Documentation of DEVIS

immediate

July 1, 2022

building blocks for **DEV**eloping your own trajectory analys**IS** tools

- python toolbox and code snippets for creation of own trajectory analyzing scripts

Contents

1 Documentation of muscit functions

1.1 Overall Idea/ General Concept

The muscit python repository is created in order to fulfill two purposes:

- providing a framework for general processing of trajectories for the calculation of RDFs, MSD, ...
- investigation of ion conduction mechanism (via the discretisation of the ion movement on a predefined lattice

2 General framework for processing of trajectories

2.1 Overall Idea/ General Concept

In order provide an efficient framework for the post-processing of trajectories, we convert the trajectory (in .xyz format) into a NumPy .npz data archive format file. This allows a much faster (repeated) reading of the trajectory information for the subsequent analysis of the trajectory. The NumPy .npz data archive format file is automatically created by execution of shell scripts provided within the chrisbase code repository. More directly, the NumPy .npz data archive format file is created by calling the easy_read function. Within the code repository, we provide:

- complex functions which can be executed from the command line and perform a complete analysis of a trajectory such as the calculation of an RDF function.
- python functions which can be imported from the chrisbase repository. These function can be directly called from the ipython interface and perform elementary steps (such as reading a trajectory) for the design of own trajectory analysis scripts.

The positions of the atoms within the .xyz-trajectotry and the NumPy .npz data archive format file do not have to agree. The positions of the atoms within the NumPy .npz data archive format file can be altered by the removal of the center of mass movement (com = True/False) or wrapping/unwrapping of the trajectory to original simulation box (wrap = True/False). We recommend the usage of the **reduce_trajec** command (with parameter "every" = 1) in order to create an exact copy of the atomic positions from the NumPy .npz data archive format file in the.xyz format:

2.2 reduce_trajec

- description: This script prints every "every"-th frame of the trajectory to a new trajectory with the name "new_filename"
- aim1: this script can be used to create a reduced twin system (as a fast testsystem) for processing of trajectories within the "create_jump_mat_li" comand
- aim2: reprinting of the given .xyz-trajectotry with the chosen options for "com" and "wrap"
- usage: reduce_trajec [-h] path pbc com wrap every new_filename
- positional arguments:
  - path1: path to xyz trajecwrap — 2.0809898376464844 seconds — remove com WARNING mass of Li is equal to zero. — 2.967358112335205 seconds —
  - pbc: path to pbc numpy mat
  - com: remove center of mass movement?
  - wrap: wrap trajectory?
  - every: every i-th step from trajec is used for new reduced trajectory
  - new_filename: name of reduced trajectory file
- If the path to the original xyz trajectory is given by the path **"trajec/dummy.xyz"** two new files are created:
  - the NumPy .npz data archive format file: trajec/**dummy.xyz*.npz**. This file consist of a list of the atoms stored in the trajectory and a three dimensional array of the positions of these atoms. The three dimensions of this array are (number of md frames, number of atoms, 3).
  - a .xyz file (new_filename.xyz)

2.3 Using the elementary functions from an ipython interface/ direct calling of python functions provided within the code repository

Here, we want to present specific examples how to use our python functions and code snippets in order to design new trajectory analysing scripts:

2.3.1 Reading the trajectory: easy_read

Starting from the directory of this manual file please type:

```
cd ../manual/example_files/li13si4
```

Launch ipython and type:

```
In [1]: import numpy as np
...: from trajec_io import readwrite
...: path = "trajec/li13si4_timestep_50fs.xyz"
...: pbc_mat = np.loadtxt("trajec/pbc_li13si4")
...: com = True #remove center of mass movement?
...: unwrap = True #wrap/unwrap trejectory?

...: coord, atom = readwrite.easy_read(path, pbc_mat, com, unwrap)
```

The array "atom" contains the types of the atoms. In our case, the system consist of 204 atoms:

```
In [2]: atom.shape
Out[2]: (204,)
```

The atom type of the first atom in the .xyz file is *'Li'*:

```
In [3]: atom[0]
Out[3]: 'Li'
```

