 2 element tuple of int

Returns Lowest possible bound and highest possible bound of the window.

check_bounds_at_max_window_at_pos (*lb*, *ub*, *pos*, *size*)
Method checks if window fits in to maximal possible window calculated according to *max_window_at_pos()*. If not window is corrected.

Parameters

- **lb** (*int*) – Lower bound of the window in question.
- **ub** (*int*) – Upper bound of the window in question.
- **pos** (*int*) – Position in question.
- **size** (*int*) – Length of the list.

Return type 2 element tuple of int

Returns Lowest possible bound and highest possible bound of the window corrected to maximal possible window.

class **IntWindow** (*window=5*, ***kwargs*)
Bases: *aquaduct.geom.smooth.GeneralWindow*

Base class for all window smoothing methods that require integer window.

__init__ (*window=5*, ***kwargs*)

Parameters **window** (*int*) – One side size of the window.

class **FloatWindow** (*window=5.0*, ***kwargs*)
Bases: *aquaduct.geom.smooth.GeneralWindow*

Base class for all window smoothing methods that require float window.

__init__ (*window=5.0*, ***kwargs*)

Parameters **window** (*float*) – Size of the window.

class **WindowSmooth** (***kwargs*)
Bases: *aquaduct.geom.smooth.Smooth*, *aquaduct.geom.smooth.IntWindow*

Defined size window smoothing.

For each coordinate a symmetrical (if possible) window of size defined by `window` is created. In case of coordinates at the edges created window is truncated to the edges. Next, all coordinates within the window are averaged with a function defined by `function`. Resulting value(s) are the smoothed coordinates.

__init__(***kwargs*)

smooth(**args*, ***kwargs*)

Parameters **coords** (*Iterable*) – Input coordinates to be smoothed.

class DistanceWindowSmooth(***kwargs*)

Bases: *aquaduct.geom.smooth.Smooth*, *aquaduct.geom.smooth.FloatWindow*

Distance defined size window smoothing.

This is modification of *WindowSmooth* method. The difference is in the definition of the window size. Here, it is an average distance between points of input coordinates. Thus, before smoothing average distance between all points is calculated and this value is used to calculate actual window size.

Next, for each coordinate a symmetrical (if possible) window of size calculated in the first step is created. In case of coordinates at the edges created window is truncated to the edges. Next, all coordinates within the window are averaged with a function defined by `function`. Resulting value(s) are the smoothed coordinates.

__init__(***kwargs*)

smooth(**args*, ***kwargs*)

Parameters **coords** (*Iterable*) – Input coordinates to be smoothed.

class ActiveWindowSmooth(***kwargs*)

Bases: *aquaduct.geom.smooth.Smooth*, *aquaduct.geom.smooth.FloatWindow*

Active size window smoothing.

Similarly to *DistanceWindowSmooth* method the window size is defined as a distance. The difference is that the actual window size is calculated for each point separately. Thus, for each coordinate the window is calculated by examining the distance differences between points. In this method window is not necessarily symmetrical. Once window is calculated all coordinates within the window are averaged with a function defined by `function`. Resulting value(s) are the smoothed coordinates.

__init__(***kwargs*)

smooth(**args*, ***kwargs*)

Parameters **coords** (*Iterable*) – Input coordinates to be smoothed.

class MaxStepSmooth(*step=1.0*, ***kwargs*)

Bases: *aquaduct.geom.smooth.Smooth*

Maximal step smoothing.

This method moves thorough coordinates and calculates distance over the traversed path. If it is then `step` the coordinate is used as a "cardinal point". The beginning and the end of the path are also added to the list of cardinal points. Next, all cardinal points and points of linear interpolation between cardinal points are returned as smoothed coordinates. Number of interpolated points is in accordance to points skipped between cardinal points.

__init__(*step=1.0*, ***kwargs*)

smooth(**args*, ***kwargs*)

Parameters **coords** (*Iterable*) – Input coordinates to be smoothed.

class SavgolSmooth(*window_length=5*, *polyorder=2*, ***kwargs*)

Bases: *aquaduct.geom.smooth.Smooth*

Savitzky-Golay based smoothing.

Method uses 1D filter available in SciPy, see `savgol_filter()`. For each dimension filter is applied separately. Only `window_length` and `polyorder` attributes are used.

__init__(*window_length=5*, *polyorder=2*, ***kwargs*)

Param int window_length: Size of the window, odd number.

Param int polyorder: Polynomial order.

smooth(**args*, ***kwargs*)

Parameters **coords** (*Iterable*) – Input coordinates to be smoothed.

class **WindowOverMaxStepSmooth**(***kwargs*)
Bases: *aquaduct.geom.smooth.Smooth*

Window smoothing over maximal step smoothing.

First, *MaxStepSmooth* is applied, and then *WindowSmooth*.

__init__(***kwargs*)

smooth(*coords*)

Parameters **coords** (*Iterable*) – Input coordinates to be smoothed.

class **ActiveWindowOverMaxStepSmooth**(***kwargs*)
Bases: *aquaduct.geom.smooth.Smooth*

Active window smoothing over maximal step smoothing.

First, *MaxStepSmooth* is applied, and then *ActiveWindowSmooth*.

