avtraj Documentation

Release 18.10.1

Thomas-Otavio Peulen

Oct 11, 2019

Contents

AvTraj is tool to calculate accessible volumes of labels attached to biomolecules. using MD trajectories as an input qvTraj can generate, write, and analyze accessible volumes (AVs) with only a few lines of Python code. By the use of LabelLib AvTraj provides programmatic access to latest developments in implicit dye models for FRET experiments.

AvTraj is a python library that allows users to perform simulations of accessible volumes for molecular dynamics (MD) trajectories. AvTraj serves as a high-level interface for the development of new methodologies for structure-based fluorescence spectroscopy.

Features include:

> A wide support of diverse MD formats by the use of MDTraj. Extremely fast calculation of AVs by the use of LabelLib. Extensive analysis functions including those that compute inter-dye distances, FRET-efficiencies, fluorescence decays, distance distributions, and an Pythonic API.

AVTraj includes a command-line application, avana, for screening and analyzing structural models.

CHAPTER 1

Usage

1.1 Installation

There are different ways of installing avtraj. Avtraj can either be installed via pip, conda, or manually. We recommend that you install `avtraj` with `pip`.

```
$ pip install avtraj
```

You can install also `avtraj` with `conda`, if you prefer.

```
$ conda install -c conda-forge avtraj
```

Conda is a cross-platform package manager built especially for scientific python. It will install `avtraj` along with all dependencies from a pre-compiled binary. If you don't have Python or the `conda` package manager, we recommend starting with the Anaconda Scientific Python distribution, which comes pre-packaged with many of the core scientific python packages that MDTraj uses (see below), or with the Miniconda Python distribution, which is a bare-bones Python installation.

AVTraj supports Python 2.7 or Python 3.4+ (recommended) on Mac, Linux, and Windows.

1.2 Tutorial

Define all parameters necessary for the calculation of an accessible volume (AV)

```
{
    "version": "0.0.1",
    "Distances": {
        "344-496": {
            "Forster_radius": 52.0,
            "distance": 54.8,
            "distance_type": "RDAMean",
            "error_neg": 6.3,
```

(continues on next page)

(continued from previous page)

```
            "error_pos": 5.5,
            "position1_name": "344A",
            "position2_name": "496D",
            "distance_samples": 100000
        }
    },
    "Positions": {
        "344A": {
         "chain_identifier": "A",
         "residue_seq_number": 344,
         "atom_name": "CB",
         "linker_length": 20.0,
         "linker_width": 1.5,
         "radius1": 3.5,
         "strip_mask": "MDTraj: residue 344 and not (name CA or name C or name N or␣
↪name O)",
         "contact_volume_thickness": 0.0,
         "contact_volume_trapped_fraction": 0.8,
         "simulation_type": "AV1",
         "simulation_grid_resolution": 0.5,
         "label_interaction_sites": [
            {
               "selection": "MDTraj: resSeq 344",
               "weight": 1.0,
               "radius": 6.0
            }
         ]
        },
        "496D": {
         "chain_identifier": "A",
         "residue_seq_number": 496,
         "atom_name": "CB",
         "linker_length": 20.0,
         "linker_width": 1.5,
         "radius1": 3.5,
         "strip_mask": "MDTraj: residue 496 and not (name CA or name C or name N or␣
↪name O)",
         "contact_volume_thickness": 0.0,
         "contact_volume_trapped_fraction": 0.8,
         "simulation_type": "AV1",
         "simulation_grid_resolution": 0.5,
         "label_interaction_sites": [
            {
               "selection": "MDTraj: resSeq 496",
               "weight": 1.0,
               "radius": 6.0
            }
         ]
        }
    }
}
```

```
import mdtraj as md
import avtraj as avt
import json
import pylab as p
```

(continues on next page)

(continued from previous page)

```
import avtraj.trajectory

traj = md.load('./examples/hGBP1_out_3.h5')
labeling = json.load(open('./examples/labeling.fps.json', 'r'))

# Pass a trajectory to fps.AVTrajectory. This creates an object, which can be
# accessed as a list. The objects within the "list" are accessible volumes
av_traj_1 = avtraj.trajectory.AVTrajectory(traj, '18D', attachment_atom_selection=
↪'resSeq 18 and name CB', strip_mask='resSeq 18')
av_traj_1 = avtraj.trajectory.AVTrajectory(traj, '18D', attachment_atom_selection=
↪'resSeq 18 and name CB', strip_mask='resSeq 7')

# These accessible volumes can be saved as xyz-file
av_traj_1[0].save_xyz('test_18_0.xyz')
av0 = av_traj_1[0]

# Pass a trajectory to fps.AVTrajectory. This creates an object, which can be
# accessed as a list. The objects within the "list" are accessible volumes
av_traj_2 = avtraj.AVTrajectory(traj, '18D', attachment_atom_selection='resSeq 550␣
↪and name CB', strip_mask='resSeq 550')
# These accessible volumes can be saved as xyz-file
av1 = av_traj_2[0]
x, y = av1.pRDA(av0)
p.plot(x, y)


# The dye parameters can either be passed explicitly on creation of the object
av_traj = avtraj.AVTrajectory(traj, '18D', attachment_atom_selection='resSeq 7 and␣
↪name CB', linker_length=25., linker_width=1.5, radius_1=6.0)

# or they can be selected from a predefined set of parameters found in the JSON file␣
↪dye_definition.json located within
# the package directory
av_traj = avtraj.AVTrajectory(traj, '18D', attachment_atom_selection='resSeq 7 and␣
↪name CB', dye_parameter_set='D3Alexa488')

# To calculate a trajectory of distances and distance distributions first a labeling␣
↪file and a "distance file"
# needs to be specified. The distance file contains a set of labeling positions and␣
↪distances and should be compatible
# to the labeling files used by the software "Olga"
av_dist = avtraj.AvDistanceTrajectory(traj, './examples/hGBP1_distance.json')

# For every frame the 'chi2' (for the provided set of distances), the FRET efficiency␣
↪expressed in units of
# a distance 'rDAE', the distance between the mean dye positions 'rMP', and the donor-
↪acceptor distance
# distribution 'pRDA' is calculated. 'pRDA' is a histogram. The histogram bins can be␣
↪specified upon initialization
# of the AvDistanceTrajectory object and are returned as a value.
print av_dist[0]
```

