mdtoolbox Documentation

Release 1.0

Yasuhiro Matsunaga

Jun 19, 2019

Contents

CHAPTER 1

Introduction

1.1 Introduction

1.1.1 What is MDToolbox?

MDToolbox is a MATLAB/Octave toolbox for statistical analysis of molecular dynamics (MD) simulation data of biomolecules. It consists of a collection of functions covering the following types of scientific computations:

- I/O for trajectory, coordinate, and topology files used for MD simulation
- Least-squares fitting of structures
- Potential mean force (PMF) or free energy profile from scattered data
- Statistical estimates (WHAM and MBAR methods) from biased data
- Dimensional reductions (Principal Component Analysis, and others)
- Elastic network models (Gaussian and Anisotropic network models)
- Utility functions, such as atom selections

MDToolbox is developed on GitHub. Freely available under the BSD license.

1.1.2 Requirements

MDToolbox is developed and tested on MATLAB R2013a and later versions.

We are also testing the toolbox on GNU Octave version 3.8.2. As far as we have checked, most of functions should work on Octave version 3.8.2 or laters.

1.1.3 Download

Zip arichive or tarball of the latest version is available from GitHub, or the repository can be directly cloned from GitHub by using git,

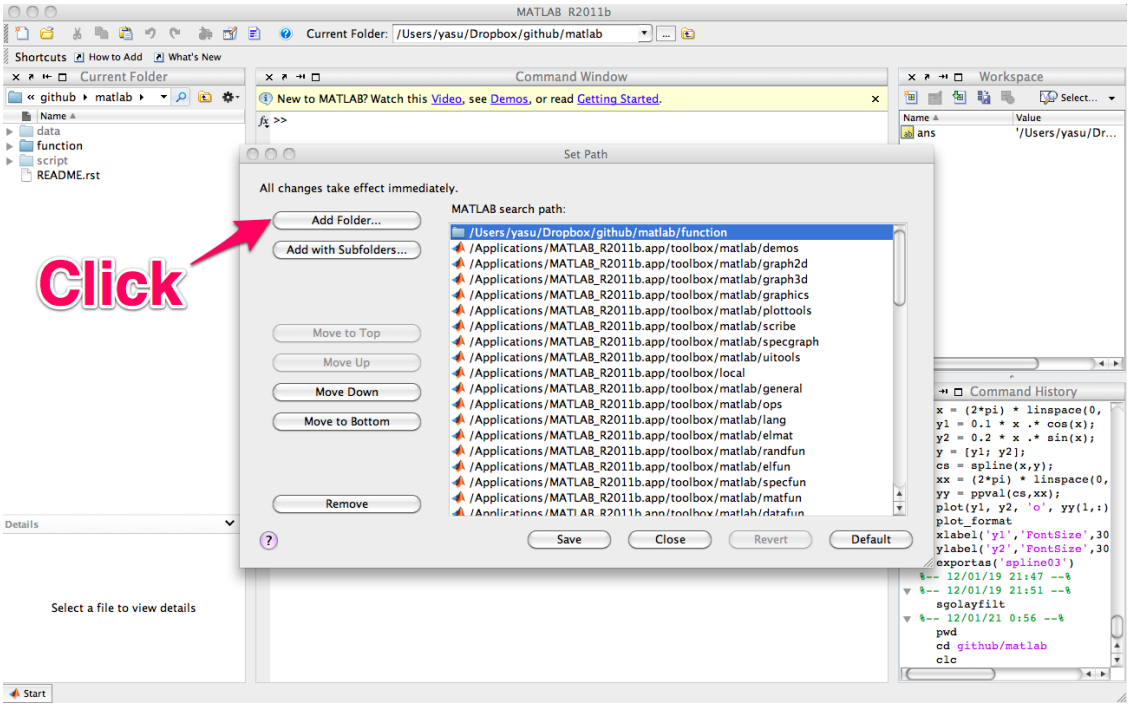
```
$ git clone https://github.com/ymatsunaga/mdtoolbox.git
```

1.1.4 Installation for MATLAB

For personal installation, the personal startup file may be found at `~/matlab/startup.m`. If it does not exist, create one. Add the following line to `startup.m` with full path to the directory of MDToolbox m-files,

```
addpath('/path/to/mdtoolbox/mdtoolbox/')
```

For system-wide installation, call `pathtool` command in MATLAB and add the directory to the user's MATLAB search path (root permission is required to save the path),



1.1.5 Installation for Octave

For personal installation, the personal startup file may be found at `~/octaverc`. If it does not exist, create one. Add the following line to `~/octaverc` with full path to the directory of MDToolbox m-files,

```
addpath('/path/to/mdtoolbox/mdtoolbox/')
```

To use I/O functions for NetCDF files (e.g., AMBER NetCDF trajectory), netcdf package needs to be installed in Octave.

1.1.6 Compiling MEX-files and multithreading

In addition to the m-files, MEX-files are prepared for core functions to accelerate the performance. **We strongly recommend to use these MEX-files for reasonable performance in MATLAB/Octave**. To use MEX-files, the user needs to compile the files in advance. For the compilation, use `make.m` script in MATLAB/Octave:

```
>> cd /path/to/mdtoolbox/
>> make
```

Warnings during the compilation can be safely ignored.

On Linux platforms, OpenMP option can be enabled for further performance by parallel computation (multithreading),

```
>> make('openmp')
```

For parallel run, make sure to set your environment variable (`OMP_NUM_THREADS`) before starting up MATLAB/Octave. For example, if you want to use 8 threads(=CPU cores) parallelization, the variable should be set from the shell prompt as follows:

```
# for sh/bash/zsh
$ export OMP_NUM_THREADS=8
# for csh/tcsh
$ setenv OMP_NUM_THREADS 8
```

1.1.7 Docker image for MDToolbox

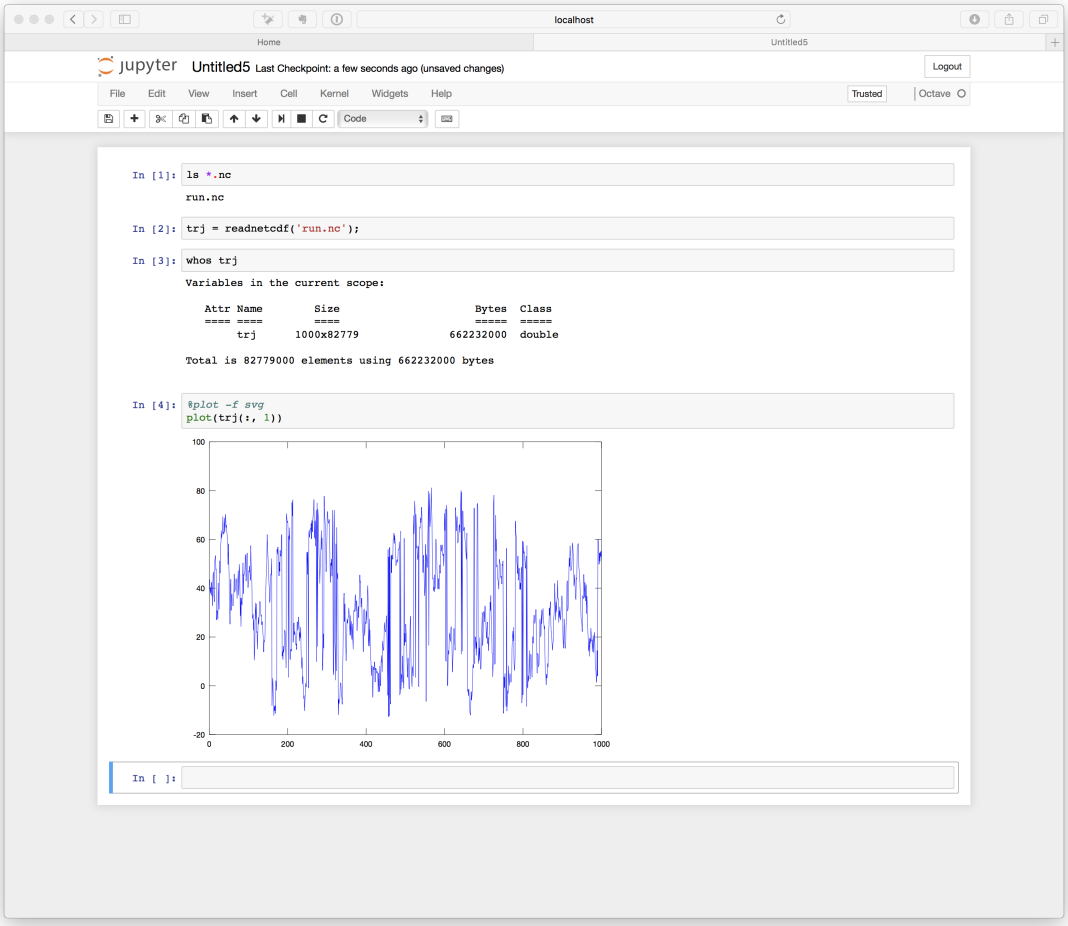
A docker image for MDToolbox is avaiable here. In this docker image, you can use Octave already configured for use with MDToolbox (downloading MDtoolbox, path setup, MEX file compiling, etc are already completed).