The array "coord" contains the coordinates each atom in each timestep. In our case, the positions (x,y and z component) of 204 atoms for 2079 frames are stored:

```
In [4]: coord.shape
Out[4]: (2079, 204, 3)
```

The coordinates of the 13-th atom in the first time step can be accessed via:

```
In [5]: coord[0,12,:]
Out[5]: array([-6.86230942, -4.26715054, -2.77046196])
```

156 Lithium and 48 Silizium atoms are within the simulation box:

```
In [9]: atom[atom == "Li"].shape
Out[9]: (156,)
In [10]: atom[atom == "Si"].shape
Out[10]: (48,)
```

Within one line, we can extract an array containing only the postions of the 156 Li atoms:

```
In [11]: coord_li = coord[:, atom == "Li", :]
In [12]: coord_li.shape
Out[12]: (2079, 156, 3)
```

The function get_com calculates the center of mass of a group of atoms:

```
In [4]: readwrite.get_com(coord[0,:,:], atom, pbc_mat) #calculate center of mass
Out[4]: array([  1.94553368e-15,   1.33226763e-15,   2.74912368e-16])
```

Distances between two positions can be obtained by simple subtraction

```
In [8]: coord_li[2000,42,:] - coord_li[0,42,:] #calculate simple distance by substraction
Out[8]: array([ 17.32749687,   0.70498052,   0.23202467])
```

or by consideration of minimum image convention / periodic boundary conditions (PBC):

```
In [9]: readwrite.pbc_dist(coord_li[2000,42,:], coord_li[0,42,:], pbc_mat) #pbc corrected distance
Out[9]: array([ 1.42949687,  0.70498052,  0.23202467])
```

3 Ion conduction related function

3.1 Overall Idea/ General Concept of ion conduction related functions/code

Within the program package lithium conduction will be analyzed using a user-defined discrete lattice of possible lithium positions. The heart of the program package is the create_jump_mat_li comand.

All subsequent analysis scripts will ONLY make use of the files created by the create_jump_mat_li comand (essentially three different files). Thus, these analysis scripts will have no dependencies with respect to the chrisbase code and can be fully understood by the several lines of code which constitute them. These analysis scripts do not have to be part of the chrisbase repo can be distributed as stand-alone files.

3.2 The create_jump_mat_li comand

- Essentially, three different files (referred to as master files) are necessary to execute all back-end analysis of the conduction mechanism.
- These master files are constructed by the **create_jump_mat_li** command. The **create_jump_mat_li** comes along with the chrisbase package:
  - usage: create_jump_mat_li [-h] path1 pbc com wrap lattice_param jump speed
  - positional arguments:
    - path1: path to xyz trajec
    - pbc: path to pbc numpy mat
    - com: remove center of mass movement?
    - wrap: wrap trajectory?
    - lattice_param: initial_frame or path to lattice coordinates
    - jump: which atoms are transfered?
    - speed: every i-th step from trajec is used for neighbor mat
- If the path to the original xyz trajectory is given by the path **"trajec/dummy.xyz"** these three master files read as:
  - trajec/**dummy.xyz*.npz**: dim(number of md frames, number of atoms, 3) storage of atomic positions and atom files for faster read operations with numpy
  - trajec/**dummy.xyz*neighbor.npz.npy**: dim(number of md frames, number of li atoms, number of lattice sites): for every md frame and lithium atom the next lattice side is stored
  - **jump_mat.npy**: dim(number of md frames, number of lattice sites , number of lattice sites): for each pair of lithium
- Please note, that by execution of **create_jump_mat_li** these master files are only constructed if they do not already exist. This can lead to problems if there are changes in the code of **create_jump_mat_li** or the usage of another xyz file as lattice (i.e. the change of the lattice within a directory is not recommended)
- special care has to be taken with the parameter **lattice_param**:
  - lattice_param = initial_frame uses the lithium positions of the first frame of the trajectory as lattice sites
  - lattice_param = PATH_TO_lattice_xyz_file uses the lithium atoms of the specified xyz file as lattice sites. Please note: **The ordering of the lithium atoms in the lattice xyz file and in the original trajectory has to be equal.**

