INDUS
User Manual

Sean M. Marks
(semarks@seas.upenn.edu)
Amish J. Patel*
(amish.patel@seas.upenn.edu)

Patel Group
University of Pennsylvania

October 5, 2020

Contents

1 Introduction

The INDUS code described here computes the number of particles, N_v, and the coarse-grained number of particles, $\tilde{N}_v$, in probe volumes, v, of several different geometries. It is primarily intended to be used as an extension to PLUMED, a popular plugin for molecular dynamics (MD) engines such as GROMACS, LAMMPS, AMBER, and NAMD. As part of PLUMED, the INDUS code can be used to post-process trajectories stored in a variety of formats (see `driver`). If PLUMED+INDUS is subsequently patched into an MD engine, it can be used to perform biased simulations with $\tilde{N}_v$.

Note that INDUS can also be compiled as a standalone code. However, this is intended primarily for development purposes.

For more on the INDUS method in theory and practice, here is a (non-exhaustive) list of topics and relevant references:

- INDUS for beginners [1]
- INDUS in probe volumes of all shapes and sizes [2]
- INDUS for biological systems, especially proteins [3, 4]
- INDUS and the sparse sampling method, an alternative to traditional umbrella sampling for efficiently computing $F_v(\tilde{N})$ [5, 6]

1.1 Units

The INDUS code uses the same basic set of units as PLUMED, which also happens to be the same as GROMACS.

Quantity	**Units**
Length	nm = 10^{-9} m = 10 Å
Time	ps = 10^{-12} s
Mass	amu
Energy	kJ/mol

2 Getting the Code

The code is publicly available on GitHub (link).

3 Installation

The following section details the ways in which INDUS can be installed.

3.1 Patching INDUS into PLUMED

It is recommended that you make use of the INDUS code as an extension to PLUMED. Make sure that you perform the following steps **before** configuring and compiling PLUMED!

A shell script to aid you in the patching process is available in the INDUS code repository under `indus/plumed_patch/patch_plumed.sh`. When you run the script, simply pass to it the location of the root directory of the PLUMED repository (*i.e.* the one that contains the `Makefile` and so on):

```
./patch_plumed.sh <location-of-PLUMED-source-code>
```

Then simply configure, compile, and install PLUMED normally. For instructions on how to install PLUMED, see here. For those who are already using PLUMED, I recommend making a separate installation for PLUMED+INDUS.

Once installation is complete, a new action, `INDUS`, will be available to you the next time you run PLUMED. You can check that the procedure was successful by running the following command:

```
plumed --no-mpi manual --vim --action INDUS
```

This will print out a list of all the actions and command-line tools registered in PLUMED, followed by options which are specific to the `INDUS` action. (*Note*: The `-no-mpi` option flag is to prevent PLUMED from looking for an MPI library, since none is required for this check. The `-vim` flag causes output to be printed in a console-friendly format.)

3.2 Patching PLUMED+INDUS into an MD Code

Once you have installed PLUMED+INDUS, follow the instructions here to add it to your MD engine of choice. Note that a given version of PLUMED only supports certain versions of each MD code. To check which MD codes and versions are supported by each version of PLUMED, check out the online manual (link).

3.3 Installing INDUS as a Standalone Program

The INDUS code can be compiled as a standalone program that can analyze XTC files. Before proceeding, you will need to install the `xdrfile` library written by the GROMACS development team (see here). Make note of where you install it: you will need this information later.

To install INDUS as a standalone program, you will need to use CMake (download). The minimum version currently required is 3.15. Note that while it should be possible to use CMake to compile INDUS on Windows, I have only tested the procedure on MacOS and Ubuntu.