1.3 AV Parameters

1.3.1 Parameter overview

The software Olga and the library avtraj use text files in JavaScript Object Notation (JSON) as exchange data format (Fig. 1). These JSON files:

1. define all necessary parameters for the calculation of accessible volumes (AVs)
2. store experimental data to validate models against experimentally determined distances
3. store information on necessary pre-processing of structural models for successful calculations of accessible volumes
4. instruct the software Olga to actions, e.g., saving accessible volumes to later visualization

```
{
    "version": "0.0.1",
    "Distances": {
        "344-496": {
            "Forster_radius": 52.0,
            "distance": 54.8,
            "distance_type": "RDAMean",
            "error_neg": 6.3,
            "error_pos": 5.5,
            "position1_name": "344A",
            "position2_name": "496D",
            "distance_samples": 100000
        }
    },
    "Positions": {
        "344A": {
         "chain_identifier": "A",
         "residue_seq_number": 344,
         "atom_name": "CB",
         "linker_length": 20.0,
         "linker_width": 1.5,
         "radius1": 3.5,
         "strip_mask": "MDTraj: residue 344 and not (name CA or name C or name N or␣
↪name O)",
         "contact_volume_thickness": 0.0,
         "contact_volume_trapped_fraction": 0.8,
         "simulation_type": "AV1",
         "simulation_grid_resolution": 0.5,
         "label_interaction_sites": [
            {
               "selection": "MDTraj: resSeq 344",
               "weight": 1.0,
               "radius": 6.0
            }
         ]
        },
        "496D": {
         "chain_identifier": "A",
         "residue_seq_number": 496,
         "atom_name": "CB",
         "linker_length": 20.0,
         "linker_width": 1.5,
```

(continues on next page)

(continued from previous page)

```
            "radius1": 3.5,
            "strip_mask": "MDTraj: residue 496 and not (name CA or name C or name N or␣
↪name O)",
            "contact_volume_thickness": 0.0,
            "contact_volume_trapped_fraction": 0.8,
            "simulation_type": "AV1",
            "simulation_grid_resolution": 0.5,
            "label_interaction_sites": [
                {
                    "selection": "MDTraj: resSeq 496",
                    "weight": 1.0,
                    "radius": 6.0
                }
            ]
          }
     }
}
```

The top-level of these JSON files is a dictionary where the most relevant keys are

1. "Distances"
2. "Positions"
3. "version".

Via the keys "Positions" and "Distances" dictionaries can be accessed, which store the necessary information to simulate and compare simulated and experimental distances, respectively (Fig. 1.). The value accessed by the "version" key refers to the version of the JSON file.

The keys of the "Positions" dictionary serve as identifier of the labeling positions, which are referred to be the position names in the "Distances" dictionary.

1.3.2 Distances

Table 1: AV parameters: Distances

Parameter, (type), optional	Options	Description	Example	FPS	Olga	avtraj
distance_type (string), mandatory	"RDAMean", "RDAMeanE", "Rmp", "Efficiency"	The type of distance that is calculated between for the set of labeling positions.	"RDAMean"	Same for all distances	+	+
position1_name (string), mandatory		The name of the first position. This name refers to the entry defined in the dictionary of labeling positions	"Labeling_site_A"	+	+	+
position2_name (string), mandatory		The name of the second position. This name refers to the entry defined in the dictionary of labeling positions	"LP123"	+	+	+
distance (float), mandatory		The reference distance (typically the experimental distance).	45.5	+	+	+
error_neg (float), mandatory		"error_pos" applies if Model – Data > 0 (see description "error_neg").	1.2	+	+	+
error_pos (float), mandatory		"error_pos" applies if Model – Data > 0 (see description "error_neg").	1.4	+	+	+
Forster_radius (float), mandatory for the distance_type options "Efficiency" and "RDAMeanE"		The Forster radius of the dye pair selected by position1_name and position2_name.	51.5	Same for all distances	+	+
distance_samples (int), optional		Optional parameter to set the number of random distance samples to calculate the distance between the two positions. Default value 200000	10000	Same for all distances	+	+
distance_sampling_method (string), optional	"sobol_sequence", "random" (default), "weighted_random"	All distances are weighted by the combined probability. "random": Calculates distances between random points (NOT taken according their weight) in the AVs. "sobol": Calculates for distances between grid points from Sobol-sequence.		-	-	+

1.3.3 Positions

1.4 API Reference

1.4.1 The avtraj API reference

The "av_functions" module

CHAPTER 2

Indices and tables

- genindex
- modindex
- search