By just running a docker command, you can immediately use MDToolbox (without GUI),

```
$ docker run -it --rm -v $(pwd):/home/jovyan/work ymatsunaga/octave octave
```

Or you can use Octave + MDToolbox within Jupyter notebook (with GUI),

```
$ docker run --rm -p 8888:8888 -v $(pwd):/home/jovyan/work ymatsunaga/octave
then, access to the Jupyter notebook via browser
```



For details of the usage, please see our docker image site.

1.1.8 Summary of main functions

Representative functions of MDToolbox are summarized in the tables below. For detail of each function, use `help` command in MATLAB. For example, usage of `readdcd()` function can be obtained as follows:

```
>> help readpdb
readdcd
read xplor or charmm (namd) format dcd file

 % Syntax
 # trj = readdcd(filename);
 # trj = readdcd(filename, index_atom);
 # [trj, box] = readdcd(filename, index_atom);
 # [trj, box, header] = readdcd(filename, index_atom);
 # [trj, ~, header] = readdcd(filename, index_atom);

 % Description
  The XYZ coordinates of atoms are read into 'trj' variable
  which has 'nstep' rows and '3*natom' columns.
  Each row of 'trj' has the XYZ coordinates of atoms in order
  [x(1) y(1) z(1) x(2) y(2) z(2) ... x(natom) y(natom) z(natom)].
```

(continues on next page)

(continued from previous page)

```
 * filename   - input dcd trajectory filename
 * index_atom - index or logical index for specifying atoms to be read
 * trj        - trajectory [nstep x natom3 double]
 * box        - box size [nstep x 3 double]
 * header     - structure variable, which has header information
                [structure]

% Example
# trj = readdcd('ak.dcd');

% See also
 writedcd

% References for dcd format
 MolFile Plugin http://www.ks.uiuc.edu/Research/vmd/plugins/molfile/dcdplugin.html
 CafeMol Manual http://www.cafemol.org/doc.php
 EGO_VIII Manual http://www.lrz.de/~heller/ego/manual/node93.html
```

Inuput/Output

name	description
readpdb	read Protein Data Bank (PDB) file
writepdb	write Protein Data Bank (PDB) file
readprmtop	read amber parameter/topology file
readambercrd	read amber coordinate/restart file
readamberout	read amber output file
readmdcrd	read amber ascii-format trajectory file
readmdcrdbox	read amber ascii-format trajectory file including box size
readnetcdf	read amber netcdf file
writeambercrd	write amber coordinate/restart file
writemdcrd	write amber ascii-format trajectory format file
writenetcdf	write amber netcdf file
readpsf	read charmm or xplor type Protein Structure File (PSF)
readdcd	read xplor or charmm (namd) format dcd file
readnamdbin	read namd restart (namdbin) file
readnamdout	read namd output file
writedcd	write xplor or charmm (namd) format dcd file
writenamdbin	write namd restart (namdbin) file
readgro	read gromacs gro (Gromos87 format) file
writegro	write gromacs gro (Gromos87 format) file
readdx	read dx (opendx) format file
writedx	write dx (opendx) format file

Geometry calculations (fitting of structures, distance, angles, dihedrals, etc)

name	description
superimpose	least-squares fitting of structures
meanstructure	calculate average structure by iterative superimposing
decenter	remove the center of mass from coordinates or velocities
orient	orient molecule using the principal axes of inertia
searchrange	finds all the atoms within cutoff distance from given atoms
calcbond	calculate distance from the Cartesian coordinates of two atoms
calcangle	calculate angle from the Cartesian coordinates of three atoms
calcdihedral	calculate dihedral angle from the Cartesian coordinates of four atoms
calcpairlist	make a pairlist by searching pairs within a cutoff distance
calcqscore	calculate Q-score (fraction of native contacts) from given heavy atom coordinates

Statistics (WHAM, MBAR, clustering, etc)

name	description
wham	Weighted Histogram Analysis method (WHAM)
ptwham	Parallel tempering WHAM (PTWHAM)
mbar	multi-state Bennett Acceptrance Ratio Method (MBAR)
mbarpmf	evaluate PMF from the result of MBAR
calcpmf	calculate 1D potential of mean force from scattered 1D-data (using kernel density estimator)
calcpmf2d	calculate 2D potential of mean force from scattered 2D-data (using kernel density estimator)
calcpca	peform principal component analysis (PCA)
calctica	perform time-structure based Independent Component Analysis (tICA)
clusterkcenters	clustering by K-centers
clusterhybrid	Hybrid clustering by using K-centers and K-medoids
clusterkmeans	clustering by K-means

Anisotropic Network Model

name	description
anm	calculate normal modes and anisotropic fluctuations by using Anisotropic Network Model.
anmsparse	calculate normal modes of ANM using sparse-matrix for reducing memory size
anmsym	calculate normal modes of ANM for molecule with circular symmetry using symmetric coordinates
transformframe	transform the normal modes from the Eckart frame to a non-Eckart frame

Utility functions (atom selections, index operations, etc)

name	description
selectname	used for atom selection. Returns logical-index for the atoms which matches given names
selectid	used for atom selection. Returns logical-index for the atoms which matches given index
selectrange	used for atom selection. Returns logical-index for the atoms within cutoff distance from given atoms
to3	convert 1…N atom index (or logical-index) to 1…3N xyz index (or logical-index)
formatplot	fomart the handle properties (fonts, lines, etc.) of the current figure
exportas	export fig, eps, png, tiff files of the current figure
addstruct	create a structure by making the union of arrays of two structure variables
substruct	create a subset structure from a structure of arrays

1.2 Getting Started

1.2.1 Input/Output

Typical usages of I/O functions for MD files are follows.

PDB

PDB file

```
pdb = readpdb('protein.pdb'); % pdb is a structure variable containg ATOM records
[pdb, crd] = readpdb('protein.pdb'); % if you want to extract the coordinate
% after some calculations
writepdb('protein_edit.pdb', pdb);
writepdb('protein_edit.pdb', pdb, crd); % if you want to replace coordinate with crd
```

AMBER files

AMBER log

```
ene = readamberout('amber.out'); % ene is a structure variable containing energy terms
```

AMBER parameter/topology file

```
prmtop = readprmtop('run.prmtop'); % prmtop is a structure variable containing␣
↪topology information
```

AMBER trajectory file

```
natom = 5192; % the number of atoms is required for reading AMBER trajectory
trj = readmdcrd(natom, 'run.trj'); % trajectory file without box size
trj = readmdcrdbox(natom, 'run.trj'); % trajectory file with box size
% after some calculations
writemdcrd('run_edit.trj', trj);
```

AMBER NetCDF trajectory file

```
trj = readnetcdf('run.nc');
% after some calculations
writenetcdf('run_edit.nc', trj);
```

CHARMM/NAMD files

NAMD log

```
ene = readnamdout('namd.out'); % ene is a structure variable containing energy terms
```

PSF file

```
psf = readpsf('run.psf'); % psf is a structure variable containing energy terms
```

DCD file

```
trj = readdcd('run.dcd');
% after some calculations
writedcd('run_edit.dcd', trj);
```

GROMACS files

GRO file

```
gro = readgro('run.gro'); % gro is a structure variable containg ATOM records
% after some calculations
writegro('run_edit.gro', gro);
```

Support of TRR and XTC files is on-going.

1.2.2 Coordinate and trajectory variables

MDToolbox assumes a simple vector/matrix form for coordinate/trajectory.

Coordinate variable is a row vector whose elements are the XYZ coordinates of atoms in order

```
[x(1) y(1) z(1) x(2) y(2) z(2) .. x(natom) y(natom) z(natom)]
```

Thus, for example, translation in the x-axis by 3.0 Angstrom is coded as follows:

```
crd(1:3:end) = crd(1:3:end) + 3.0;
```

Likewise, the y-coordinate of geometrical center is calculated by

```
center_y = mean(crd(2:3:end));
```

Trajectory variable is a matrix whose row vectors consist of coordinate variables. The rows represent frames in the trajectory. For simulation data, the sequence of frames represents molecular dynamics.

Thus, for example, the coordinate at the 10th frame is extracted by

```
crd = trj(10, :);
```

Translation in the x-axis throughout all frames is coded as

```
trj(:, 1:3:end) = trj(:, 1:3:end) + 3.0;
```

Average coordinate over all frames (without fitting) is obtained by

```
crd = mean(trj);
```

1.2.3 Atom selection

MDToolbox uses logical indexing for atom selection. Logical indexing is a vector or matrix whose elements consist of logical variables, i.e., true(==1) and false(==0). Logical indexing is useful for selecting subset of vector/matrix that matches a given condition in MATLAB.

For example, the following example returns a logical indexing whose elements are greater than 1:

```
>> x = [1 2 3];
>> index = x > 1

index =

     0     1     1

>> whos index
  Name               Size             Bytes  Class      Attributes

  index              1x3                  3  logical

>> x(index)

ans =

     2     3
```

Another advantage of logical indexing is that it is easy to combine the results of different conditions to select subset on multiple criteria. The following example selects the subset whose elements are greater than 1, and also smaller than 3:

```
>> index2 = x < 3

index2 =

     1     1     0

>> index3 = index & index2  % Boolean AND

index3 =

     0     1     0

>> x(index3)

ans =

     2
```

MDToolbox has three types of atom-selection functions; `selectname()`, `selectid()`, and `selectrange()`. All of them returns logical indexing for use with other MDToolbox functions or selecting subset of coordinate or trajectory variable.

