GEMC_histogram_analysis

Release v1.0

Robert F. DeJaco

Jul 21, 2019

CONTENTS:

CHAPTER

ONE

GEMC_HISTOGRAM_ANALYSIS

An analysis package utilizing histograms to calculate accurate vapor and liquid coexistence densities, saturated vapor pressure, and compressibility factor from a Gibbs ensemble Monte Carlo trajectory for unary vapor–liquid equilibria near the critical point.

CHAPTER

TWO

CITATION

Users of this package should cite:

B.L. Eggimann, Y.Z.S. Sun, R.F. DeJaco, R. Singh, M. Ahsan, T.R. Josephson, and J.I. Siepmann. Assessing the quality of molecular simulations for vapor–liquid equilibria: An analysis of the TraPPE database. *J. Chem. Eng. Data*, submitted for publication on 7/22/2019.

CHAPTER

THREE

INSTALLATION

This program is intended to be used on python3.X. It won't work on python2.X because it uses new types of importing that are only available in python3.X. The python packages can be installed by the usual route like

```
pip install requirements.txt
```

but use of a virtualenvironment or a conda environment is recommended. If the user is unfamiliar with these approaches, the necessary packages will come with almost all anaconda python installations which are available on supercomputers.

To test your installation of the program, look through the functions in *examples.py* and run one of the examples.

GEMC_histogram_analysis runs on Python 3.x only (not Python 2.x).

CHAPTER

FOUR

USAGE

The program can generate two types of histograms. A histogram of densities for determining bulk properties is generated with the file *run_histogram.py*. A histogram of the sampling trajectory can be generated with the file *plot_trajectory.py*. Either program can be run by

```
python eitherprogram -fi file1 file2 file3 ... -T T1
```

where *eitherprogram* is either *run_histogram* or *plot_trajectory.py*, *file1*, *file2*, *file3*, and . . . are the relative paths to all the trajectory files from one independent simulation, and *T1* is the temperature at which the simulation was conducted in Kelvin. All arguments (including those which are optional) can be seen by appending *-h* or *–help* to the line above.

It is not necessary to use this command line approach to run these programs. To see how to do this, take a look at the file *examples.py* or the functions *example_butane()* and *example_pentanal()* within the file *plot_trajectory.py*.

CHAPTER

FIVE

USEFUL SCRIPTS

5.1 run_histogram.py

The *run_histogram.py* program will calculate mean and errors for pressures, densities, and compressibilities, and compare to the block averages approach.

The mean and error of the density values with the histogram approach are obtained from the density probability distributions.

The pressure and compressibilities obtained from the histogram approach are calculated as the average of all pressures and compressibilities at MCCs where the pressure was calculated and the density was in the high density or low density regime (as defined by 75% of the respective peak height). The errors are the standard deviations of those set of values. These values for one independent simulation correspond to the fluctuations.

5.2 plot_trajectory.py

CHAPTER

SIX

METHODOLOGY

The statistics from the production trajectory of a NVT Gibbs ensemble Monte Carlo simulation are collected. These statistics entail $\mathcal{D}$, the distribution of total box densities $\rho_{i,j}$ calculated in each simulation box i at the end of the j-th Monte Carlo Cycle (MCC), and $\mathcal{P}$, the distribution of box pressures $P_{i,k}$ calculated in each simulation box i at the end of the k-th MCC where the pressure was calculated. Usually, the box pressures are calculated once every five to ten MCCs, so that the number of entries in $\mathcal{P}$ is less than those in $\mathcal{D}$.

The distribution of all observed box densities $\mathcal{D}$ is partitioned into subsets of low ($\mathcal{L}$) and high ($\mathcal{H}$) densities, such that $\rho_{i,l} < \rho_m \forall i \forall l \in \mathcal{L}$ and $\rho_{i,h} \geq \rho_m \forall i \forall h \in \mathcal{H}$ where ρ_m is the mean of all density values in $mathcal D$.