Below is a Bash script that demonstrates the process of configuring and installing standalone INDUS on a generic Unix-based system.

```
#!/bin/bash -eE

# This variable indicates where you installed xdrfile (i.e. the directory containing
# the folders bin, include, and lib).
XDR_DIR=<your-xdrfile-here>

```

```
# Put here the location where the driver should be installed
install_dir="${HOME}/programs/indus"

# If you aren't using a compiler that supports MPI, change the following lines
# - MacOS: CC=clang, CXX=clang++
# - Linux: CC=gcc,   CXX=g++
CC=mpicc
CXX=mpic++

# Create a build directory
mkdir -p build
cd build

# Configure INDUS. If you don't want to (or can't) compile with MPI or OpenMP support,
# simply change the corresponding options below from ON to OFF.
cmake .. \
  -DCMAKE_INSTALL_PREFIX="$install_dir" \
  -DCMAKE_BUILD_TYPE="RelWithDebInfo"   \
  -DXDRFILE_DIR="$XDRFILE_DIR"          \
  -DMPI_ENABLED=ON      \
  -DOPENMP_ENABLED=ON   \
  -DGPTL_ENABLED=OFF

# Compile
make -j

# Run tests
make test

# Install
make install
```

4 Using the Code

This section discusses how to use the INDUS code as part of PLUMED, and as a standalone code. Section 5 will discuss the details of the INDUS input file (`indus.input`). Note that this file is separate from the normal PLUMED input file (`plumed.dat`) and uses a different syntax. If you are not familiar with the input syntax for PLUMED, please consult the online manual for your version of choice (link).

4.1 As Part of PLUMED

Below is a sample PLUMED input file. Observe that the INDUS input file, `indus.input`, is an input to the `INDUS` action.

```
# plumed.dat - PLUMED input file

# Create an instance of the INDUS action, which governs the interface between
# PLUMED and the INDUS code
indus: INDUS INPUTFILE=indus.input

# Put a harmonic bias on Ntilde_v of the form U = KAPPA/2*(Ntilde_v - AT)^2,
# where KAPPA=0.5 kJ/mol and AT=3
restraint: RESTRAINT ARG=indus.ntilde KAPPA=0.5 AT=3.0

# Print some important quantities
# - indus.n        = N_v
```

```
# - indus.ntilde    = Ntilde_v
# - restraint.bias = value of the biasing potential
# - STRIDE=500      = print every 500 steps (i.e. every 1 ps when dt=0.002ps)
PRINT ...
  LABEL=print
  ARG=indus.n,indus.ntilde,restraint.bias
  STRIDE=500
  FILE=plumed.out
... PRINT
```

4.2 As a Standalone Program (XTC files only)

When compiled as a standalone program, the INDUS code can be used to analyze XTC files. Invoke from the command line using the following syntax:

```
./indus <indus.input>
```

See Section 5.5 for input options particular to running in standalone mode.

4.3 Limitations of the Code and Warnings to Users

1. Only orthorhombic boxes are supported (i.e. rectangular box where the side lengths may differ from each other)
2. Periodic boundary conditions (PBCs) in all directions are assumed.
3. Unless specified otherwise, probe volumes should not cross the periodic boundaries.
4. Only one probe volume may be specified per instance of INDUS.
5. The only indicator/switching functions supported for INDUS are the ones outlined in refs. [1, 2]. Unlike most of PLUMED's switching functions, these have continuous first derivatives and therefore produce smooth forces.
6. You should be able to pass `indus.ntilde` and `indus.ubias` to any other PLUMED action with no problems. For example, it should be possible to perform metadynamics with $\tilde{N}_v$. However, I have not tested this aside from passing `indus.ntilde` and `indus.ubias` to `RESTRAINT`.
7. I used PLUMED 2.4.0 and GROMACS 2016.3 when writing and testing the code. Any other versions of PLUMED that are 2.2 or higher should work just fine, but I have not tested these and cannot guarantee that they will work. The program should run fine with any simulation package that your favorite PLUMED version supports, but—again—I haven't tested it myself.

5 Input Options

This section discusses the syntax of the INDUS input file, which is typically named `indus.input`. For more on the input syntax for PLUMED, consult the online manual (link).

Note: In the following section, angle brackets (<>) are used to indicate places where a value should be placed. The angle brackets themselves are not part of the actual input syntax. Vertical lines (|) are used to enumerate options when only a limited number of values are permitted. For example, `axis = <x|y|z>` indicates that you must choose `axis` to be either `x`, `y`, or `z`.