__init__(***kwargs*)

smooth(*coords*)

Parameters **coords** (*Iterable*) – Input coordinates to be smoothed.

class **DistanceWindowOverMaxStepSmooth**(***kwargs*)
Bases: *aquaduct.geom.smooth.Smooth*

Distance window smoothing over maximal step smoothing.

First, *MaxStepSmooth* is applied, and then *DistanceWindowSmooth*.

__init__(***kwargs*)

smooth(*coords*)

Parameters **coords** (*Iterable*) – Input coordinates to be smoothed.

aquaduct.geom.traces module

diff(*trace*)
This function calculates the distance between 2 given points.

Parameters **trace** – coordinates in numpy array object

Returns distance between points

tracepoints(*start*, *stop*, *nr*)

Parameters

- **start** – coordinates of the first point as a numpy array object
- **stop** – coordinates of the second point as a numpy array object
- **nr** – number of elements between the first and second point

Returns two-dimentional numpy array; number of dimentions depends on nr parameter

midpoints(*paths*)

The function returns a tuple of numpy arrays extended with mid point spanning last and first element(column) of these arrays.

Parameters **paths** – a tuple of 2-dimentional np.arrays that hold 3D coordinates; each element holds one trace, all elements are supposed to make one path divided in to sections

Returns paths elements with additional mid points as a generator object

length_step_std(*trace*)
This function calculates sum, mean and standard deviation from all segments of a trace.

Parameters **trace** – coordinates of points as numpy array

Returns a tuple with basics statistics of a trace

derrivative(*values*)

vector_norm(*V*)

Parameters **V** – a vector in a form of array-like object, tuple or a list

Returns normalized length of a vector

triangle_angles(*A*, *B*, *C*)
Parameters are coordinates of points which are tops of triangle. The function calculates angles in a triangle formed by given coordinates.

Parameters

- **A** – coordinates of the first point
- **B** – coordinates of the second point
- **C** – coordinates of the third point

Returns list of arguments where angle is given in radians , the output is as follow: [BAC,CAB,ABC]

triangle_angles_last(*A*, *B*, *C*)
Parameters are coordinates of points which are tops of triangle. The function calculates the [ABC] angle.

Parameters

- **A** – coordinates of the first point [A top]
- **B** – coordinates of the second point [B top]
- **C** – coordinates of the third point [C top]

Returns list with one value of ABC angle in radians

triangle_height(*A*, *B*, *C*)
Parameters are coordinates of points which are tops of triangle. The function calculates the ABC triangle height.

Parameters

- **A** – coordinates of the first point [A top]
- **B** – coordinates of the second point [B top]
- **C** – coordinates of the third point [C top]

Returns one value of ABC triangle height

vectors_angle(*A*, *B*)
This function calculates the angle between two given vectors (starting from the [0,0,0] to the given coordinates.

Parameters

- **A** – coordinates of the first point which is the end of the vector
- **B** – coordinates of the second point which is the end of the vector

Returns the angle between vectors in question (in radians)

vectors_angle_alt(*A*, *B*)

This function calculates the angle between two given vectors (starting from the [0,0,0] to the given coordinates

- alternative method.

Parameters

- **A** – coordinates of the first point which is the end of the vector
- **B** – coordinates of the second point which is the end of the vector

Returns the angle between vectors in question (in radians)

vectors_angle_alt_anorm(*A*, *B*, *A_norm*)

This function calculates the angle between two given vectors (starting from the [0,0,0] to the given coordinates

- alternative method with additional A_norm holding norm of A.

Parameters

- **A** – coordinates of the first point which is the end of the vector
- **B** – coordinates of the second point which is the end of the vector
- **A_norm** – additional parameter holding normalized of vector A

Returns the angle between vectors in question (in radians)

vectors_angle_anorm(*A*, *B*, *A_norm*)

This function calculates the angle between two given vectors (starting from the [0,0,0] to the given coordinates using additional A_norm holding norm of A.

Parameters

- **A** – coordinates of the first point which is the end of the vector
- **B** – coordinates of the second point which is the end of the vector
- **A_norm** – additional parameter holding normalized of vector A

Returns the angle between vectors in question (in radians)

class LinearizeOneWay

Bases: object

here(*coords*)

This function simplifies the trace by removing the redundant, linear points :param coords: 3D coordinates of a trace as an array-like object :return: indices of coordinates which are a staring and ending points of linear fragments and other non-linear points of the trace

class LinearizeHobbit

Bases: *aquaduct.geom.traces.LinearizeOneWay*

and_back_again(*coords*)

__call__(*coords*)

class LinearizeRecursive

Bases: object

Base class for linearization methods classes.

It implements recursive algorithm.

here(*coords*, *depth=0*)

Core of recursive linearization argorithm.

It checks if the first, the last and the middle point are linear according to the criterion. The middle point is a selected point that is in the middle of length of the paths made by input coordinates.

If these points are linear their indices are returned. Otherwise, coordinates are split into two parts. First part spans points from the first point to the middle point (inclusive) and the second part spans points from the middle (inclusive) to the last point. Next, these two parts are submitted recursively to *here()*.

Results of these recursive calls are joined, redundant indices are removed and sorted result is returned.