`selectname()` returns a logical indexing which matches given names (characters). The following code returns logical indexing of alpha-carbon atoms,

```
[pdb, crd] = readpdb('example/anm_lys/lys.pdb'); %pdb is a structure variable␣
↪containing PDB records
index_ca = selectname(pdb.name, 'CA');
```

`selectid()` returns a logical indexing which matches given IDs (integers). The following code returns logical indexing of atoms of the 1st and 2nd residue IDs.

```
index_resid = selectid(pdb.resseq, 1:2);
```

`selectrange()` returns a logical indexing of atoms within cutoff distance of given reference coordinate. The following code returns logical indexing of atoms within 8.0 Angstrom distance of the 1st and 2nd residue.

```
index_range = selectrange(crd, index_resid, 8.0);
```

As noted above, logical indexing can be combined to select subset on multiple conditions. For example, alpha-carbons of the 1st and 2nd residues are selected by

```
index = index_ca & index_resid;  % Boolean AND
```

Obtained logical indexings can be used with other MDToolbox functions, such as I/O functions. The following reads the trajectory of subset atoms specified by the logical index `index`:

```
trj = readdcd('run.dcd', index);
```

As an alternative way, users can directly choose subset from coordinate or trajectory variable. This can be done by using a utility function of MDToolbox `to3()`. `to3()` converts given logical indexing to XYZ-type logical indexing. For example, the following code extracts the subset trajectory as same as above.

```
trj_all = readdcd('run.dcd');
trj = trj_all(:, to3(index));
```

The following explains how `to3()` works by using simple indexing:

```
>> index = [true false true]

index =

     1     0     1

>> to3(index)

ans =

     1     1     1     0     0     0     1     1     1
```

CHAPTER 2

Examples

2.1 1D Umbrella Sampling of Tri-Alanine and WHAM

Files for this example can be downloaded from here. This example is located in `mdtoolbox_example/umbrella_alat/wham/`.

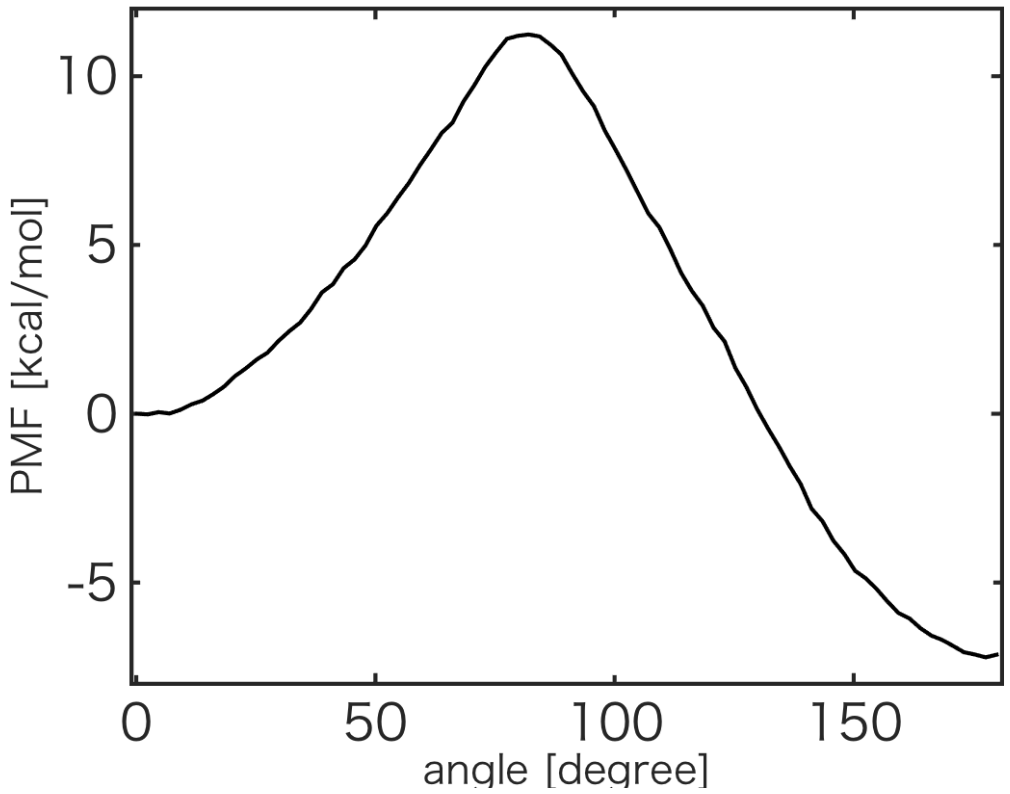
```
% this routine calculates the Potential of Mean Force (PMF) from
% umbrella sampling data by using WHAM

%% setup constants
C = getconstants();
KBT = C.KB*300; % KB is the Boltzmann constant in kcal/(mol K)

%% umbrella window centers
umbrella_center = 0:3:180;
K = numel(umbrella_center);

%% define edges for histogram bin
M = 80; % number of bins
edge = linspace(-1, 181, M+1);
bin_center = 0.5 * (edge(2:end) + edge(1:(end-1)));

%% read dihedral angle data
data_k = {};
for k = 1:K
  filename = sprintf('../3_prod/run_%d.dat', umbrella_center(k));
  x = load(filename);
  data_k{k} = x(:, 2);
end

%% calculate histogram (h_km)
% h_km: histogram (data counts) of k-th umbrella data counts in m-th data bin
h_km = zeros(K, M);
for k = 1:K
```

(continues on next page)

(continued from previous page)

```
  [~, h_m] = assign1dbin(data_k{k}, edge);
  h_km(k, :) = h_m;
end

%% bias-factor
% bias_km: bias-factor of k-th umbrella-window evaluated at m-th bin-center
bias_km = zeros(K, M);
for k = 1:K
  for m = 1:M
    spring_constant = 200 * (pi/180)^2; % conversion of the unit from kcal/mol/rad^2␣
↪to kcal/mol/deg^2
    bias_km(k, m) = (spring_constant./KBT)*(minimum_image(umbrella_center(k), bin_
↪center(m))).^2;
  end
end

%% WHAM
% calculate probabilities in the dihedral angle space, and evaluate the potential of␣
↪mean force (PMF)
[f_k, pmf] = wham(h_km, bias_km);
pmf = KBT*pmf;
pmf = pmf - pmf(1);

%% plot the PMF
hold off
plot(bin_center, pmf, 'k-');
formatplot
xlabel('angle [degree]', 'fontsize', 20);
ylabel('PMF [kcal/mol]', 'fontsize', 20);
axis([-1 181 -8 12]);
exportas('analyze')
hold off

%% save results
save analyze.mat;
```

```
function dx = minimum_image(center, x)
dx = x - center;
dx = dx - round(dx./360)*360;
```



2.2 1D Umbrella Sampling of Tri-Alanine and dTRAM

Files for this example can be downloaded from here. This example is located in `mdtoolbox_example/umbrella_alat/dtram/`.

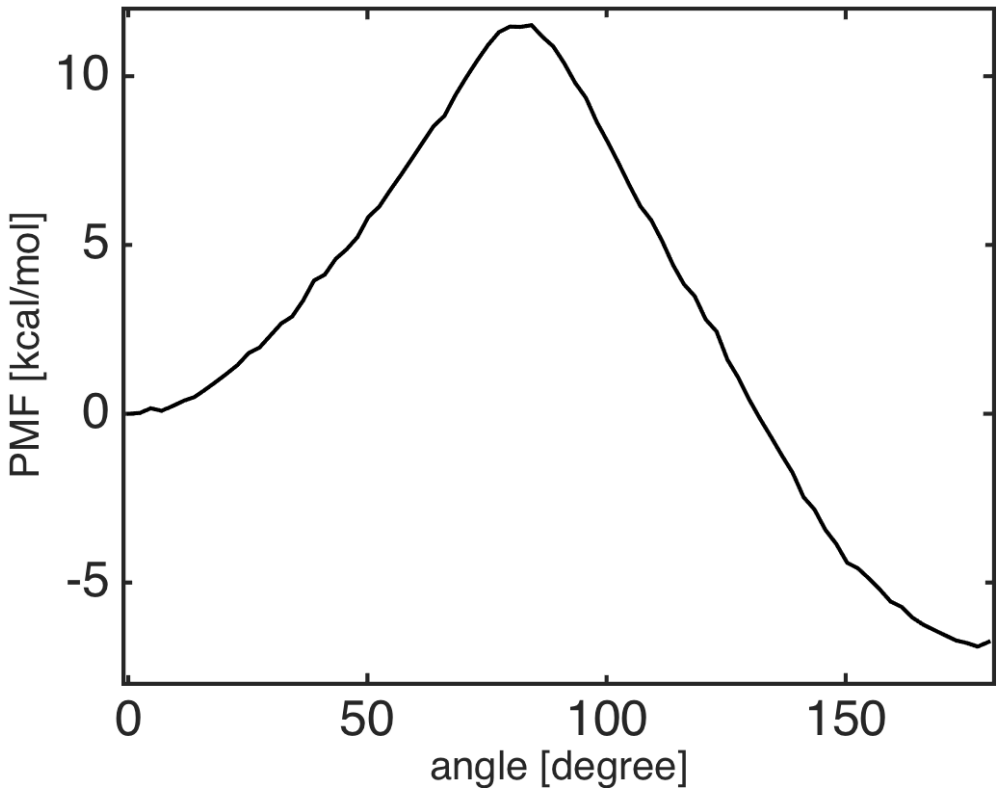
```
% this routine calculates the Potential of Mean Force (PMF) from umbrella sampling␣
↪data by using dTRAM

%% setup constants
C = getconstants();
KBT = C.KB*300; % KB is the Boltzmann constant in kcal/(mol K)

%% umbrella window centers
umbrella_center = 0:3:180;
K = numel(umbrella_center); % number of umbrella window

%% define edge for bin
M = 80; % number of Markov sates
edge = linspace(-1, 181, M+1);
bin_center = 0.5 * (edge(2:end) + edge(1:(end-1)));

%% bias-factor
% bias_km: bias-factor for k-th umbrella-window evaluated at m-th bin-center
bias_km = zeros(K, M);
for k = 1:K
  for m = 1:M
```