5.1 Input File Structure

The INDUS input file is organized into key-item pairs, where the *item* can be one of the following:

1. a *value*: a string or number
2. a *vector*: a sequence of values enclosed by square brackets, [ ]. Sometimes I will also use the term *array*, which is simply a vector that has a fixed number of values.
3. a *block*: an arbitrary grouping of values, vectors, and possibly other blocks, all enclosed by curly braces, . A block functions like an object (in the sense of object-oriented programming) and all of the entries within its curly braces are said to be in its *scope*.

It is instructive to consider an example:

```
MyBlock = {
  # This is a comment
  some_value  = 2.718
  some_vector = [ 1.0  2.0  3.0 ]   # This is a comment at the end of a line

  # Presumably, MyBlock corresponds to an object in the code that owns
  # another object of type NestedBlock
  NestedBlock = {
    another_value = 3.14159
    AnotherBlock = { another_vector = [ we are values ] }
  }
}
```

Observe that each key is separated from its corresponding item by an equals sign (`=`). Comments are indicated by the hash character (`#`). Each token in the input file (e.g. keys, values, equals signs, brackets, braces, comment characters, etc.) must be separated from the others by whitespace (single/multiple spaces or a new line).

Aside from the aforementioned restrictions, the input syntax is quite flexible. Indentation is not necessary, and braces/brackets/equals signs need not be on the same line as any other token. For example, the following is an eyesore but perfectly legal:

```
MyBlock
= { some_vector = [
1.0  2.0
3.0 ] }
```

Each key-item pair exists within a scope. The highest scope in the input file is the *file scope*. This encompasses all the values, vectors, and blocks that are not enclosed in other blocks. For instance, `MyBlock` in the previous examples is a block at file scope. In contrast, `NestedBlock` and `some_vector` are within the scope of `MyBlock`. In this way of thinking, the input file *itself* is like an object at the top of the hierarchy.

5.2 Target Atom Selection

Target atoms are indicated by the key `Target` at file scope. This is a unique key, *i.e.* multiple target selections are not permitted.

5.2.1 By Index

Atoms are selected as ranges of indices of the form `<start>-<stop>:<stride>`, the same as in PLUMED. This selects every `<stride>`th atom from index `start` to `stop`. All indices start at 1. if `:<stride>` is omitted, the stride is assumed to be 1.

Here are some examples.

```
# Select every 4th atom from index 1 to index 16500
Target = [ atom_index 1-16500:4 ]

# Select every atom from index 500 to index 1000
Target = [ atom_index 500-1000 ]

# Ranges can be combined using a comma (no spaces)
Target = [ atom_index  1-16500:4,32000-64000:3 ]
```

5.3 Probe Volume Geometries

Probe volumes are defined at global scope using the keyword `ProbeVolume`. Only one probe volume geometry may be specified per instance of INDUS. To define multiple probe volumes using PLUMED, one must create multiple instances of the `INDUS` action.

Each probe volume definition includes two coarse-graining parameters: σ (`sigma`) and α_c (`alpha_c`). These determine how closely the coarse-grained number of particles, $\tilde{N}$, is correlated with the integer number of particles, N. For a detailed explanation, see refs. [1] and [3]. In short, choosing smaller values leads to tighter correlation between $\tilde{N}$ and N, but larger forces. This can cause instability and simulation crashes if σ and/or α_c are chosen too small.

We recommend the values $\sigma = 0.01$ nm and $\alpha_c = 2\sigma = 0.02$ nm for simulations of water; these are the defaults if `sigma` and `alpha_c` are omitted from the `ProbeVolume` definition. In practice, we have found that these values lead to stable simulations with tight correlation between N and $\tilde{N}$, to the extent that we consider the two essentially interchangeable.

The following subsections describe the syntax for the probe volume geometries currently supported.