Parameters

- **coords** (*numpy.ndarray*) – Input coordinates.
- **depth** (*int*) – Depth of recurence.

Returns Indices of `coords` points that can be used instead of all points in visulatization.

Return type list of int

__call__(*coords*)

class TriangleLinearize(*threshold=0.01*)

Bases: `object`

__init__(*threshold=0.01*)

is_linear(*coords*, ***kwargs*)

class VectorLinearize(*treshold=0.05236*)

Bases: `object`

Base class for linearization methods classes.

It implements vector linearization criterion.

__init__(*treshold=0.05236*)

is_linear_core(*coords*, *depth=None*)

Method checks if input coordinates are linear according to the threshold and depth.

It begins with calculation of the threshold. If *depth* is None it is set to 1. Current threshold is calculated with following simple equation:

$$threshold_{current} = threshold_{initial} * (2 - 0.9^{depth})$$

Next, in a loop over all points but the first and the last the angle is calculated between two vectors. The first one made by the point and the first point, and the second vector made by the last and the first point. If any of the calculated angles is bigger the the treshold methods returns False; otherwise method returns True.

Parameters

- **coords** (*numpy.ndarray*) – Coordinates for which linearization criterion is checked.
- **depth** (*int*) – Depth of recurence.

Returns True if input coordinates are linear and False otherwise.

Return type bool

is_linear(*coords*, *depth=None*, ***kwargs*)

For more detail see *is_linear_core()* which is used as the criterion of linearity in this method.

Parameters

- **coords** (*numpy.ndarray*) – Coordinates for which linearization criterion is checked.

- **depth** (*int*) – Depth of recurence.

Returns True if input coordinates are linear and False otherwise. Criterion is checked for coordinates in normal and reverse order.

Return type bool

class LinearizeRecursiveVector(*treshold=0.05236*)

Bases: *aquaduct.geom.traces.LinearizeRecursive*, *aquaduct.geom.traces.VectorLinearize* Class provides recursive linearization of coordinates with *LinearizeRecursive* algorithm and the criterion of linearity implemented by *VectorLinearize*. This is default method.

class LinearizeRecursiveTriangle(*threshold=0.01*)

Bases: *aquaduct.geom.traces.LinearizeRecursive*, *aquaduct.geom.traces.TriangleLinearize*

Class provides recursive linearization of coordinates with *LinearizeRecursive* algorithm and the criterion of linearity implemented by *TriangleLinearize*.

class LinearizeHobbitVector(*treshold=0.05236*)

Bases: *aquaduct.geom.traces.LinearizeHobbit*, *aquaduct.geom.traces.VectorLinearize*

Class provides recursive linearization of coordinates with *LinearizeHobbit* algorithm and the criterion of linearity implemented by *VectorLinearize*.

class LinearizeHobbitTriangle(*threshold=0.01*)

Bases: *aquaduct.geom.traces.LinearizeHobbit*, *aquaduct.geom.traces.TriangleLinearize*

Class provides recursive linearization of coordinates with *LinearizeHobbit* algorithm and the criterion of linearity implemented by *TriangleLinearize*.

class LinearizeOneWayVector(*treshold=0.05236*)

Bases: *aquaduct.geom.traces.LinearizeOneWay*, *aquaduct.geom.traces.VectorLinearize*

Class provides recursive linearization of coordinates with *LinearizeOneWay* algorithm and the criterion of linearity implemented by *VectorLinearize*.

class LinearizeOneWayTriangle(*threshold=0.01*)

Bases: *aquaduct.geom.traces.LinearizeOneWay*, *aquaduct.geom.traces.TriangleLinearize*

Class provides recursive linearization of coordinates with *LinearizeOneWay* algorithm and the criterion of linearity implemented by *TriangleLinearize*.

Module contents

aquaduct.traj package

Submodules

aquaduct.traj.dumps module

class TmpDumpWriterOfMDA

Bases: object

__init__()

dump_frames(*reader*, *frames*, *selection='protein'*)

close()

__del__()

aquaduct.traj.inlets module

class ProtoInletTypeCodes

surface = 'surface'

internal = 'internal'

incoming = 'inin'

outgoing = 'inout'

class InletTypeCodes
Bases: *aquaduct.traj.inlets.ProtoInletTypeCodes*

all_surface = [('surface', 'inin'), ('surface', 'inout')]

all_internal = [('internal', 'inin'), ('internal', 'inout')]

all_incoming = [('surface', 'inin'), ('internal', 'inin')]

all_outgoing = [('surface', 'inout'), ('internal', 'inout')]

surface_incoming = ('surface', 'inin')

internal_incoming = ('internal', 'inin')

internal_outgoing = ('internal', 'inout')

surface_outgoing = ('surface', 'inout')

itype = 'internal'

class InletClusterGenericType(*inp*, *out*)
Bases: object

__init__(*inp*, *out*)

input

output

static **cluster2str**(*cl*)

__getitem__(*item*)

__len__()

__str__()

__repr__()

make_val(*base*)

__cmp__(*other*)

__hash__()

make_spherical(*xyz*)

class InletClusterExtendedType(*surfin*, *interin*, *interout*, *surfout*)
Bases: *aquaduct.traj.inlets.InletClusterGenericType*