For each distribution, $\mathcal{L}$ and $\mathcal{H}$, a histogram of the densities is performed with N_B equally-spaced bins (edges) from the minimum to maximum value of density using [NUMPY]. The probabilities $Y_i \forall i \in \mathcal{Y}$ of observing each density within each i nodes (adjacent bins) are normalized so that the sum is equal to unity. The $N_\mathrm{B} - 1$ density values $\rho_i \forall i \in \mathcal{Y}$ for each node (within adjacent edges) are assumed to be the midpoint between adjacent edges. A subset of representative probabilities, $\mathcal{Y}^0$, is chosen as follows

$$Y_j \geq f \max_{i \in \mathcal{Y}} Y_i \quad \forall j \in \mathcal{Y}^*$$

where f is the fraction of peak maximum. The fraction of peak maximum is typically 0.75 [DINPAJOOH2015]. These probabilities and associated densities, $\rho_i \forall i \in \mathcal{Y}^*$ are fit to a gaussian function. The initial guess for the peak height, mean, and standard deviation of the gaussian are obtained from $\max_{\forall i \in \mathcal{Y}^*} Y_i$, $\mu_{\forall i \in \mathcal{Y}^*} \rho_i$, and the standard deviation of of the distribution $\sigma_{\mathcal{Y}^*}$, respectively. The lower bounds allowed for the peak height, mean, and standard deviation are zero, the minimum density, and zero, respectively. The upper bounds allowed for the peak height and mean are $2 \max_{\forall i \in \mathcal{Y}^*} Y_i$ and $\max_{\forall i \in \mathcal{Y}^*} \rho_i$, respectively. The maximum in the standard deviation parameter obtained from the fitting of the gaussian is not constrained to a maximum finite value. The constrained parameter fitting is performed using the Trust Region Reflective algorithm as implemented in [SCIPY].

Finally, the minimum density, $\rho_{\chi,\min}$, and maximum density $\rho_{\chi,\max}$, for each distribution χ ($\mathcal{H}$ or $\mathcal{L}$) corresponding to the chosen fraction of peak maximum f are calculated from the gaussian function and the optimized parameters for each peak.

The final high ($\mathcal{H}^*$) and low ($\mathcal{L}^*$) distributions of densities are determined as follows

$$\begin{aligned} \rho_{\mathcal{H},\min} \leq \rho_i \leq \rho_{\mathcal{H},\max} \quad & \forall i \in \mathcal{H}^*, \ni i \in \mathcal{H} \\ \rho_{\mathcal{L},\min} \leq \rho_i \leq \rho_{\mathcal{L},\max} \quad & \forall i \in \mathcal{L}^*, \ni i \in \mathcal{L} \end{aligned}$$

Similarly, the pressures corresponding to the high ($\mathcal{P}_{\mathcal{H}^*}$) and low ($\mathcal{P}_{\mathcal{L}^*}$)distributions of densities are determined as

$$\begin{aligned} \rho_{\mathcal{H},\min} \leq \rho_i \leq \rho_{\mathcal{H},\max} \quad & \forall i \in \mathcal{P}_{\mathcal{H}^*}, \ni i \in \mathcal{P} \\ \rho_{\mathcal{L},\min} \leq \rho_i \leq \rho_{\mathcal{L},\max} \quad & \forall i \in \mathcal{P}_{\mathcal{L}^*}, \ni i \in \mathcal{P} \end{aligned}$$

The compressibility distributions at the high ($\mathcal{Z}_{\mathcal{H}^*}$) and low ($\mathcal{Z}_{\mathcal{L}^*}$) densities are calculated at the densities and pressures within $\mathcal{P}_{\mathcal{H}^*}$ and $\mathcal{P}_{\mathcal{L}^*}$, respectively. The mean and standard deviation of each density regime ($\chi \in \{\mathcal{H}^*, \mathcal{L}^*\}$) for pressure, compressibility, and density are determined from the χ, $\mathcal{P}_\chi$, and $\mathcal{Z}_\chi$ distributions, respectively.

CHAPTER

SEVEN

HISTOGRAM

class histogram.density_histogram.**Blocks**(***kwargs*)

Read input arguments and initialize data types for one independent simulation.

check_arguments()
Check that arguments passed in are realistic :return: None

choose_output_file_path()
Based off of input files, try to guess where to output

Returns None

initialize_data()
Read all trajectory files and input into class attribute data types

Returns None

main()
Main method of :class: First, calls *choose_output_file_path()* Then, calls *initialize_data()* Finally, outputs data from *output_blocks()* :return: None

output_blocks()
Output block averages into a csv file or pprint based off of input arguments.