5.3.1 Sphere

```
ProbeVolume = {
  type   = sphere
  r_max  = <r>
  center = [ <x> <y> <z> ]
  # Coarse-graining
  sigma    = <σ>
  alpha_c = <α_c>
}
```

5.3.2 Box

```
ProbeVolume = {
  type   = box
  # Boundaries of box along each axis
  x_range = [ <x_min> <x_max> ]
  y_range = [ <y_min> <y_max> ]
  z_range = [ <z_min> <z_max> ]
  # Coarse-graining
  sigma    = <σ>
  alpha_c = <α_c>
}
```

5.3.3 Cylinder

```
ProbeVolume = {
  type   = cylinder
  # Cylinder will have its axis parallel to this axis
  # - Default: parallel to z-axis
  axis    = <x|y|z>
  # Location of the center of the base
  base    = [ <x> <y> <z> ]
  radius  = <r>
  height  = <h>
  # Coarse-graining
  sigma   = <σ>
  alpha_c = <α_c>
}
```

5.4 Biasing $\tilde{N}_v$ Using the INDUS Code's Potential

Recall that PLUMED's `RESTRAINT` action can be used to set the form of the biasing potential on $\tilde{N}_v$. As an alternative, the INDUS code makes available makes available its own biasing potential of the following general form:

$$\mathcal{U}_{\text{bias}}(x) = \mathcal{U}_{\text{harmonic}}(x) + \mathcal{U}_{\text{linear}}(x) + \mathcal{U}_{\text{left harmonic}}(x) + \mathcal{U}_{\text{right harmonic}}(x) \tag{1}$$

$$= \frac{1}{2}\kappa(x - x^*)^2 + (\phi x + c) + \frac{1}{2}k_{\text{left}}(x - x_{\text{left}})^2\Theta(x_{\text{left}} - x) + \frac{1}{2}k_{\text{right}}(x - x_{\text{right}})^2\Theta(x - x_{\text{right}}) \tag{2}$$

where $\Theta(x)$ is the unit step function, which is zero for $x < 0$ and one for $x \geq 1$. Note that PLUMED supports similar functionality with a combination of the `RESTRAINT`, `LOWERWALLS`, and `UPPERWALLS` actions.

The corresponding syntax for the INDUS input file is:

```
Bias = {
  # Tells INDUS that x=Ntilde_v
  order_parameter = ntilde

  # Harmonic
  x_star = <x*>
  kappa  = <κ>

  # Linear
  phi      = <φ>
  constant = <c>

  # Left one-sided harmonic
  x_left = <x_left>
  k_left = <k_left>

  # Right one-sided harmonic
  x_right = <x_right>
  k_right = <k_right>
}
```

Any parameters not specified in the input file are set to zero automatically.

Then, in the PLUMED input file, `RESTRAINT` is called with `ARG`$=\mathcal{U}_{\text{bias}}(\tilde{N}_v)$. This is done to ensure that the biasing forces are correctly passed to the PLUMED core.

```
restraint: RESTRAINT ARG=indus.ubias SLOPE=1.0 AT=0.0 KAPPA=0.0
```

5.5 Standalone Mode Options

The following options are supported at file scope.

```
# The XTC file to analyze
XtcFile = <path_to_file>

# Range of times for which calculations are performed (in ps)
t0 = <t_0>
tf = <t_f>
```

All of these options are ignored when running the INDUS code as part of PLUMED.

6 Examples

6.1 Post-Processing a Trajectory with PLUMED's `driver`

Suppose that we have run simulation (biased or unbiased) and would now like to compute N and $\tilde{N}_v$ for each frame of the trajectory, which is saved as an XTC file named `traj_comp.xtc`. The probe volume is a cylinder with the center of its base at $\mathbf{r}_0 = (2.0, 2.0, 2.0)$ nm with radius $r = 1.0$ nm and height $h = 0.5$ nm. The cylinder is parallel to the y-axis and extends from $y_{\text{low}} = 2.0$ nm to $y_{\text{high}} = y_{\text{low}} + h = 2.5$ nm. The target atoms have indices 1, 4, 7, ... , 16498 (i.e. every 3 atoms from 1 to 16,500, indexed from 1). The coarse-graining parameters are $\sigma = 0.01$ nm and $\alpha_c = 2\sigma = 0.02$ nm.