__init__(*surfin*, *interin*, *interout*, *surfout*)

generic

class Inlet(*coords*, *type*, *reference*)
Bases: tuple

__getnewargs__()
Return self as a plain tuple. Used by copy and pickle.

__getstate__()
Exclude the OrderedDict from pickling

static __new__(*_cls*, *coords*, *type*, *reference*)
Create new instance of Inlet(coords, type, reference)

__repr__()
Return a nicely formatted representation string

__slots__ = ()

_asdict()
Return a new OrderedDict which maps field names to their values

_fields = ('coords', 'type', 'reference')

classmethod _make(*iterable*, *new=<built-in method __new__ of type object>*, *len=<built-in function len>*)
Make a new Inlet object from a sequence or iterable

_replace(*_self*, ***kwds*)
Return a new Inlet object replacing specified fields with new values

coords
Alias for field number 0

reference
Alias for field number 2

type
Alias for field number 1

class Inlets(*spaths, center_of_system=None, onlytype=[('surface', 'inin'), ('surface', 'inout')]*)
Bases: object

__init__(*spaths, center_of_system=None, onlytype=[('surface', 'inin'), ('surface', 'inout')]*)

extend_inlets(*spath*, *onlytype=None*)

add_cluster_annotations(*clusters*)

size

coords

types

refs

call_clusterization_method(*method*, *data*)

perform_clustering(*method*)

perform_reclustering(*method*, *skip_outliers=False*, *skip_size=None*)

recluster_cluster(*method*, *cluster*)

recluster_outliers(*method*)

small_clusters_to_outliers(*maxsize*)

renumber_clusters()

sort_clusters()

clusters_list

clusters_centers

clusters_size

clusters_std

spaths2ctypes (**args*, ***kwargs*)

lim_to (*what*, *towhat*)

lim2spaths (*spaths*)

lim2types (*types*)

lim2clusters (*clusters*)

limspaths2 (**args*, ***kwargs*)

aquaduct.traj.paths module

union_full (*a*, *b*)

union_smartr (*a*, *b*)

union (*a*, *b*, *smartr=True*)

glue (*a*, *b*)

xor_full (**args*, ***kwargs*)

xor_smartr (**args*, ***kwargs*)

xor (*a*, *b*, *smartr=True*)

left (*a*, *b*, *smartr=True*)

right (*a*, *b*, *smartr=True*)

class **SmartRangeFunction** (*element*, *times*)

Bases: `object`

__init__ (*element*, *times*)

__repr__ ()

get ()

rev ()

isin (*element*)

class **SmartRangeEqual** (*element*, *times*)

Bases: *aquaduct.traj.paths.SmartRangeFunction*

get ()

rev ()

isin (*element*)

class **SmartRangeIncrement** (*element*, *times*)

Bases: *aquaduct.traj.paths.SmartRangeFunction*

get ()

rev ()

isin (*element*)

class **SmartRangeDecrement** (*element*, *times*)

Bases: *aquaduct.traj.paths.SmartRangeFunction*

get ()

rev ()

isin (*element*)

class **SmartRange**(*iterable=None*)

Bases: object

__init__(*iterable=None*)

last_element()

last_times()

raw

append(*element*)

get()

rev()

__len__()

min()

max()

isin(*element*)

class **PathTypesCodes**

path_in_code = 'i'

path_object_code = 'c'

path_out_code = 'o'

class **GenericPathTypeCodes**

object_name = 'c'

scope_name = 's'

out_name = 'n'

class **GenericPaths**(*id_of_res*, *min_pf=None*, *max_pf=None*)

Bases: object, *aquaduct.traj.paths.GenericPathTypeCodes*

__init__(*id_of_res*, *min_pf=None*, *max_pf=None*)

types

frames

max_frame

min_frame

add_coord(*coord*)

add_object(*frame*)

add_scope(*frame*)

add_type(*frame*, *ftype*)

_gpo()

_gpi()

get_paths_in()

get_paths_out()

find_paths(*fullonly=False*, *smartr=True*)

find_paths_coords_types(*fullonly=False*)

get_single_path_coords_types(*spath*)

barber_with_spheres(*spheres*)

class SinglePathID(*path_id=None*, *nr=None*)
Bases: object

__init__(*path_id=None*, *nr=None*)

__str__()

yield_single_paths(*gps*, *fullonly=False*, *progress=False*)

class SinglePath(*path_id*, *paths*, *coords*, *types*)
Bases: object, *aquaduct.traj.paths.PathTypesCodes*, *aquaduct.traj.inlets.InletTypeCodes*

empty_coords = array([], shape=(0, 3), dtype=float64)

__init__(*path_id*, *paths*, *coords*, *types*)

path_in

path_object

path_out

types_in

types_object

types_out

coords_first_in

coords_last_out

coords_filo

get_inlets()

coords

coords_cont

paths

paths_cont

types

types_cont

gtypes

gtypes_cont

etypes

etypes_cont

size

begins

ends

has_in

has_object

has_out

get_coords(**args*, ***kwargs*)

get_coords_cont(*smooth=None*)

_make_smooth_coords(**args*, ***kwargs*)

apply_smoothing(*smooth*)

get_distance_cont(*smooth=None*, *normalize=False*)

get_distance_rev_cont(**args*, ***kwargs*)

get_distance_both_cont(**args*, ***kwargs*)

get_velocity_cont(**args*, ***kwargs*)

get_acceleration_cont(**args*, ***kwargs*)