Convert units to g/mL if this was requested in args

Returns None

class histogram.density_histogram.**DPD**(***kwargs*)
Read input arguments and initialize data types for one independent simulation. Then, make associated histograms, plot them if desired, and output them into desired format.

calculate_P_Z()
Calculate pressure and compressibilities for two different types of distributions

Store data of mean and standard deviation of raw data values in `histograms` within `data`

Returns None

check_arguments()
Check if arguments passed in are realistic :return: None

fit_density_peaks()
Fit gaussian to both density peaks. The following procedure is followed

1. Convert total box densities for each box type into one single array.
2. Calculate the mean density out of all densities observed.

3. **Partition the densities into two regions–those which** have values lower or higher than this mean value.
4. **Within each new distribution,** perform a histogram using the number of bins requested during instantiation. The edges of the histogram correspond to the number of bins chosen, and the values are their corresponding probabilities normalized so that the sum of all probabilities is 1.
5. **Partition histogram bins into those which have values** above the fractional threshold of the maximum value. Fit a gaussian function of the probabilities of observing each density as a function of density.

Store the results of the binning for each region in `histograms` for use later (plotting, outputting)

Returns None

fit_gaussian(*bin_means*, *values*)

Fit gaussian function, *gaussian()*

Parameters

- **bin_means** – means of density bins (nodes)
- **values** – values at means of density bins

The initial guess for the peak height, mean, and standard deviation of the gaussian are obtained from the maximum of the values above the threshold, the mean of the densities above the threshold, and the standard deviation of the densities above the threshold, respectively.

The minimum value allowed for the peak height, mean, and standard deviation of the gaussian are 0, the minimum density above the threshold, and 0, respectively.

The maximum value allowed for the peak height and mean of the gaussian are twice the maximum of the values above the threshold and the maximum density above the threshold, respectively. The standard deviation of the gaussian is not constrained to a finite maximum threshold.

The parameters are fit using curve_fit() with the analytical jacobian `J_gaussian()` and the initial guess and bounds listed above.

Finally, the bounds for the values according to the fraction of the threshold are converted into continuous bounds in density.

Returns

- params(list) the gaussian parameters fit
- covariance(list) is the covariance of the gaussian parameters
- min_rho_gauss(float) is the minimum density of the gaussian within the threshold specified
- max_rho_gauss(float) is the minimum density of the gaussian within the threshold specified
- num_indices(int) the number of indices used in the fitting of the gaussian

histogram_P_Z_data(*data*, *data_type*)

Obtain histogram(s) of pressure or compressibility data within region(s) or box(es)

Parameters **data** (*dict*) –

Returns results of histograms

Return type dict

main()
Main method of :class: First, calls `choose_output_file_path()` Then, calls `initialize_data()` The peaks are fit using *`fit_density_peaks()`* and the corresponding pressure and compressibilities are obtained from *`calculate_P_Z()`* and then plot results if `make_plot` == 'Yes' and then output results with `output_results()` :return: None

output_results()
Output information stored in `data`

Change to g/mL units if was desired during instantiation

Output results either by printing or saving to csv file, as requested by input arguments

Returns None

store_data()
Store data of mean and standard deviation of raw data values in `histograms` within `data`

Returns None

`histogram.density_histogram.`**calculate_compressibility**(*total_densities*, *pressures*, *T*)
Calculate compressibility factors from total densities and pressures

Parameters

- **total_densities** (*dict*) – total densities of boxes in molecules / nm**3
- **pressures** (*dict*) – pressures in kPa
- **T** (*float*) – Temperature in Kelvin

Returns List of compressibility factors of each box for each point provided

`histogram.density_histogram.`**calculate_density**(*volumes*, *number_of_chains*, *cycles*)
Convert input number of chains and volumes to densities

Parameters

- **volumes** (*dict*) – volume of box at end of each cycle in nm**3
- **number_of_chains** (*dict*) – number of molecules of each type and in each box during each cycle
- **cycles** (*list*) – cycles over which to calculate density