The INDUS input file will be:

```
# indus.input

# Atoms targeted by INDUS
Target = [ atom_index 1-16500:3 ]

# Probe volume
ProbeVolume = {
  type   = cylinder
  axis   = y
  # Dimensions and location
  base   = [ 2.0 2.0 2.0 ]
  r_max  = 1.0
  height = 0.5
  # Coarse-graining
  sigma   = 0.01
  alpha_c = 0.02
}
```

The corresponding PLUMED input file is:

```
# plumed.dat

# Create an instance of the INDUS action, which computes N_v and Ntilde_v
indus: INDUS INPUTFILE=indus.input

# Print N_v and Ntilde_v for each frame to a file named 'plumed.out'
```

```
PRINT ...
  LABEL=print
  ARG=indus.n,indus.ntilde
  FILE=plumed.out
... PRINT
```

PLUMED's `driver` can be invoked from the command line as follows:

```
plumed driver --plumed "plumed.dat" --ixtc "traj_comp.xtc"
```

For more `driver` options—including how to post-process trajectories in other file formats—please consult the PLUMED manual.

6.2 Running a Biased Simulation with PLUMED

Suppose we want to run a biased simulation with a linear umbrella sampling potential on $\tilde{N}_v$. The probe volume and target atoms are the same as in the last example. Use PLUMED's `RESTRAINT` action to apply a bias of the form $\mathcal{U}_{\text{bias}} = \phi\tilde{N}$ with $\phi = 0.6$ kJ/mol.

The INDUS input file is unchanged from before. The PLUMED input now contains an instance of `RESTRAINT`, as follows:

```
# plumed.dat

# Create an instance of the INDUS action, which computes N_v and Ntilde_v
indus: INDUS INPUTFILE=indus.input

# Put a linear bias on Ntilde_v
restraint: RESTRAINT ARG=indus.ntilde SLOPE=0.6 AT=0.0

# Print N_v, Ntilde_v, and U_bias every 500 steps
PRINT ...
  LABEL=print
  ARG=indus.n,indus.ntilde,restraint.bias
  STRIDE=500
  FILE=plumed.out
... PRINT
```

The syntax for passing this PLUMED input file to the MD engine will depend on your choice of MD engine. Please consult the PLUMED manual for details (link).

6.3 Standalone Mode

Post-process a trajectory stored as an XTC file named `my_trajectory.xtc`. Compute N_v and $\tilde{N}_v$ in a spherical probe volume centered at $\mathbf{r}_0 = (2.5, 2.5, 2.5)$ nm and with radius $r = 0.6$ nm. The target atoms have indices 1, 5, 9, ... , 16497 (i.e. every 4 atoms from 1 to 16,500, indexed from 1). The coarse-graining parameters are $\sigma = 0.01$ nm and $\alpha_c = 2\sigma = 0.02$ nm. Print the time series of N_v and $\tilde{N}_v$ for $t \geq 100$ ps and $t \leq 500$ ps. Also print the forces that would be obtained for these frames under a harmonic potential on $\tilde{N}_v$ with $\kappa = 0.98$ kJ/mol and $N^* = -5$.

```
# indus.input

# Atoms targeted by INDUS
Target = [ atom_index 1-16500:4 ]

```

```
# Probe volume
ProbeVolume = {
  type   = sphere
  center = [ 2.5 2.5 2.5 ]
  r_max  = 0.6
  # Coarse-graining
  sigma   = 0.01
  alpha_c = 0.02
}

# Add a harmonic bias
Bias = {
  order_parameter = ntilde
  x_star = -5
  kappa  = 0.98
}

# XTC file to analyze
XtcFile = my_trajectory.xtc

# Production phase (ps)
t0 = 100
tf = 500

# Print time series of N_v and Ntilde_v, and biasing forces
PrintOutput = yes
PrintForces = yes
```

7 Choice of Biasing Parameters for Umbrella Sampling

When performing umbrella sampling with $\tilde{N}_v$ to obtain the free energy landscape, $F_v(\tilde{N}) = -k_{\mathrm{B}}T \ln P_v(\tilde{N})$, the parameters in the biasing potential used must be chosen carefully to ensure accurate results. Appropriate heuristics for choosing the parameters will depend upon the system of interest, the form of the potential(s) used, and the features of its expected free energy landscape. Here, I will present heuristics that we have found useful for biasing $\tilde{N}_v$ in atomistic simulations of dense liquids.