_SinglePath__paths

class **MasterPath**(*sp*)
Bases: *aquaduct.traj.paths.SinglePath*

__init__(*sp*)

add_width(*width*)

aquaduct.traj.reader module

class **Reader**(*topology*, *trajectory*)
Bases: object

__init__(*topology*, *trajectory*)

open_trajectory()

number_of_frames

set_current_frame(*frame*)

next_frame()

iterate_over_frames()

parse_selection(*selection*)

select_resnum(*resnum*)

select_multiple_resnum(*resnum_list*)

class **ReadViaMDA**(*topology*, *trajectory*)
Bases: *aquaduct.traj.reader.Reader*

number_of_frames

set_current_frame(*frame*)

next_frame()

parse_selection(*selection*)

select_resnum(*resnum*)

select_multiple_resnum(*resnum_list*)

__enter__()

__exit__(*typ*, *value*, *traceback*)

open_trajectory()

class **ReadAmberNetCDFviaMDA**(*topology*, *trajectory*)
Bases: *aquaduct.traj.reader.ReadViaMDA*

open_trajectory()

class ReadDCDviaMDA(*topology*, *trajectory*)
Bases: *aquaduct.traj.reader.ReadViaMDA*

open_trajectory()

VdW_radii = {'ge': 2.11, 'gd': 2.34, 'ga': 1.87, 'la': 2.43, 'li': 1.82, 'tl': 1.96, 'lu': 2.24, 'lr': 2.46, 'th': 2.45, 'ti': 2.11, 'te':
Dictionary of VdW radii.

Data taken from L. M. Mentel, mendeleev, 2014. Available at: https://bitbucket.org/lukaszmentel/mendeleev. Package **mendeleev** is not used because it depends on too many other libraries.

atom2vdw_radius(*atom*)
Function tries to guess atom element and checks if it is in *VdW_radii* dictionary. If it fails 1.4 is returned. Guessing is done twice:

1. Function `MDAnalysis.topology.core.guess_atom_element()` is used.
2. `MDAnalysis.core.AtomGroup.Atom.element` is used.

Parameters **atom** (*MDAnalysis.core.AtomGroup.Atom*) – Atom of interest.

Return type float

Returns VdW radius.

aquaduct.traj.selections module

class Selection
Bases: `object`

def __init__(self,selection,selection_string=None):

self.selection_object = selection self.selection_string = selection_string

center_of_mass()

iterate_over_residues()

unique_resids()

unique_resids_number()

atom_positions()

center_of_mass_of_residues()

get_convexhull_of_atom_positions()

contains_residues(*other*, *convex_hull=False*, *map_fun=None*)

containing_residues(*other*, **args*, ***kwargs*)

uniquify()

__add__(*other*)

first_resid()

class SelectionMDA(*atoms*)
Bases: `MDAnalysis.core.AtomGroup.AtomGroup`, *aquaduct.traj.selections.Selection*

iterate_over_residues()

unique_resids(*ikwid=False*)

atom_positions()

__add__(*other*)

uniquify()

class **CompactSelectionMDA**(*sMDA*)

Bases: object

__init__(*sMDA*)

toSelectionMDA(*reader*)

Module contents

aquaduct.utils package

Submodules

aquaduct.utils.clui module

Module comprises convieniences functions and definitios for different operations related to command line user interface.

emit_message_to_file_in_root_logger(*mess*)

message_special(*mess*)

message(*mess*, *cont=False*)

Prints message to standard error. If FileHandler is present in the root_logger the same message is appended to the log file.

Parameters

- **mess** (*str*) – message to print
- **cont** (*bool*) – if set True no new line is printed

class **fbm**(*info*, *cont=True*)

Bases: object

__init__(*info*, *cont=True*)

__enter__()

__exit__(*typ*, *value*, *traceback*)

__call__(*info*)

class **tictoc**(*mess*)

Bases: object

__init__(*mess*)

__enter__()

__exit__(*typ*, *value*, *traceback*)

gregorian_year_in_days = 365.2425

Length of Gregorian year in days. Average value. Source: https://en.wikipedia.org/wiki/Year

smart_time_string(*s*, *rl=0*, *t=1.1*, *maximal_length=None*, *maximal_units=5*)

Function transforms time in seconds to nicely formatted string of length defined by maximal_length. Depending on number of seconds time is represented with one or more of the following units:

Unit name	Unit abbreviation
seconds	s
minutes	m
hours	h
days	d
years	y

Maximal number of units used in time string can be set with maximal_units.

Parameters

- **s** (*int*) – Input time in seconds.
- **rl** (*int*) – Number of units already used for representing time.
- **t** (*float*) – Exces above standard number of current time units.
- **maximal_length** (*int*) – Maximal length of the output string. Must be greater then 0.
- **maximal_units** (*int*) – Maximal number of units used in the output string. Must be greater then 0 and lower then 6.

Returns string of nicely formated time

Return type str

gsep(*sep='-'*, *times=72*, *length=None*)
Generic separator.