(continues on next page)

(continued from previous page)

```
    spring_constant = 200 * (pi/180)^2; % conversion of the unit from kcal/mol/rad^2␣
↪to kcal/mol/deg^2
    bias_km(k, m) = (spring_constant./KBT)*(minimum_image(umbrella_center(k), bin_
↪center(m))).^2;
  end
end

%% read dihedral angle data
data_k = {};
for k = 1:K
  filename = sprintf('../3_prod/run_%d.dat', umbrella_center(k));
  x = load(filename);
  data_k{k} = x(:, 2);
end

%% calculate count matrix for transition between bins
index_k = {};
for k = 1:K
  index_k{k} = assign1dbin(data_k{k}, edge);
end

c_k = {};
for k = 1:K
  c_k{k} = msmcountmatrix(index_k{k}, 1, M);
end

%% dTRAM
pmf = dtram(c_k, bias_km);
pmf = KBT*pmf;
pmf = pmf - pmf(1);

%% plot the PMF
hold off
plot(bin_center, pmf, 'k-');
formatplot
xlabel('angle [degree]', 'fontsize', 20);
ylabel('PMF [kcal/mol]', 'fontsize', 20);
axis([-1 181 -8 12]);
exportas('analyze')
hold off

%% save results
save analyze.mat;
```

```
function dx = minimum_image(center, x)
dx = x - center;
dx = dx - round(dx./360)*360;
```



2.3 1D Umbrella Sampling of Tri-Alanine and MBAR

Files for this example can be downloaded from here. This example is located in `mdtoolbox_example/umbrella_alat/mbar/`.

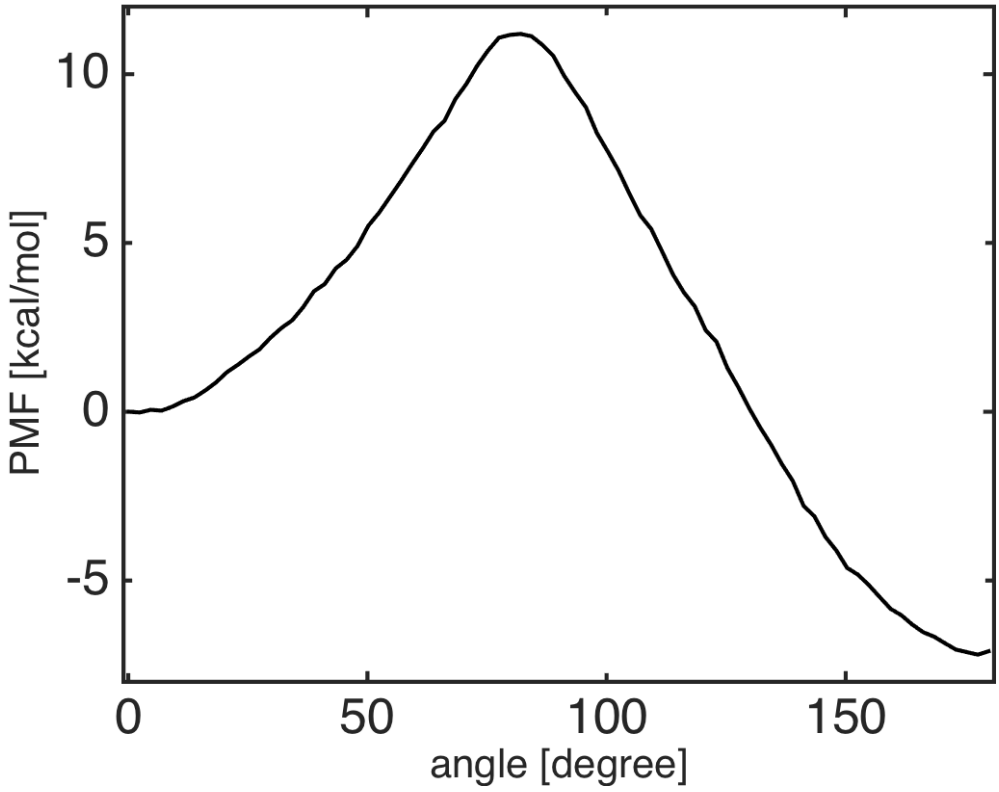
```
% this routine calculates the Potential of Mean Force (PMF) from
% umbrella sampling data by using MBAR

%% constants
C = getconstants();
KBT = C.KB*300; % KB is the Boltzmann constant in kcal/(mol K)

%% define umbrella window centers
umbrella_center = 0:3:180;
K = numel(umbrella_center);

%% define function handles of bias energy for umbrella windows
for k = 1:K
  spring_constant = 200 * (pi/180)^2; % conversion of the unit from kcal/mol/rad^2 to␣
↪kcal/mol/deg^2
  fhandle_k{k} = @(x) (spring_constant/KBT)*(minimum_image(x, umbrella_center(k))).^2;
end

%% read dihedral angle data
data_k = {};
for k = 1:K
```

(continues on next page)

(continued from previous page)

```
  filename = sprintf('../3_prod/run_%d.dat', umbrella_center(k));
  x = load(filename);
  data_k{k} = x(:, 2);
end

%% evaluate u_kl: reduced bias-factor or potential energy of umbrella simulation data␣
↪k evaluated by umbrella l
for k = 1:K
  for l = 1:K
    u_kl{k, l} = fhandle_k{l}(data_k{k});
  end
end

%% MBAR
% calculate free energies of umbrella systems
f_k = mbar(u_kl);

%% PMF
M = 80; % number of bins
edge = linspace(-1, 181, M+1);
bin_center = 0.5 * (edge(2:end) + edge(1:(end-1)));
for k = 1:K
  bin_k{k} = assign1dbin(data_k{k}, edge);
end
pmf = mbarpmf(u_kl, bin_k, f_k);
pmf = KBT*pmf;
pmf = pmf - pmf(1);

%% plot the PMF
hold off
plot(bin_center, pmf, 'k-');
formatplot
xlabel('angle [degree]', 'fontsize', 20);
ylabel('PMF [kcal/mol]', 'fontsize', 20);
axis([-1 181 -8 12]);
exportas('analyze')
hold off

%% save results
save analyze.mat;
```

```
function dx = minimum_image(center, x)
dx = x - center;
dx = dx - round(dx./360)*360;
```



2.4 Conventional MD of Alanine-Dipeptide and 2D PMF surface

Files for this example can be downloaded from here. This example is located in `mdtoolbox_example/md_alad/pmf/`.

We calculate the surface of potential of mean force (PMF) in a 2-dimensional dihedral angle space. Molecular dynamics trajectory of alanine-dipeptide solvated in explicit water models is used for the demonstration.

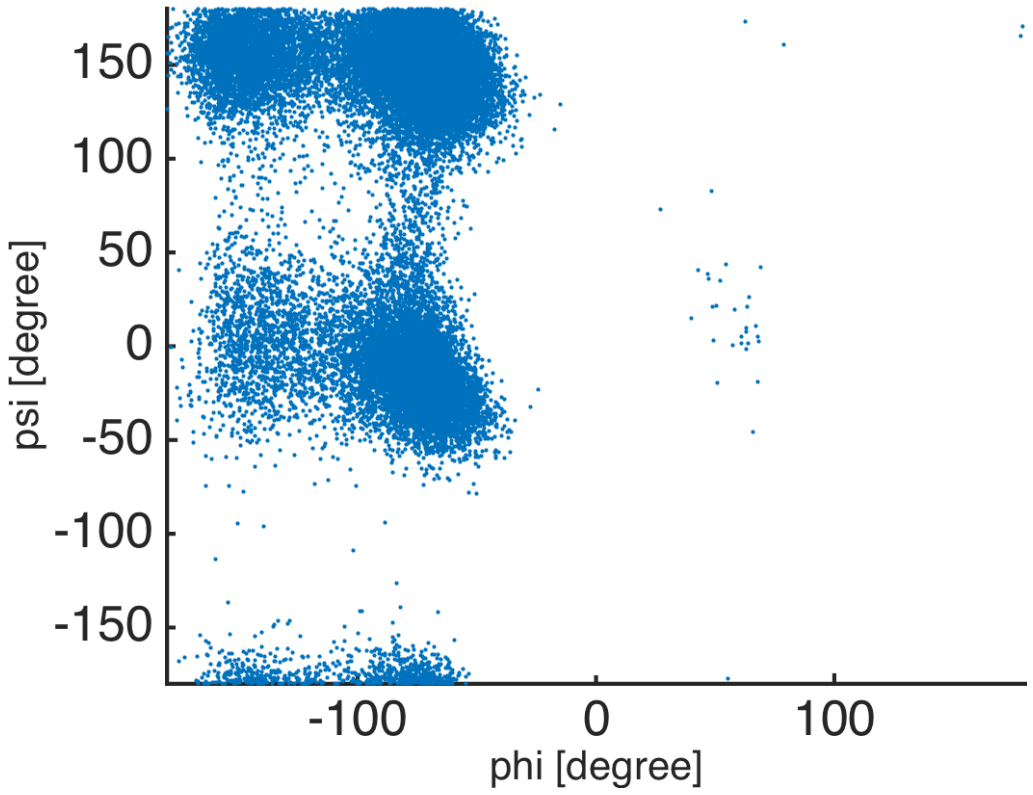
First, we extract dihedral angles from the trajectory:

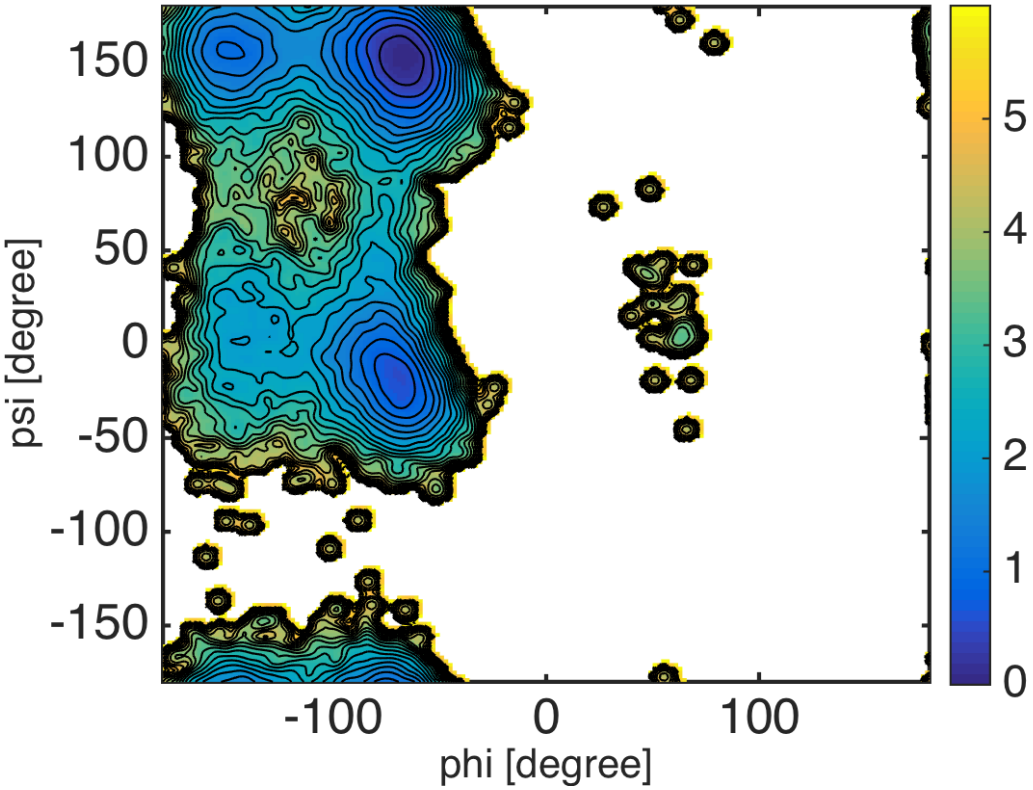
```
%% read data
trj = readnetcdf('../3_prod/run.nc');

%% define atom indices for dihedral angles
index_phi = [5 7 9 15];
index_psi = [7 9 15 17];

%% calculate dihedral angles
phi = calcdihedral(trj, index_phi);
psi = calcdihedral(trj, index_psi);

%% convert the unit from radian to degree
phi = phi.*180./pi;
psi = psi.*180./pi;
```

Next, the probability density function (PDF) in the 2-dimentional dihedral space is estimated from the scattered data (`phi` and `psi`). This can be done by using the bivariate kernel density estimation (`ksdensity2d.m`) which is called

in `calcpmf2d.m` routine.

```
%% scattered plot of the dihedral angles
scatter(phi, psi, 5, 'filled');
axis([-180 180 -180 180]); axis xy;
formatplot2
xlabel('phi [degree]', 'FontSize', 20, 'FontName', 'Helvetica');
ylabel('psi [degree]', 'FontSize', 20, 'FontName', 'Helvetica');
exportas('scatter');

%% calculate PMF and visualize the surface
xi = -180:2:180; % grids in x-axis
yi = -180:2:180; % grids in y-axis
pmf = calcpmf2d([phi psi], xi, yi, [3.0 3.0], [360 360]);
s   = getconstants(); % get Boltzmann constant in kcal/mol/K
T   = 300.0;          % set temperature
pmf = s.KB*T*pmf;     % convert unit from KBT to kcal/mol

%% visualization
landscape(xi, yi, pmf, 0:0.25:6); colorbar;
axis([-180 180 -180 180]);
xlabel('phi [degree]', 'FontSize', 20, 'FontName', 'Helvetica');
ylabel('psi [degree]', 'FontSize', 20, 'FontName', 'Helvetica');
exportas('pmf');
```





Note that the kernel density estimator tends to broaden the "true" PDF surface by a convolution with a Gaussian kernel. So, we should be careful especially when interested in small dips or barrier heights on the surface.

2.5 2D Umbrella Sampling of Alanine-Dipeptide and WHAM

Files for this example can be downloaded from here. This example is located in `mdtoolbox_example/umbrella_alad/wham/`.

```
% this routine calculates free energies of umbrella systems by using WHAM

%% setup constants
C = getconstants();
KBT = C.KB*300; % KB is the Boltzmann constant in kcal/(mol K)

%% umbrella window centers
center_phi = -180:15:-15;
center_psi = 0:15:165;
K = numel(center_phi)*numel(center_psi);

umbrella_center = zeros(K, 2);
k = 0;
for j = 1:numel(center_psi)
  for i = 1:numel(center_phi)
    k = k + 1;
    umbrella_center(k, :) = [center_phi(i) center_psi(j)];
  end
```

(continues on next page)

(continued from previous page)

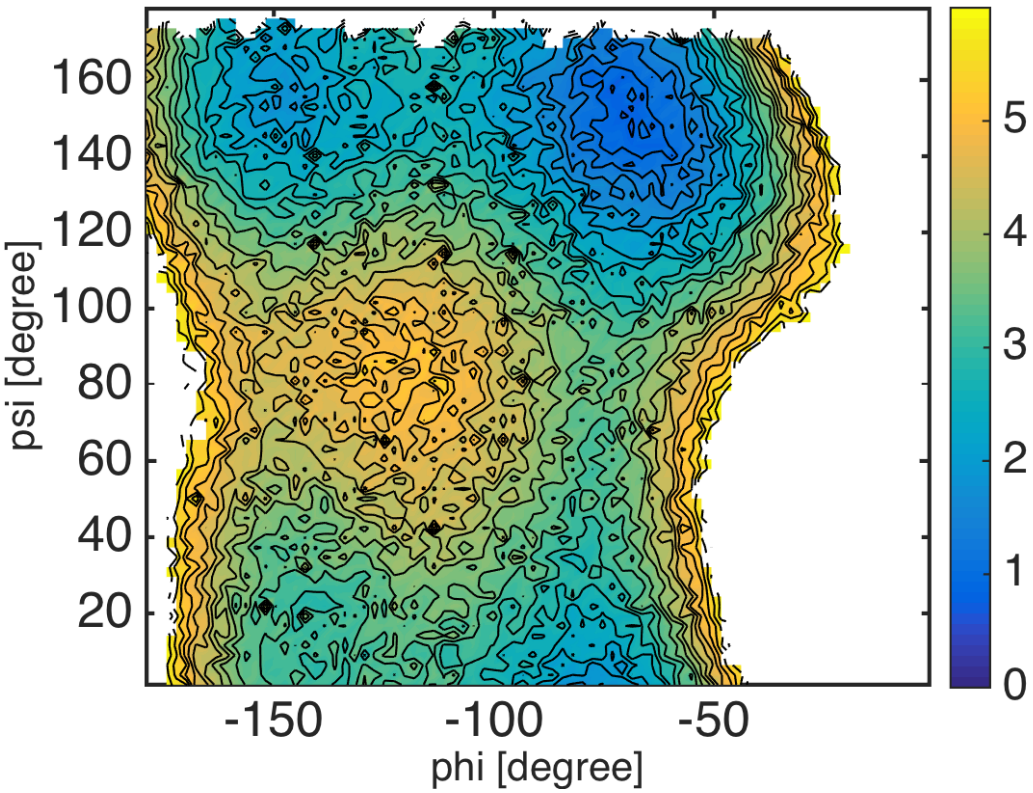
```
end

%% define edges for histogram bin
edge_phi = linspace(-180, 0, 81);
edge_psi = linspace(0, 180, 71);
bin_center_phi = 0.5 * (edge_phi(2:end) + edge_phi(1:(end-1)));
bin_center_psi = 0.5 * (edge_psi(2:end) + edge_psi(1:(end-1)));
M = numel(bin_center_phi)*numel(bin_center_psi);

bin_center = zeros(M, 2);
m = 0;
for j = 1:numel(bin_center_psi)
  for i = 1:numel(bin_center_phi)
    m = m + 1;
    bin_center(m, :) = [bin_center_phi(i) bin_center_psi(j)];
  end
end

%% read dihedral angle data
data_k = {};
for k = 1:K
  filename = sprintf('../4_prod/run_%d_%d.dat', umbrella_center(k, 1), umbrella_
↪center(k, 2));
  x = load(filename);
  data_k{k} = x(:, 2:3);
end

%% calculate histogram (h_km)
% h_km: histogram (data counts) of k-th umbrella data counts in m-th data bin
h_km = zeros(K, M);
for k = 1:K
  [~, histogram] = assign2dbin(data_k{k}, edge_phi, edge_psi);
  h_km(k, :) = histogram(:)';
end

%% bias-factor
% bias_km: bias-factor of k-th umbrella-window evaluated at m-th bin-center
bias_km = zeros(K, M);
for k = 1:K
  for m = 1:M
    spring_constant = 50 * (pi/180)^2; % conversion of the unit from kcal/mol/rad^2␣
↪to kcal/mol/deg^2
    bias_km(k, m) = (spring_constant./KBT)* sum(minimum_image(umbrella_center(k, :),␣
↪bin_center(m, :)).^2, 2);
  end
end

%% WHAM
% evaluate the potential of mean force (PMF) in dihedral angle space
[f_k, pmf] = wham(h_km, bias_km);
pmf = KBT*pmf;
pmf = pmf - min(pmf(:));

%% visualization
pmf2d = zeros(numel(bin_center_phi), numel(bin_center_psi));
pmf2d(:) = pmf(:);
landscape(bin_center_phi, bin_center_psi, pmf2d', 0:0.25:6); colorbar;
```