Suppose that we want to bias $\tilde{N}_v$ using a harmonic potential,

$$\mathcal{U}_{\mathrm{bias}}(\tilde{N}_v) = \frac{1}{2}\kappa(\tilde{N}_v - N^*)^2, \tag{3}$$

where κ and N^* are constants (the biasing parameters). For INDUS in dense liquids such as water, we use the statistics of N_v in the liquid basin in an unbiased simulation to guide our choice of biasing parameters. Let the average and variance of N_v in the liquid basin be $\langle N_v \rangle_0$ and $\langle (\delta N_v)^2 \rangle_0$, respectively. Assuming that the liquid basin is approximately Gaussian, its curvature is

$$\kappa_0 = \frac{k_{\mathrm{B}}T}{\langle (\delta N_v)^2 \rangle_0} \tag{4}$$

For most systems of interest, we have found that κ_0 is a decent estimate of the maximum curvature of the underlying landscape, $F_v(\tilde{N})$. We therefore choose κ as a multiple of κ_0:

$$\kappa = \alpha\kappa_0 \tag{5}$$

where $\alpha = 2$ or 3.

Accurate WHAM results also require overlap of the biased free energy landscapes, $F_v^{(\kappa,N^*)}(\tilde{N})$, between adjacent windows. For a desired overlap of $\Delta F_{\text{overlap}}$, assuming that the biased ensembles for each N^*-value are Gaussian leads an estimated spacing between N^*-values of

$$\Delta N^* = 2\sqrt{\frac{(\beta \Delta F_{\text{overlap}})\langle(\delta N_v)^2\rangle_0}{1+\alpha}} \tag{6}$$

where $\beta = (k_{\text{B}}T)^{-1}$. For an initial guess, we typically choose $\alpha = 2$ and $\Delta F_{\text{overlap}} = 4\ k_{\text{B}}T$. This leads to:

$$\kappa = 2\kappa_0 \tag{7}$$

$$\Delta N^* = \frac{4}{\sqrt{3}}\sqrt{\langle(\delta N_v)^2\rangle_0} \tag{8}$$

Note that sampling $N_v = 0$ may require you to use windows with $N^* < 0$.

7.1 Example: Umbrella Sampling in Bulk Water

To illustrate the heuristics outlined above, I describe here the procedure to compute the free energy as a function of $\tilde{N}$, $F_v(\tilde{N})$, in a sphere (radius $r = 0.6$ nm) in bulk water at 300 K and 1 bar. Since it is infeasible to use standard MD simulations to sample $\tilde{N}$ outside of a narrow range near $\langle N\rangle_0$, an enhanced sampling technique is required. I chose to use umbrella sampling [7, 1]. Sample `plumed.dat` and `indus.input` input files are available in the code repository under `manual/examples/bulk_water`.

All simulations were performed using GROMACS 2016.3 [8], patched with an installation of PLUMED+INDUS based on PLUMED 2.4.0 [9]. The simulation consisted of about 4,000 SPC/E water molecules [10] in a square box. The temperature was maintained at $T = 300$ K using the stochastic velocity rescale thermostat [11] with a time constant of $\tau_T = 0.5$ ps, and the pressure was maintained at $P = 1$ bar using an isotropic Parrinello-Rahman barostat [12] with time constant $\tau_P = 2$ ps. The simulation was run for 2 ns with the leapfrog integrator and a time step of 2 fs. Samples were taken every 1 ps (500 steps).

First, I performed an unbiased simulation to guide the choice of biasing parameters for a harmonic potential (see Equation 3). I discarded the first 500 ps and used the remaining frames to compute $\langle N_v\rangle_0 \approx 30$ and $\langle(\delta N_v)^2\rangle_0 \approx 5.09855$. From this, I computed $\kappa \approx 0.98$ kJ/mol and $\Delta N^* \approx 5$ using Equations 7 and 8.