Parameters

- **sep** (*str*) – Element(s) of separator.
- **times** (*int*) – Number of times `sep` is printed.
- **length** (*int*) – Optional maximal length of output.

Returns String separator.

Return type str

tsep(*line*)

Parameters **line** (*str*) – Input line.

Returns Returns default *gsep()* of length of `line`.

underline(*line*)

Parameters **line** (*str*) – Input line.

Returns String made by concatenation of `line`, `os.linesep`, and output of *tsep()* called with `line`.

Return type str

thead(*line*)

Parameters **line** (*str*) – Input line.

Returns String made by concatenation of output of *tsep()* called with `line`, `line`, `os.linesep`, and again output of *tsep()* called with `line`.

Return type str

class SimpleProgressBar(*maxval=None*, *mess=None*)
Bases: `object`

Simple progress bar displaying progress with percent indicator, progress bar and ETA. Progress is measured by iterations.

Variables

- ***rotate*** (*str*) – String comprising characters with frames of a rotating toy.
- ***barlenght*** (*int*) – Length of progress bar.
- **maxval** (*int*) – maximal number of iterations
- **current** (*int*) – current number of iterations
- **overrun_notice** (*bool*) – if True, overrun above `maxval` iterations causes insert of newline

- **overrun** (*bool*) – flag of overrun
- **begin** (*int*) – time in seconds at the initialization of the *SimpleProgressBar* class.
- **tcurrent** (*int*) – time in seconds of current iteration

rotate = '\\|/-'

barlenght = 24

__init__(*maxval=None*, *mess=None*)

Parameters

- **maxval** (*int*) – Maximal number of iterations stored to `maxval`.
- **mess** (*str*) – Optional message displayed at progress bar initialization.

bar()

ETA()

Returns ETA calculated on the basis of current number of iterations `current` and current time `tcurrent`. If number of iterations is 0 returns `?`. Time is formated wiht *smart_time_string()*.

Returns ETA as string.

Return type str

percent()

Returns float number of precent progress calculated in the basis of current number of iterations `current`. Should return number between 0 and 100.

Returns percent progress number

Return type float

show()

Shows current progress.

If value returned by *percent()* is =< 100 then progres is printed as percent indicator leaded by ETA calculated by *ETA()*.

If value returned by *percent()* is > 100 then progress is printed as number of iterations and total time.

Progress bar is writen to standard error.

heartbeat()

update(*step*)

Updates number of current iterations `current` by one if `step` is > 0. Otherwise number of current iterations is not updated. In boths cases time of current iteration `tcurrent` is updated and *show()* is called.

Parameters **step** (*int*) – update step

ttime()

Calculates and returns total time string formated with *smart_time_string()*.

Returns string of total time

Return type str

finish()

Finishes progress bar. First, *update()* is called with `step` = 0. Next message of total time is writen to standard error.

pbar

alias of *SimpleProgressBar*

get_str_timestamp()

aquaduct.utils.helpers module

Collection of helpers - functions and decorators.

combine(*seqin*)

This is an alien function. It is not extensively used.

Directly taken form http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/302478/index_txt

Returns a list of all combinations of argument sequences. For example, following call:

```
combine(((1,2),(3,4)))
```

gives following list of combinations:

```
[[1, 3], [1, 4], [2, 3], [2, 4]]
```

Parameters **seqin** (*tuple*) – Tuple of sequences to combine.

Returns All possible combinations of all input sequences.

Return type list of lists

is_number(*s*)

lind(*l*, *ind*)

Indexes lists using lists of integers as identificators. For example:

```
lind(['a','b','c','d','e'],[1,4,2])
```

returns:

```
['b', 'e', 'c']
```

Parameters

- **l** (*list*) – List to be indexed.
- **ind** (*list*) – Integer indexes.

Returns Reindexed list.

Return type list

class Auto

Auto type definition. The class is used as an alternative value for options (if particular option supports it). If options (or variables/parameters etc.) have value of *Auto* it means that an automatic process for parametrization should be performed.

For example, if the input parameter is set to *Auto* it is supposed that its value is calculated on the basis of input data or other parameters.

__repr__()

Returns String `Auto`.

Return type str

__str__()

Calls *__repr__()*.

create_tmpfile(*ext=None*)

Creates temporary file. File is created, closed and its file name is returned.

Note: It is responsibility of the caller to delete the file.

Parameters **ext** (*str*) – Optional extension of the file.

Returns File name of created temporary file.

Return type str

range2int(*r*, *uniq=True*)

Transforms a string range in to a list of integers (with added missing elements from given ranges).

For example, a following string:

```
'0:2 4:5 7 9'
```

is transformed into:

```
[0,1,2,4,5,7,9]
```

Parameters

- **r** (*str*) – String of input range.
- **uniq** (*bool*) – Optional parameter, if set to *True* only unique and sorted integers are returned.

Returns List of integers.

Return type list of int

int2range(*l*)

Transforms a list of integers in to a string of ranges.

For example, a following list:

```
[0,1,2,4,5,7,9]
```

is transformed into:

```
0:2 4:5 7 9
```

Parameters **l** (*list*) – input list of int

Returns String of ranges.

Return type str

is_iterable(*l*)

Checks if provided object is iterable. Returns True is it is iterable, otherwise returns False.