(continues on next page)

(continued from previous page)

```
xlabel('phi [degree]', 'FontSize', 20, 'FontName', 'Helvetica');
ylabel('psi [degree]', 'FontSize', 20, 'FontName', 'Helvetica');
exportas('analyze');

%% save results
save analyze.mat;
```

```
function dx = minimum_image(center, x)
dx = x - center;
dx = dx - round(dx./360)*360;
```



2.6 2D Umbrella Sampling of Alanine-Dipeptide and MBAR

Files for this example can be downloaded from here. This example is located in `mdtoolbox_example/umbrella_alad/mbar/`.

```
% this routine calculates free energies of umbrella systems by using MBAR

%% constants
C = getconstants();
KBT = C.KB*300; % KB is the Boltzmann constant in kcal/(mol K)

%% define umbrella window centers
center_phi = -180:15:-15;
```

(continues on next page)

(continued from previous page)

```
center_psi = 0:15:165;
K = numel(center_phi)*numel(center_psi);

umbrella_center = zeros(K, 2);
k = 0;
for i = 1:numel(center_phi)
  for j = 1:numel(center_psi)
    k = k + 1;
    umbrella_center(k, :) = [center_phi(i) center_psi(j)];
  end
end

%% read dihedral angle data
for k = 1:K
  filename = sprintf('../4_prod/run_%d_%d.dat', umbrella_center(k, 1), umbrella_
↪center(k, 2));
  x = load(filename);
  data_k{k} = x(:, 2:3);
end

%% evaluate u_kl: reduced bias-factor of umbrella simulation data k evaluated by␣
↪umbrella l
for k = 1:K
  for l = 1:K
    spring_constant = 50 * (pi/180)^2; % unit conversion from kcal/mol/rad^2 to kcal/
↪mol/deg^2
    u_kl{k, l} = (spring_constant/KBT)*sum(minimum_image(umbrella_center(l, :), data_k
↪{k}).^2, 2);
  end
end

%% MBAR: calculate free energies of umbrella systems
f_k = mbar(u_kl);

%% save results
save calc_mbar.mat;
```

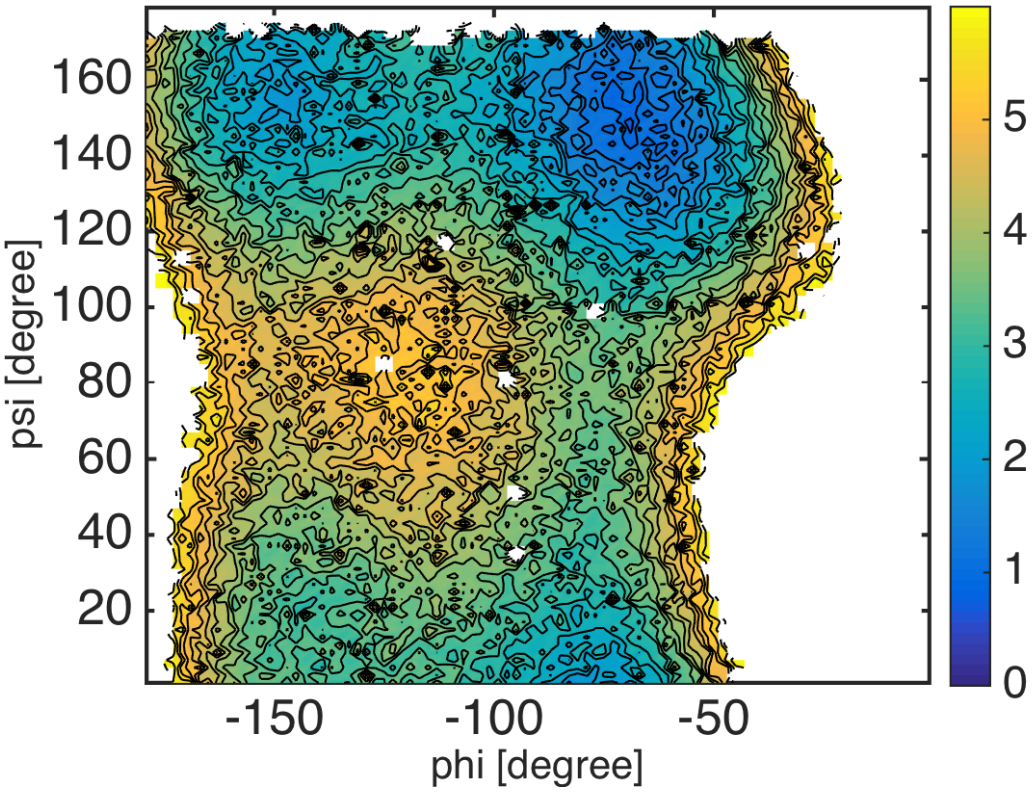
```
function dx = minimum_image(center, x)
dx = x - center;
dx = dx - round(dx./360)*360;
```

```
% this routine calculates 2-D potential of mean force (PMF) from the result of MBAR

%% read MBAR result
load calc_mbar.mat K data_k u_kl f_k KBT;

%% calculate PMF by counting weights of bins under restraint-free condition
% assign 1-dimensional bin index to 2-dimensional data
M_phi = 90;
M_psi = 90;
edge_phi = linspace(-180, 0, M_phi+1);
edge_psi = linspace(0, 180, M_psi+1);
center_phi = 0.5 * (edge_phi(2:end) + edge_phi(1:(end-1)));
center_psi = 0.5 * (edge_psi(2:end) + edge_psi(1:(end-1)));
for k = 1:K
  bin_phi = assign1dbin(data_k{k}(:, 1), edge_phi);
```

(continues on next page)

(continued from previous page)

```
  bin_psi = assign1dbin(data_k{k}(:, 2), edge_psi);
  bin_k{k} = M_psi*(bin_phi-1) + bin_psi;
end

% evaluate PMF of bins
pmf = mbarpmf(u_kl, bin_k, f_k);

% reshape PMF data
pmf2 = zeros(M_phi*M_psi, 1);
pmf2(:) = NaN;
pmf2(1:numel(pmf)) = pmf;
pmf2 = KBT*pmf2; % convert unit from KBT to kcal/mol
pmf2 = pmf2 - min(pmf2(:));
pmf = reshape(pmf2, M_psi, M_phi);

%% visualization
landscape(center_phi, center_psi, pmf, 0:0.25:6); colorbar;
xlabel('phi [degree]', 'FontSize', 20, 'FontName', 'Helvetica');
ylabel('psi [degree]', 'FontSize', 20, 'FontName', 'Helvetica');
exportas('pmf_histogram');
```