Using these parameters, I ran a series of biased simulations with $N^* = 25, 20, 15, 10, 5, 0,$ and -5 for 2 ns each. The INDUS coarse-graining parameters were $\sigma = 0.01$ nm and $\alpha_c = 2\sigma = 0.02$ nm. I discarded the first 200 ps of each trajectory and used the remaining data (along with the data from the unbiased simulation) to compute the unbiased free energy distribution, $F_v(\tilde{N})$, using UWHAM [13]. Figure 1 illustrates the results. Note that if the free energy as a function of N is desired, these results can be reweighted to obtain $F_v(N)$ (see ref. [1]).

8 Developer Notes

This section is intended for those who wish to implement new probe volume geometries and/or understand the inner workings of the INDUS code. A working understanding of C++ is assumed.

NOTE: This section currently under construction.

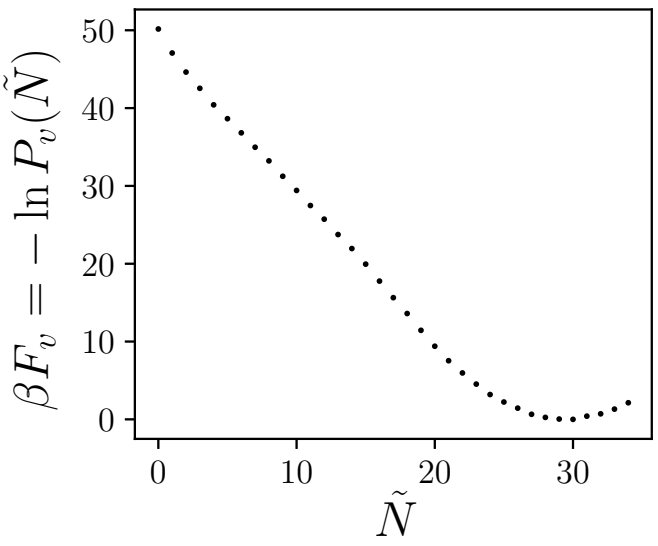

Figure 1: The free energy landscape for the coarse-grained number of waters, $\tilde{N}$, in a small probe volume in bulk water at $T = 300$ K and $P = 1$ bar. The probe volume, v, is a sphere of radius $r = 0.6$ nm in a box of about $4,000$ SPC/E water molecules.

8.1 Adding New Probe Volumes

Implementing a new probe volume geometry is relatively simple. All geometries must be derived classes of abstract base class `ProbeVolume`, which handles the interface between particular geometries and the rest of the code. New probe volumes must implement the following methods:

- Constructor
- `isInProbeVolume`: given the position of a particle i, this function returns a `bool` indicating whether or not the particle is in the nominal probe volume, *i.e.* whether it contributes to N_v. It also computes indicator functions $h_v(i)$ and $\tilde{h}_v(i)$, the derivatives $\partial\tilde{h}_v(i)/\partial\mathbf{r}_i$, and whether the particle is in shell 1 or shell 2.
- (some other helper methods ...)

Additionally, you must register your probe volume with the INDUS driver. Each probe volume has a unique name associated with it, which I refer to as its *type*. This corresponds to the `type` field when declaring a `ProbeVolume` in `indus.input`. For an example of how to do this, see the code at the top of `ProbeVolume_Sphere.cpp`.

8.1.1 What are the "shells"?

As you look through the source code, you will see many references to "shell 1" and "shell 2." In the context of the probe volume, these are regions of space that are within a certain distance of the edges of the coarse-grained probe volume. You can think of shell 1 as the first coordination shell around the probe volume, and shell 2 as the second coordination shell. This functionality is not currently used, but will be important for future additions that I would like to make.