Parameters **l** (*list*) – input object

Returns True if submitted object is iterable otherwise returns False.

Return type bool

Warning: Current implementation cannot be used with generators!

Todo

Current implementation is primitive and HAVE TO be replaced.

sortify(*gen*)
Decorator to convert functions' outputs into a sorted list. If the output is iterable it is converted in to a list of appropriate length. If the output is not iterable it is converted in to a list of length 1.

Written on the basis of *listify()*.

Returns Output of decorated function converted to a sorted list.

Return type list

uniqify(*gen*)
Decorator to convert functions' outputs into a sorted list of unique objects. If the output is iterable it is converted in to a list of appropriate length. If the output is not iterable it is converted in to a list of length 1.

Written on the basis of *listify()*.

Returns Output of decorated function converted to a sorted list of unique objects.

Return type list

listify(*gen*)
Decorator to convert functions' outputs into a list. If the output is iterable it is converted in to a list of appropriate length. If the output is not iterable it is converted in to a list of length 1.

This function was copied from:

http://argandgahandapandpa.wordpress.com/2009/03/29/python-generator-to-list-decorator/

and further improved by tljm@wp.pl.

Returns Output of decorated function converted to a list.

Return type list

tupleify(*gen*)
Decorator to convert functions' outputs into a tuple. If the output is iterable it is converted in to a tuple of apropriate length. If the output is not iterable it is converted in to a tuple of length 1.

Written on the basis of *listify()*.

Returns Output of decorated function converted to a tuple.

Return type tuple

arrayify(*gen*)
Decorator to convert functions' outputs into a 2D numpy array. If the output is iterable it is converted in to a 2D numpy array of appropriate shape. If the output is not iterable it is converted in to a 2D numpy array of shape 1x1.

Written on the basis of *listify()*.

Returns Output of decorated function converted to a 2D numpy array.

Return type numpy.ndarray

arrayify1(*gen*)
Decorator to convert functions' outputs into a 1D numpy array. If the output is iterable it is converted in to a 2D numpy array of appropriate shape. If the output is not iterable it is converted in to a 2D numpy array of shape 1x1.

Written on the basis of *listify()*.

Returns Output of decorated function converted to a 1D numpy array.

Return type numpy.ndarray

list_blocks_to_slices(*l*)
Slices list in to block according to its elements identity. Resulting slices correspond to blocks of identical elements.

Parameters **l** (*list*) – List of any objects.

Returns Generator of slices.

Return type generator

split_list(*l*, *s*)

what2what(*what*, *towhat*)
This function search if elements of the one list (:attr: 'what') are present in the other list (:attr: 'towhat') and returns indices of elements form :attr:'what' list as a tuple. If elements from the first list are not present in the second list the tuple is empty. :param list what: Input list for which indices of elements present in `towhat` are returned. :param list towhat: List of elements which input list is indexed to. :return: Indices of `what` list that are present in `towhat` list. :rtype: tuple

make_iterable(*something*)
If input object is not iterable returns it as one element list. Otherwise returns the object.

Parameters **something** (*object*) – Input object.

Returns Iterable object.

Return type iterable or list

strech_zip(**args*)

compress_zip(**args*)

zip_zip(**args*, ***kwargs*)

xzip_xzip(**args*, ***kwargs*)

concatenate(**args*)
Concatenates input iterable arguments in to one generator.

class Bunch(***kwds*)
Bases: object

http://code.activestate.com/recipes/52308 foo=Bunch(a=1,b=2)

__init__(***kwds*)

aquaduct.utils.maths module

class NumpyDefaultsStorageTypes
Bases: object

float_default
alias of float64

int_default
alias of int64

make_default_array(*array_like*)

aquaduct.utils.multip module

class CpuThreadsCount
Bases: object

cpu_count = 2

threads_count = None

Module contents

aquaduct.visual package

Submodules

aquaduct.visual.cmaps module

aquaduct.visual.helpers module

euclidean(*A*, *B*)

cityblock(*A*, *B*)

cc_safe(*c*)

cc(*c*)

color_codes(*code*, *custom_codes=None*)

get_cmap(*size*)

class **ColorMapDistMap**

Bases: object

grey = (0.5, 0.5, 0.5, 1)

__init__()

distance(*E1*, *E2*)

static **color_distance**(*e1*, *e2*)

__call__(*node*)

_ColorMapDistMap__do_cadex()

f_like(*n*)

aquaduct.visual.pymol_cgo module

aquaduct.visual.pymol_connector module

class **BasicPymolCGO**

Bases: object

cgo_entity_begin = []

cgo_entity_end = []

__init__()

clean()

new()

get()

static **make_color_triple**(*color_definition*)

class **BasicPymolCGOLines**

Bases: *aquaduct.visual.pymol_connector.BasicPymolCGO*

cgo_entity_begin = [2.0, 1.0]

cgo_entity_end = [3.0]

add(*coords=None*, *color=None*)

class BasicPymolCGOSpheres
Bases: *aquaduct.visual.pymol_connector.BasicPymolCGO*

cgo_entity_begin = []

cgo_entity_end = []

add(*coords=None*, *radius=None*, *color=None*)

class BasicPymolCGOPointers
Bases: *aquaduct.visual.pymol_connector.BasicPymolCGO*

cgo_entity_begin = []

cgo_entity_end = []

add_cone(*coords1=None*, *coords2=None*, *radius1=None*, *radius2=None*, *color1=None*, *color2=None*)

add_pointer(*point=None*, *direction=None*, *length=None*, *color=None*, *reverse=False*)

class SimpleTarWriteHelper
Bases: object

__init__()

open(*filename*)

save_object2tar(*obj*, *name*)

save_file2tar(*filename*, *name*)