Returns densities

`histogram.density_histogram.`**calculate_total_densities**(*dens*, *boxes*)
Calculate the total (molar) density in a box by summing individual number densities

ho_{mathrm{total}} = sum_i ho_i

param dens densities in molecule/nm**3 for each box for each molecule

type dens dict

param boxes boxes of simulation

type boxes list

return

histogram.density_histogram.**gaussian**(*x*, *a*, *b*, *c*)
Gaussian function

$$f(x) = ae^{-(x-b)^2/2/c^2}$$

Parameters

- **x** – input variable
- **a** – height of curve peak
- **b** – center of peak
- **c** – standard deviation

Returns f(x)

histogram.density_histogram.**jac_gaussian**(*x*, *a*, *b*, *c*)
Jacobian of *gaussian()*

Parameters

- **x** – input variable
- **a** – height of curve peak
- **b** – center of peak
- **c** – standard deviation

Returns

$$\frac{\partial f}{\partial a}, \frac{\partial f}{\partial b}, \frac{\partial f}{\partial c}$$

histogram.density_histogram.**make_block_averages**(*data*, *number_blocks*)
Make block averages based off of number of blocks and data provided.

Parameters

- **data** (*dict*) – a thermodynamic property measured at each point for each cycle
- **number_blocks** (*int*) – number of nodes over which to average property

Returns means and standard deviations of data for each box

histogram.density_histogram.**parse_arguments**()
Command-line parser for input data

Returns arguments to pass to either *Blocks* or *DPD* during instantiation

Return type dict

histogram.density_histogram.**update_data_box**(*store_data*, *new_data*)
Updata information stored for data that is only box specific

Parameters

- **store_data** – stored information of given data
- **new_data** – new information to add to stored information

Returns None

histogram.density_histogram.**update_data_molty**(*store_data*, *new_data*)
Updata information stored for data that depends on both box and molty

Parameters

- **store_data** – stored information of given data
- **new_data** – new information to add to stored information

Returns None

histogram.reader.**read_fort12**(*file_name*, *cycle_start*)

Read fort.12 (trajectory) file for MCCCS-MN code output file style. Most inner data type is {cycle: value}. This is helpful for finding out what cycle a certain value occured at, etc.

Parameters

- **file_name** (*str*) – name of file to open and read
- **cycle_start** (*int*) – cycle number to start at (for labeling data explicitly at each cycle)

Returns (molecular_weights, number_of_molecules, box_volume, box_pressure, iratp, cycle_counter)

class histogram.trajectory_histogram.**Trajectory**(***kwargs*)

initialize_data()

Read all trajectory files and input into class attribute data types

Returns None

main()

Main workflow for trajectory

Returns None

CHAPTER

EIGHT

SCRIPTING FILES

run_histogram.**run**()
Perform histogram analysis. Main function of run_histogram.py. :return: None

example workflows for using the histogram code. To run the examples, call the functions in this file

examples.**butane**(*T*)

analyze the data for butane at T K, located in the example_data/butane_XXXK directory where XXX is the temperature

examples.**example_bin_width**()

analyze effect of number of bins chosen on the data for pentanal at 360K, located in the example_data/pentanal_360K directory

examples.**pentanal**(*list_of_indep_sims*, *T=0*, *number_of_bins=0*, *average_fmt=''*)

analyze the data for pentanal at a given temperature=T, located in the example_data/pentanal_XXXK directory where XXX is the temperature

Parameters

- **list_of_indep_sims** (*list*) – independent simulations for which to average over
- **T** (*float*) – temperature of simulation in K
- **number_of_bins** (*int*) – number of bins to analyze over for each region
- **average_fmt** (*str*) – subscript for identifier in data output

CHAPTER

NINE

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[NUMPY] Oliphant, T. NumPy: A guide to NumPy. 2006–2019. USA: Trelgol Publishing. http://www.numpy.org/

[DINPAJOOH2015] Dinpajooh M, Bai P, Allan DA, and Siepmann JI. Accurate and precise determination of critical properties from Gibbs ensemble Monte Carlo simulations. J. Chem. Phys. 143, 114113 (2015), https://doi.org/10.1063/1.4930848

[SCIPY] Jones E, Oliphant E, Peterson P, et al. SciPy: Open Source Scientific Tools for Python. 2001–2019. http://www.scipy.org/

PYTHON MODULE INDEX

INDEX