Practical 3: Metadynamics Analysis

Introduction

In the last tutorial we used well-tempered metadynamics to influence alanine-dipeptide to make more transitions between minima in the phi angle space. More transitions were observed. But, because we add a bias potential the probability distribution is distorted and the associated free energy is aldo distorted.

We need to correct for the added bias in order to obtain the unbiased probability and unbiased free energy from the biased simulation.

The correction can be mathematically done using the following equations…

$$P(s) = \frac{1}{N}\sum_{k=1}^{N} \delta(s - s(\vec{x}_k))w(\vec{x}_k)$$

Where

$$w(\vec{x}) = \frac{P(\vec{x})}{P_{bias}(\vec{x})} = \frac{Z'}{Z}e^{\frac{V_{bias}}{k_B T}}$$

$$P(s) = \int \delta(s - s(\vec{x}))\, P_{bias}(\vec{x})w(\vec{x})d\vec{x}$$

Then, from that expression we can get the unbiased free energy with respect to the collective variable.

$$F(s) = -k_B T ln\big(P(s)\big)$$

Metadynamics Analysis: Free Energy Map Plotting

We will download the COLVAR file which contains the bias information. Then, we will upload this file to google drive and use google collab codes with ml_colvar to make a plot of the free energy with respect to collective coordinate.

1.) Download the COLVAR file

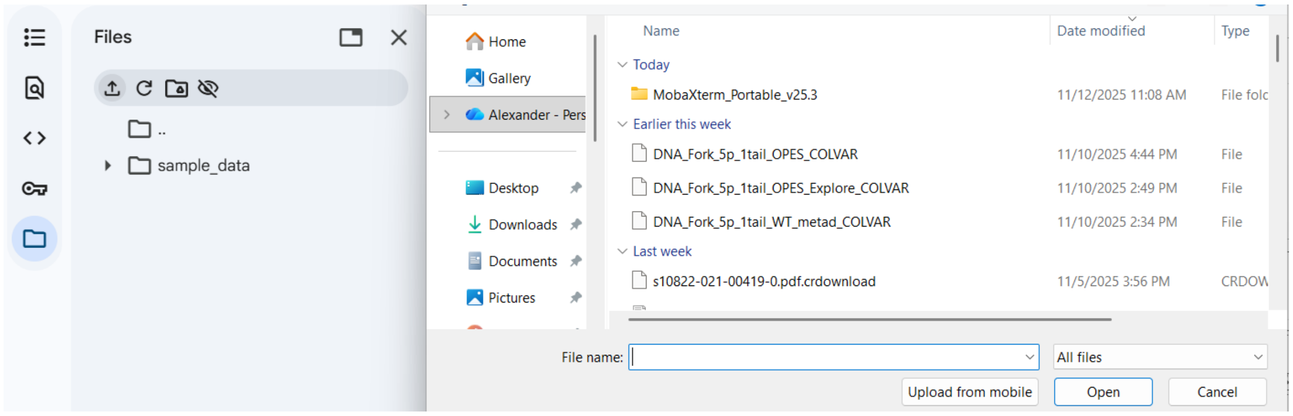
2.) Open your google drive

3.) Click new -> more -> connect more apps -> google collaboratory

4.) Create a folder where you will store your google collab codes

5.) Create a new google collab and name it "Dipeptide_metadynamics_analysis"

6.) Upload the COLVAR file using folder icon and upload symbol



Name	Date modified	Type
Today		
MobaXterm_Portable_v25.3	11/12/2025 11:08 AM	File folc
Earlier this week		
DNA_Fork_5p_1tail_OPES_COLVAR	11/10/2025 4:44 PM	File
DNA_Fork_5p_1tail_OPES_Explore_COLVAR	11/10/2025 2:49 PM	File
DNA_Fork_5p_1tail_WT_metad_COLVAR	11/10/2025 2:34 PM	File
Last week		
s10822-021-00419-0.pdf.crdownload	11/5/2025 3:56 PM	CRDOW

7.) The first part of the script that you need to type is…

```
!git clone https://github.com/luigibonati/mlcolvar.git
```

This clones code from github called mlcolvar which is a very useful package of codes to plot corrected free energy surfaces from enhanced sampling simulations.

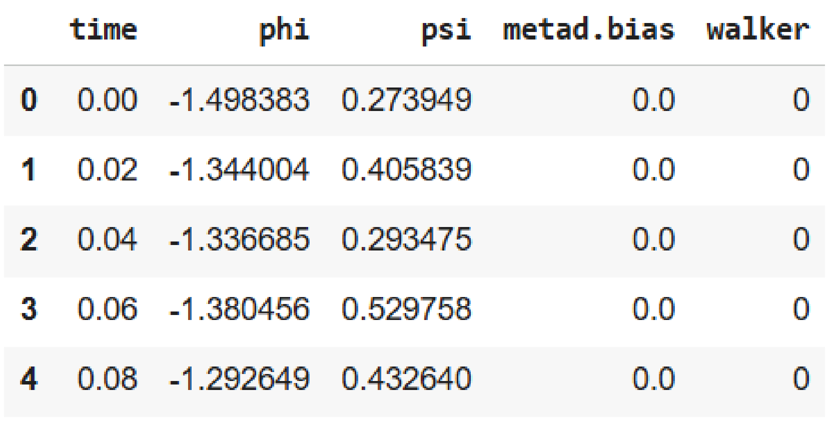
8.) Next we need to pip install the codes to be able to use them

```
# Activate here your Python virtual environment (e.g., with venv or conda).
!pip install mlcolvar/.
```