__del__()

class ConnectToPymol
Bases: object

cgo_line_width = 2.0

ct_pymol = 'pymol'

ct_file = 'file'

__init__()

decode_color(**args*, ***kwargs*)

init_pymol()

init_script(*filename*)

add_cgo_object(*name*, *cgo_object*, *state=None*)

del_cgo_object(*name*, *state=None*)

load_pdb(*name*, *filename*, *state=None*)

orient_on(*name*)

__del__()

class SinglePathPlotter(*pymol_connector*, *linearize=None*)
Bases: object

__init__(*pymol_connector*, *linearize=None*)

add_single_path_continous_trace(*spath*, *smooth=None*, *plot_in=True*, *plot_object=True*, *plot_out=True*, ***kwargs*)

paths_trace(*spaths*, *smooth=None*, *name='paths'*, *state=None*, ***kwargs*)

paths_inlets(*spaths*, *smooth=None*, *color=None*, *plot_in=True*, *plot_out=True*, *name='in-outlet'*, *state=None*, ***kwargs*)

scatter(*coords*, *radius=0.4*, *color='r'*, *name='scatter'*, *state=None*)

convexhull(*chull*, *color='m'*, *name='convexhull'*, *state=None*)

aquaduct.visual.quickplot module

yield_spath_len_and_smooth_diff_in_types_slices(*sp*, *smooth=None*, *smooth_len=None*, *smooth_diff=None*, *types='etypes'*)

plot_colorful_lines(*x*, *y*, *c*, ***kwargs*)

spaths_spectra(*spaths*, ***kwargs*)

plot_spath_spectrum(*sp*, ***kwargs*)

spath_spectrum(*sp*, ***kwargs*)

showit(*gen*)

get_ax3d(*fig*, *sub=111*)

class SimpleTracePlotter

Bases: object

plot_line(*coords*, *color*, ***kwargs*)

single_trace(*coords*, *color='r'*, ***kwargs*)

path_trace(*path*, *color=('r', 'g', 'b')*, *plot_in=True*, *plot_object=True*, *plot_out=True*, ***kwargs*)

class SimpleProteinPlotter

Bases: *aquaduct.visual.quickplot.SimpleTracePlotter*

protein_trace(*protein*, *smooth=None*, *color=('c', 'm', 'y')*, ***kwargs*)

class SimplePathPlotter

Bases: *aquaduct.visual.quickplot.SimpleTracePlotter*

single_path_traces(*spaths*, *smooth=None*, *color=('r', 'g', 'b')*, ***kwargs*)

class MPLTracePlotter

Bases: *aquaduct.visual.quickplot.SimplePathPlotter*, *aquaduct.visual.quickplot.SimpleProteinPlotter*

init_ax(**args*, ***kwargs*)

plot_line(**args*, ***kwargs*)

scatter(**args*, ***kwargs*)

Module contents

5.1.2 Module contents

Aqua-Duct - a collection of tools to trace residues in MD simulation.

version()

Returns *aquaduct* version number.

Returns 3 element tuple of int numbers

Return type tuple

version_nice()

Returns *aquaduct* version number as nicely formatted string.

Returns string composed on the basis of the number returned by *version()*.

Return type str

greetings()

Returns fancy greetings of *aquaduct*. It has a form of ASCII-like graphic. Currently it returns following string:

```
------------------------------------------------
          ~ ~ ~ A Q U A - D U C T ~ ~ ~
################################################
####         ########         ########         ####
##             ####             ####             ##
#                ##               ##               #
#                ##               ##               #
#                ##               ##               #
#                ##               ##               #
------------------------------------------------
```

Returns *aquaduct* fancy greetings.

Return type str

CHAPTER

SIX

AQUA-DUCT CHANGELOG

- **0.3.1**
  - AutoBarber tovdw option.
  - AutoBarber minimal and maximal cut options.
  - Fixed bug in AutoBarber: some areas were sometimes not cut.
  - Documentation improvements.
  - Valve driver simplified. Most of the functionality moved to separate module.
  - Option for single precision storage.
  - Added Savitzky-Golay smoothing; AQ requires SciPy >= 0.14 now.
  - Improved sorting of CTypes.
  - Raw and Separate paths uses SmartRanges. This allowed for excellent performance improvement of Separate paths calculation.
  - Default display of molecule changed to silver cartoon.
  - Object shape displayed in orange.
  - Fixed several small bugs.
- **0.2.26**
  - Stage execution time debug messages.
  - Total execution time debug message.
- **0.2.25**
  - initial public release

Documentation for other versions of Aqua-Duct:

- development version (use with care)

PYTHON MODULE INDEX

a

INDEX

Symbols

A

B

C

D

E

F

G

H

I

L

M

T

U

V

W

X

Y

Z