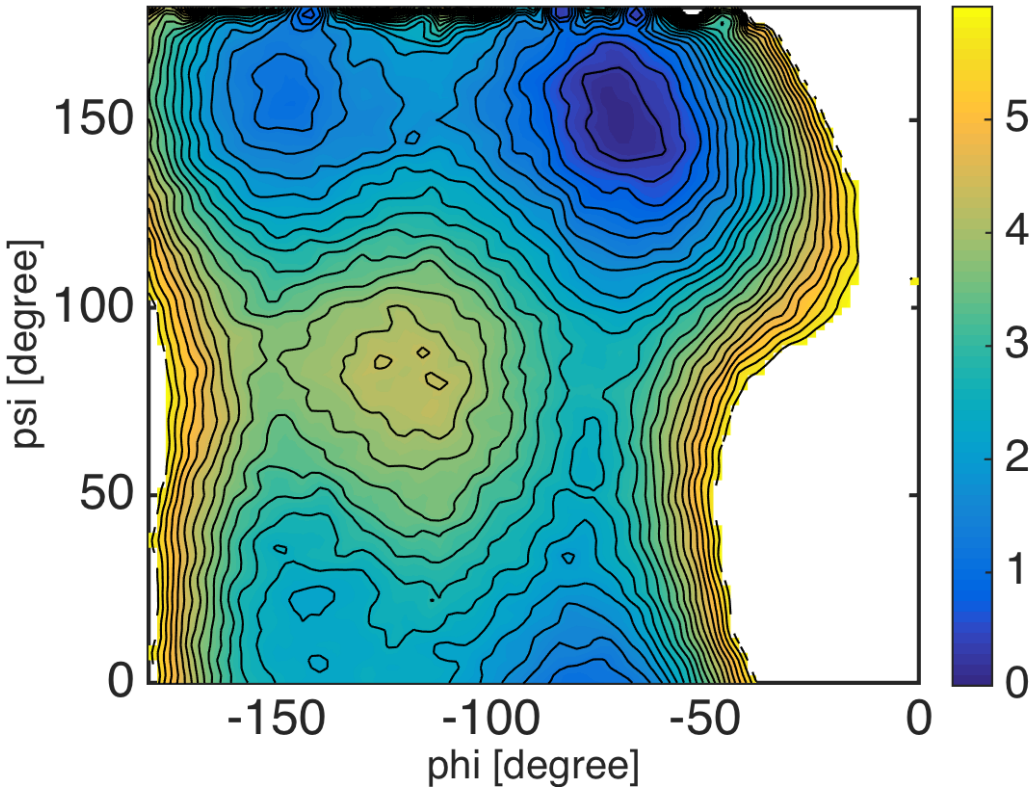



```
% this routine calculates 2-D potential of mean force (PMF) from the result of MBAR

%% read MBAR result
load calc_mbar.mat K data_k u_kl f_k KBT;

%% evaluate weights of data under restraint-free condition
```

(continues on next page)

(continued from previous page)

```
[~, w_k] = mbarpmf(u_kl, [], f_k);

%% calculate PMF by using kernel density estimation
% collect scattered data with weights
data = [];
for k = 1:K
  data = [data; data_k{k}];
end

weight = [];
for k = 1:K
  weight = [weight; w_k{k}];
end

% evaluate PMF by using a kernel density estimator
center_phi = -180:2.0:0;
center_psi = 0:2.0:180;
pmf = calcpmf2d(data, center_phi, center_psi, [2.0 2.0], [360 360], weight);
pmf = pmf*KBT; % convert unit from KBT to kcal/mol

%% visualization
landscape(center_phi, center_psi, pmf, 0:0.25:6); colorbar;
xlabel('phi [degree]', 'FontSize', 20, 'FontName', 'Helvetica');
ylabel('psi [degree]', 'FontSize', 20, 'FontName', 'Helvetica');
exportas('pmf_ksdensity');
```



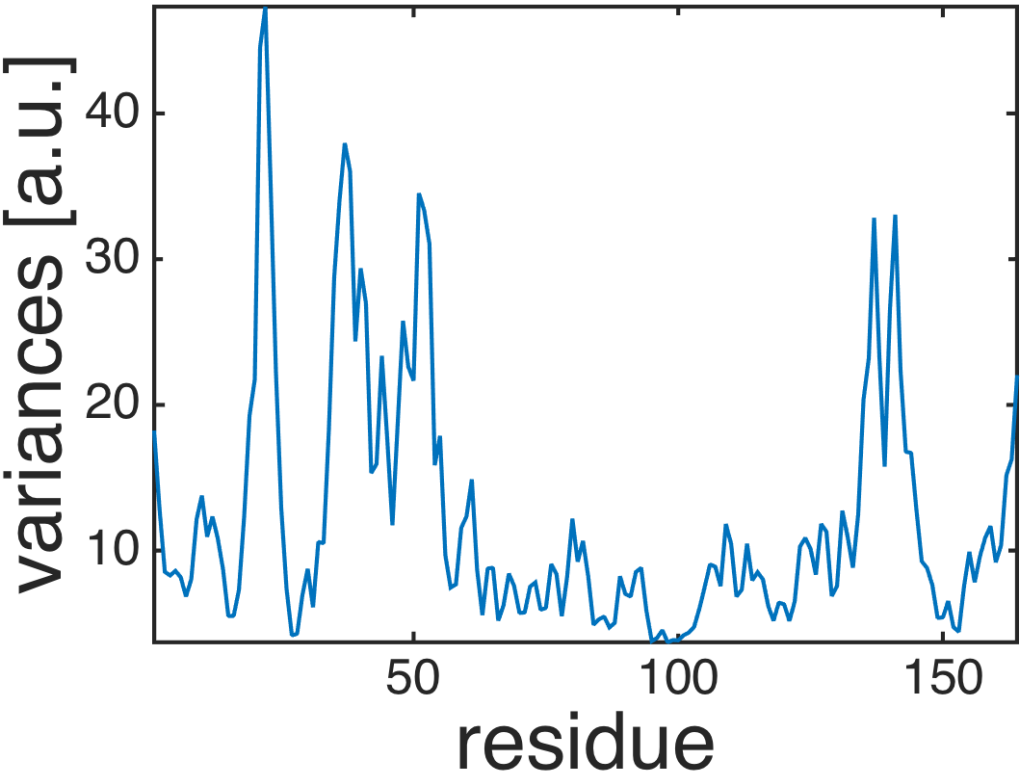
2.7 Anisotropic Network Model

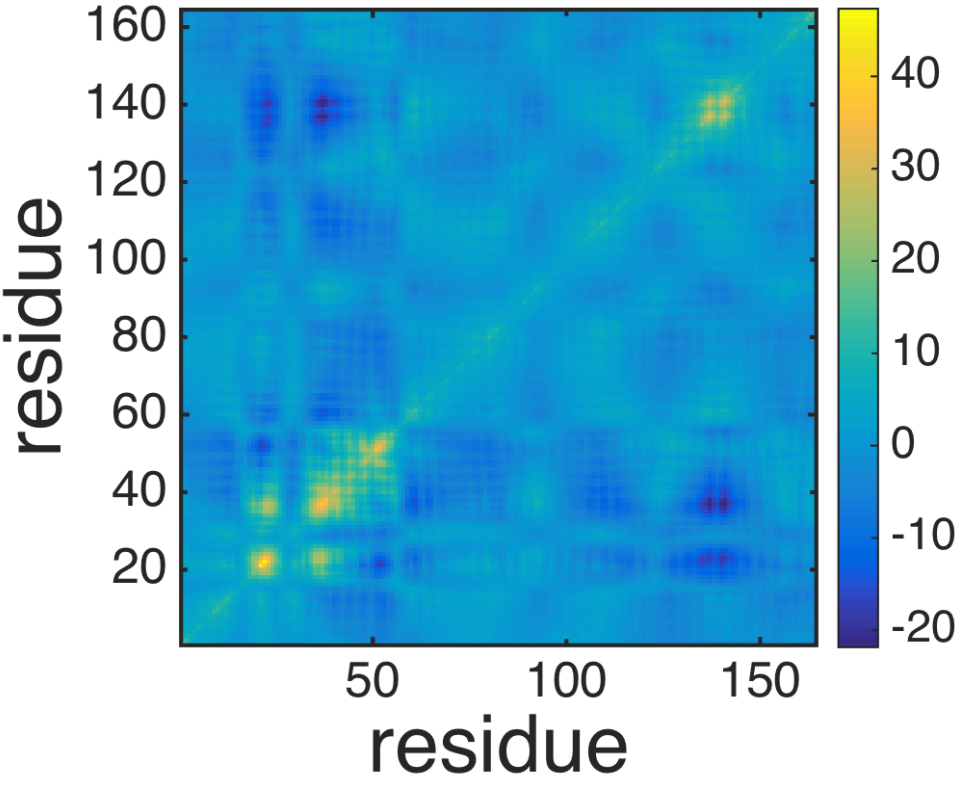
Files for this example can be downloaded from here. This example is located in `mdtoolbox_example/anm_lys/` .

2.7.1 Normal mode analysis of ANM

Normal mode analysis of Ca-based anisotropic network model of T4 lysozyme (`script_anm.m`).