To make things more concrete, let w_1 and w_2 be the widths of shell 1 and shell 2, respectively. Consider a spherical probe volume with radius r centered at point $\mathbf{r}_0$ that has coarse-graining parameters σ and α_c. A particle i is assigned to different regions depending on its distance from the center of the sphere, $r_{i,0}$. Then

$$\text{particle } i \text{ is in } \begin{cases} \text{the "nominal" probe volume, } v, & \text{for } r_{i,0} \leq r \\ \text{the "coarse-grained" probe volume, } \tilde{v}, & \text{for } r_{i,0} \leq \tilde{r} \\ \text{shell 1}, & \text{for } \tilde{r} < r_{i,0} \leq r_1 \\ \text{shell 2}, & \text{for } r_1 < r_{i,0} \leq r_2 \end{cases} \tag{9}$$

where

$$\tilde{r} = r + \alpha_c \qquad \text{(outer radius of } \tilde{v}\text{)} \tag{10}$$

$$r_1 = \tilde{r} + w_1 \qquad \text{(outer radius of shell 1)} \tag{11}$$

$$r_2 = r_1 + w_2 = \tilde{r} + w_1 + w_2 \qquad \text{(outer radius of shell 2)} \tag{12}$$

As mentioned previously, the shells are not currently used in the code but will be used in future versions to support new features. If you only care about doing standard INDUS, then you can just return `is_in_shell_1 = false` and `is_in_shell_2 = false` wherever applicable and the code will work just fine for INDUS.

8.2 Parallelization with MPI

INDUS makes use of MPI to support limited parallelization within the PLUMED framework.

References

[1] Amish J Patel, Patrick Varilly, and David Chandler. Fluctuations of water near extended hydrophobic and hydrophilic surfaces. *The Journal of Physical Chemistry B*, 114(4):1632–1637, 2010.

[2] Amish J Patel, Patrick Varilly, David Chandler, and Shekhar Garde. Quantifying density fluctuations in volumes of all shapes and sizes using indirect umbrella sampling. *Journal of statistical physics*, 145(2):265–275, 2011.

[3] Amish J Patel, Patrick Varilly, Sumanth N Jamadagni, Michael F Hagan, David Chandler, and Shekhar Garde. Sitting at the edge: How biomolecules use hydrophobicity to tune their interactions and function. *The Journal of Physical Chemistry B*, 116(8):2498–2503, 2012.

[4] Amish J Patel and Shekhar Garde. Efficient method to characterize the context-dependent hydrophobicity of proteins. *The Journal of Physical Chemistry B*, 118(6):1564–1573, 2014.

[5] Erte Xi, Richard C Remsing, and Amish J Patel. Sparse sampling of water density fluctuations in interfacial environments. *Journal of chemical theory and computation*, 12(2):706–713, 2016.

[6] Erte Xi, Sean M Marks, Suruchi Fialoke, and Amish J Patel. Sparse sampling of water density fluctuations near liquid-vapor coexistence. *Molecular Simulation*, pages 1–12, 2018.

[7] Glenn M Torrie and John P Valleau. Nonphysical sampling distributions in monte carlo free-energy estimation: Umbrella sampling. *Journal of Computational Physics*, 23(2):187–199, 1977.

[8] David Van Der Spoel, Erik Lindahl, Berk Hess, Gerrit Groenhof, Alan E Mark, and Herman JC Berendsen. Gromacs: fast, flexible, and free. *Journal of computational chemistry*, 26(16):1701–1718, 2005.

[9] Gareth A Tribello, Massimiliano Bonomi, Davide Branduardi, Carlo Camilloni, and Giovanni Bussi. Plumed 2: New feathers for an old bird. *Computer Physics Communications*, 185(2):604–613, 2014.

[10] HJC Berendsen, JR Grigera, and TP Straatsma. The missing term in effective pair potentials. *Journal of Physical Chemistry*, 91(24):6269–6271, 1987.

[11] Giovanni Bussi, Davide Donadio, and Michele Parrinello. Canonical sampling through velocity rescaling. *The Journal of chemical physics*, 126(1):014101, 2007.

[12] Michele Parrinello and Aneesur Rahman. Polymorphic transitions in single crystals: A new molecular dynamics method. *Journal of Applied physics*, 52(12):7182–7190, 1981.

[13] Zhiqiang Tan, Emilio Gallicchio, Mauro Lapelosa, and Ronald M Levy. Theory of binless multi-state free energy estimation with applications to protein-ligand binding. *The Journal of chemical physics*, 136(14):04B608, 2012.