3.3 Analysis scripts in the muscit repo

list is not complete

3.3.1 jumps_from_grid

- description: Within a given temporal interval (duration1), the script determines groups of lithium sites which are connected by lithium jumps. These groups of connected lithium sites are referred to as jump types. The script determines the different jump types as well as their occurrence.
- usage: jumps_from_grid [-h] path_jump_mat start1 end1 delay1 duration1
- positional arguments:
  - path_jump_mat path to jump mat
  - start1 first frame of jump matrix analyses
  - end1 last frame of jump matricx analyses
  - delay1 distance in frames between two analyses
  - duration1 length of analyses in frames
- description: Within a given temporal interval (duration1), the script determines groups of lithium sites which are connected by lithium jumps. These groups of connected lithium sites are referred to as jump types. The script determines the different jump types as well as their occurrence.
- lattice1.npy is required, not documentated until now (softlink?)

3.3.2 jump_trajek_from_grid

- Within a given temporal interval (duration1), the script determines groups of lithium sites which are connected by lithium jumps. Afterwards a trajectory of the corresponding interval is written and the lithium atom involved into the jump are highlighted by different colors
- usage: jump_trajek_from_grid [-h] path_jump_mat start1 end1 delay1 duration1 noa pbc jump path1 index lattice speed
- positional arguments:
  - path_jump_mat path to jump mat
  - start1 first frame of jump matrix analyses
  - end1 last frame of jump matricx analyses
  - delay1 distance in frames between two analyses
  - duration1 length of analyses in frames
  - noa number of atoms
  - pbc path to pbc numpy mat
  - jump which atoms are transfered?
  - path1 path to xyz trajec
  - index path to file with jump indices
  - lattice path to file with lattice coords
  - speed use every timestep?
- index file has to contain more than one number!
- example: in the directory

  `$PATH1/li15si4/800/short_trajek`

  execute:

  ```
  jump_trajek_from_grid ../jump_mat.npy 0 2100 300 300 76 ../trajec/pbc_li15si4 Li
  ../trajec/Li15Si4-pos-1-proc-ele-800K-100th.xyz ind1 ../trajec/geo_proc.xyz 1
  ```

3.3.3 get_jump_arrows

- description: The number of ion jumps between the lattice sites (within the entire trajectory) is stored and visualized by blue lines. The thickness of the lines is related to the number of ion jumps. The thickness is also normed with respect to the length of the trajectory and thus can be compared between different trajectories.
- usage: get_jump_arrows [-h] path_jump path_lattice filename
- positional arguments:
  - path_jump: jump_matrix
  - path_lattice: lattice_matrix
  - filename: output_name
  - pbc: file with periodic boundary conditions
  - line_rescaling: factor for rescaling of line thickness
  - show_full: only if show full = 100, also (unphysical) jumps through box are shown

3.3.4 wrap_li_to_box

- description: This functions wraps all ions back to the initial simulation box/ image. This is important for correct SDF plots.
- usage: wrap_li_to_box [-h] path pbc com wrap path_geo
- positional arguments:
  - path path to trajek
  - pbc path to pbc numpy mat
  - com remove center of mass movement
  - wrap wrap trajec
  - path_geo path to lattice xyz

3.4 associated analysis scripts not within the chrisbase repo

- path to scripts:

```
$PATH1/scripts
```

- input parameters of these scripts has to be adjusted within the python file
  - `cube_and_saddle_smooth.py:`
    calculate sdf of all li atoms and saddle points of the sdf, degree of smoothing can be adjusted
  - `pic_from_flux.py:`
    deprecated, markov arrows
  - `special_sdf.py`
    calculate sdf from lithium atoms after visiting a certain lattice site
  - `advanced_jump_stat_ready_to_print.py:`
    statistic analysis of the resulting jump types file from the chrisbase script *jumps_from_grid*, results are plotted to dictionary
  - `final_delay_search.py:`
    creates histogram of jump delays, does only need the **jump_mat.npy** file