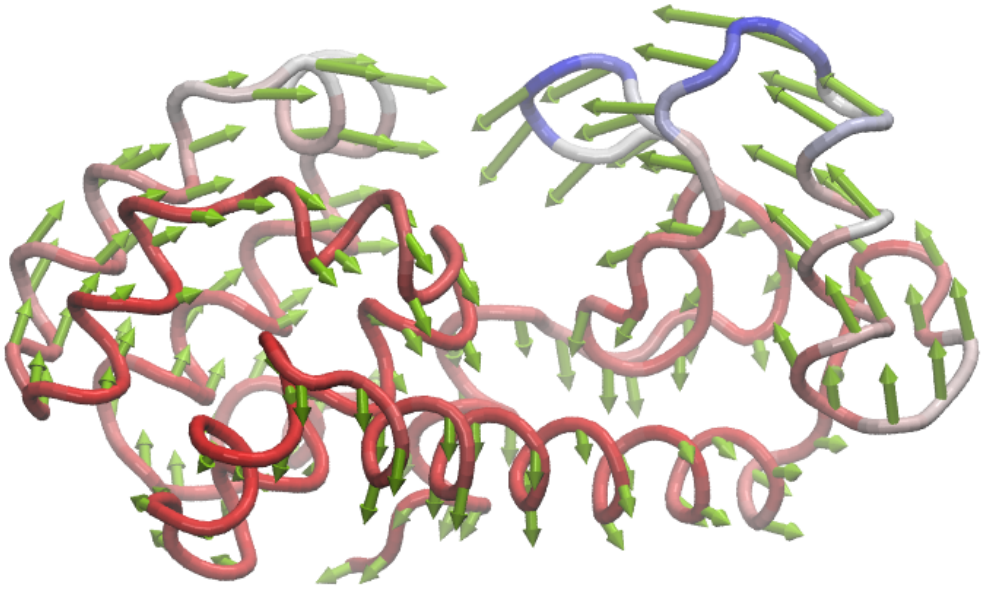
```
%% Nomal mode analysis of anisotropic network model of T4 lysozyme

% read Ca coordinates from PDB file
[pdb, crd] = readpdb('lys.pdb');
index_ca = selectname(pdb.name, 'CA');
crd = crd(to3(index_ca));
crd = decenter(crd);

% normal mode of anisotropic network model (ANM)
[emode, frequency, covar, covar_atom] = anm(crd, 8.0);

% plot root-mean-square-fluctuations (RMSF)
plot(diag(covar_atom));
axis tight;
xlabel('residue','fontsize',40);
ylabel('variances [a.u.]','fontsize',40);
formatplot
exportas('rmsf');

% plot covariance
imagesc(covar_atom);
axis xy; axis square;
xlabel('residue','fontsize',40);
ylabel('residue','fontsize',40);
colorbar;
formatplot2
exportas('covar_atom');

% export NMD file for visalizing mode structures
writenmd('anm.nmd', crd, emode);

% save data
save script_anm.mat;
```


variances [a.u.]
10
20
30
40
50
100
150
residue


residue
160
140
120
100
80
60
40
20
50
100
150
residue
40
30
20
10
0
-10
-20

Visualize mode structures by using the Normal mode wizard in VMD.

```
$ vmd
vmd > nmwiz load anm.nmd
```



2.7.2 Transformation of frame

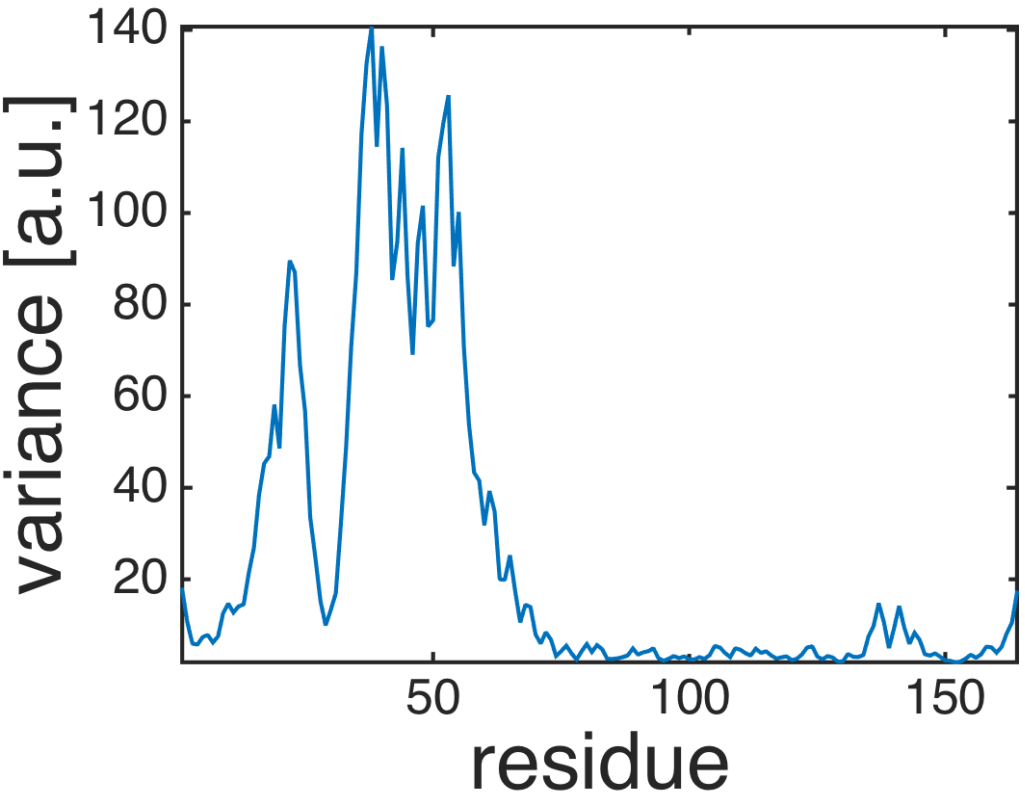
Transform from the Eckart frame to a non-Eckart frame (`script_transformframe.m`).

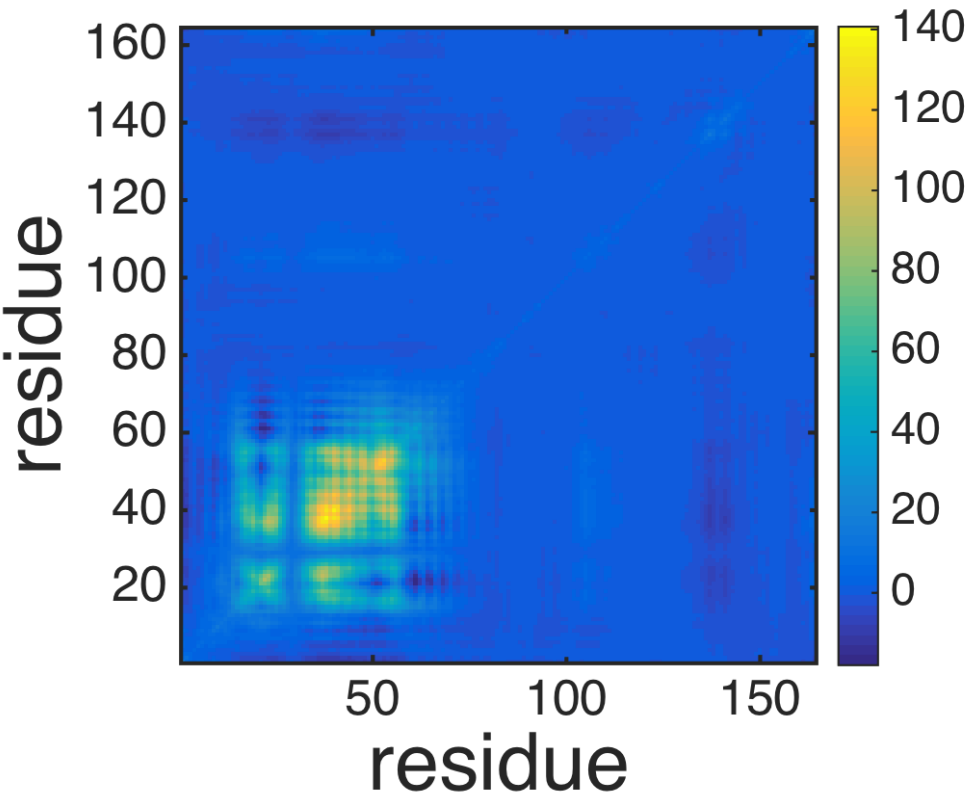
```
%% Transform from the Eckart frame to a non-Eckart frame.

% load data
load script_anm.mat;

% transform frame
index_fixeddomain = [1:11 77:164]; %atom-index for the larger domain
external_mode = emode(:,(end-5):end);
[emode2, variances2, covar2, covar2_atom] = transformframe(index_fixeddomain,␣
↪external_mode, covar);

% plot root-mean-square-fluctuations (RMSF)
plot(diag(covar2_atom));
axis tight;
xlabel('residue','FontSize',40);
ylabel('variance [a.u.]','FontSize',40);
formatplot
exportas('rmsf_ne');
```

(continues on next page)

(continued from previous page)

```
% plot covariance
imagesc(covar2_atom);
axis xy; axis square;
xlabel('residue','FontSize',40);
ylabel('residue','FontSize',40);
colorbar;
formatplot2;
exportas('covar_atom_ne');

% export PDB files for visalizing mode structures
writenmd('anm_ne.nmd', crd, emode2);

% save data
save script_transformframe.mat;
```





Visualize mode structures by using the Normal mode wizard in VMD.

```
$ vmd
vmd > nmwiz load anm_ne.nmd
```