Microstate Analysis Library

Documentation and examples using the microstate analysis library

- Microstate Analysis Library Reference

Microstate Analysis Library Reference

Microstate Analysis Library Reference

Import library

Suppose the ms_analysis.py is in the current working directory or the Python site-packages directory.

```
from ms_analysis import *
```

Global constants

Once the library is loaded, two global constants (at temperature 298.15 K) are available:

ph2Kcal: Convert ph unit to Kcal/mol
Kcal2kT: Convert Kcal/mol to kT

Load a microstate file

Go to a working directory. The essential files for microstate analysis are:

- head3.lst file
- ms_out folder that contains Monte Carlo sampling microstate output

You need to specify which file to load, such as *ms_out/pH5eH0ms.txt. The name indicates the pH and Eh condition.*

A monte carlo object is reqired to be initialized to hold the microstates with `MC()`

Finally, read the data into the object with `readms()` method.

Example:

```
cd ~/ms_analysis/4lzt
msfile = "ms_out/pH5eH0ms.txt"
mc = MC()
mc.readms(msfile)
```

Load partial Monte Carlo results. A Monte Carlo sampling is carried out 6 times and is numbered as 0, 1, 2, ..., 5. One can choose to load some of them:

```
mc.readms(msfile, MC=[1,2])
```

Data structure:

Conformer:

Conformer is a class object.

Variables:

- **iconf**: Integer - index of conformer, starting from 0
- **confid**: String - conformer name as in head3.lst
- **resid**: String - unique residue name including name, chain ID and sequence number
- **crg**: Float - net charge

Microstate:

Microstate is a class object.

Variables:

- **stateid**: String - compressed and encoded string to identify a microstate
- **E**: Float - microstate energy
- **count**: Integer - how many times this microstate is accepted

Function:

- **state()**: return a microstate, which is a list of selected conformers

Charge_Microstate:

Charge_Microstate is a class object. If we only care about residue ionization, we can reduce conformer microstates to charge microstates.

Variables:

- **crg_stateid**: String - compressed and encoded string to identify a charge microstate
- **average_E**: Float - average charge microstate energy
- **count**: Integer - how many times this charge microstate is accepted

Function:

- **state()**: return a charge microstate, which is a list of net charges, in the same order of free residues

Subset_Microstate:

Subset_Microstate is a class object. If we only care about a selected group of residues, we can group microstates based on the conformer selection of these residues only.

Variables:

- **subset_stateid**: String - compressed and encoded string to identify a subset microstate
- **average_E**: Float - average subset microstate energy
- **count**: Integer - how many times this subset microstate is accepted

Function:

- **state()**: return a subset microstate, which is a list of selected conformers of interested residues

Free_res:

Free_ress is a class object. It holds information of a free residue.

Variables:

- **resid**: String - residue identification name
- **charges**: list of floating point numbers - a list of charge choices

MC:

MC is a class object. It holds information of a Monte Carlo microstates output.

Variables:

- **T**: Float - Monte Carlo sampling temperature
- **pH**: Float - Monte Carlo sampling pH
- **Eh**: Float - Monte Carlo sampling Eh
- **method**: String - This indicates the microstates output is from either Monte Carlo sampling or Analytical Solution
- **counts**: Integer - Total number of Monte Carlo steps
- **conformers**: A list of Conformer objects that matche the entries in head3.lst
- **iconf_by_confname**: A dictionary that returns conformer index number from conformer name
- **fixedconfs**: A list of fixed confomer index numbers
- **free_residues**: A list of conformer groups (each group is a list of conformer indicies) that make up free residues
- **free_residue_names**: A list of free residue names
- **microstates**: A list of Microstate objects. They are accepted microstates.

Function:

- **readms(fname, MC=[])**: read microstate output file and return a list of microstates. You can optionally choose what parts of Monte Carlo output to load. MC=[] means to choose all. MC=[1,2] means to choose 1st and 2nd MC runs. The valid numbers are from 0 to 5.
- **get_occ(microstates)**: Convert a list of microstates to occupancy. It reads in a list of conformers and returns a list of occupancy (0.0 to 1.0) numbers on each conformer.

  > This function does not work on charge microstates or subset microstates.

- **confnames_by_iconfs(iconfs)**: Convert a list if conformer indices to a list of conformer names.
- **select_by_conformer(microstates, conformer_in=[])**: Select from given microstates if confomer is in the list. Return all if the list is empty. The input conformer_in is a list of conformer names.
- **select_by_energy(microstates, energy_in=[])**: Select from given microstates if the microstates' energy is within the range defined by energy_in. energy_in should be given an array with lower bound (inclusive) and a higher bound (exclusive).
- **convert_to_charge_ms()**: Convert all microstates to a list charge microstate objects.
- **convert_to_subset_ms(res_of_interest)**: Convert all microstates to a list subset microstate objects. The input res_of_interest is a list of residues of interest, in the form of residue names. These residues have to be free residues.

Functions:

get_erange(microstates)

Get the energy range of given microstates.

Input:

- **microstates**: A list of microstates object

Output:

> A list of two numbers that are lower bound and higher bound of energey

bin_mscounts_total(microstates, nbins=100, erange=[])

Divide microstates into bins based on energy and get the counts of total steps in each bin.

Input:

- **microstates**: A list of microstates object
- **nbins**: the number of desired bins. Default value is 100
- **erange**: custom energy range. It is a list of lower bounds of bins

Output:

> It returns two lists. The first list is the energy range in the form of lower bounds. The second list is number of microstate counts of each bin.

bin_mscounts_unique(microstates, nbins=100, erange=[])

Divide microstates into bins based on energy and get the counts of unique microstates in each bin.

Input:

- **microstates**: A list of microstates object
- **nbins**: the number of desired bins. Default value is 100
- **erange**: custom energy range. It is a list of lower bounds of bins

Output:

> It returns two lists. The first list is the energy range in the form of lower bounds. The second list is number of microstate counts of each bin.

get_count(microstates)

Divide microstates into bins based on energy and get the counts of unique microstates in each bin.

Input:

- **microstates**: A list of microstates object
- **nbins**: the number of desired bins. Default value is 100
- **erange**: custom energy range. It is a list of lower bounds of bins

Output:

> It returns two lists. The first list is the energy range in the form of lower bounds. The second list is number of microstate counts of each bin.

average_e(microstates)

Calculate the average energy of given microstates.

Input:

- **microstates**: A list of microstates object

Output:

> Average energy.

Code and example:

- Library: ms_analysis.py
- Demo: demo.ipynb