9.) Then we load the colvar data and visualize the dataframe

```
import pandas as pd
from mlcolvar.utils.io import load_dataframe
from mlcolvar.utils.timelagged import create_timelagged_dataset
from mlcolvar.data import DictModule

# load colvar files containing weights of the OPES simulations as well as
the driver with the input features
df = load_dataframe('COLVAR_Dipeptide_10ns')
df.head()
```

It should output a table that looks like



	time	phi	psi	metad.bias	walker
0	0.00	-1.498383	0.273949	0.0	0
1	0.02	-1.344004	0.405839	0.0	0
2	0.04	-1.336685	0.293475	0.0	0
3	0.06	-1.380456	0.529758	0.0	0
4	0.08	-1.292649	0.432640	0.0	0

Which contains the time, phi, psi, and bias. The metad.bias is zero initially but at the end of the file it is nonzero.

10.) Then we define the temperature and Boltzmann constant

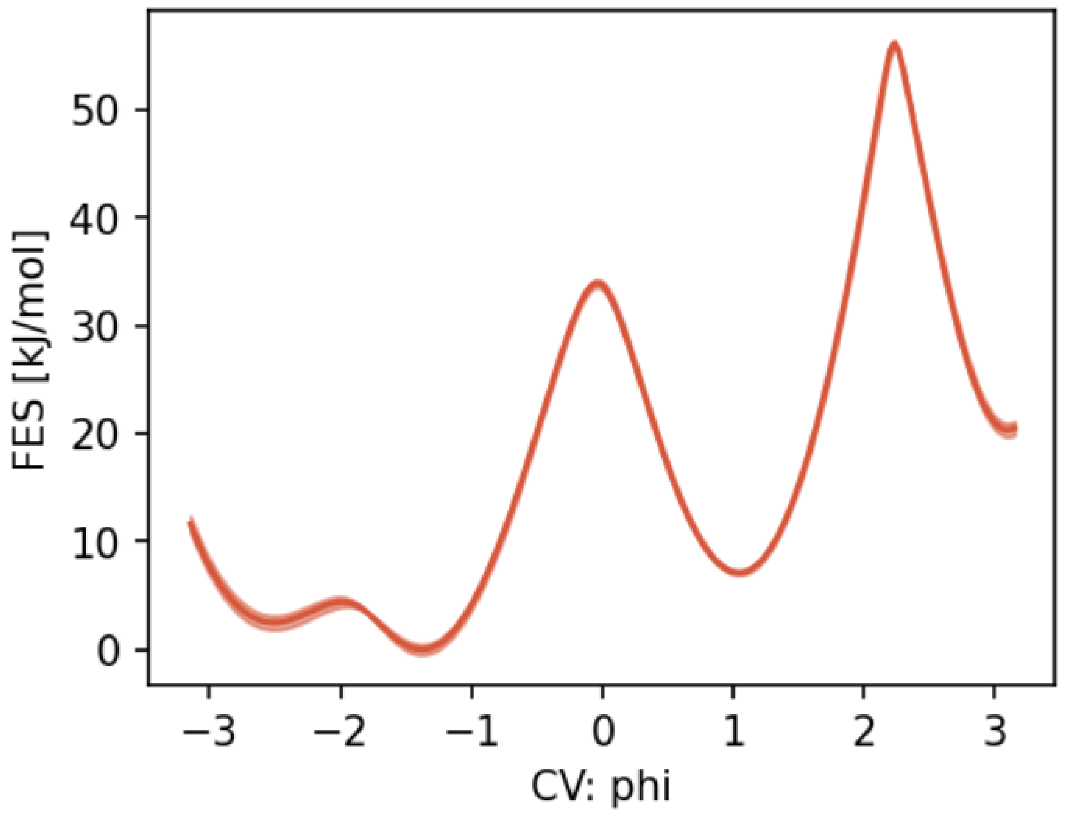
```
from mlcolvar.utils.plot import paletteFessa
from mlcolvar.utils.io import load_dataframe
from mlcolvar.utils.fes import compute_fes
import matplotlib.pyplot as plt
import numpy as np

# Load COLVAR file containing collective variables (and bias information)
colvar = load_dataframe('COLVAR_Dipeptide_10ns')

# Simulations parameters
temperature = 300
kb = 0.0083144621       # Boltzmann constant in kJ/(mol·K)
kbt = kb * temperature
```

11.) Next we define the weights.

```
# (1) unbiased simulation
#bias = None

# (2) reweight for a single bias
bias = colvar['metad.bias'].values

# (3) reweight multiple bias potentials (e.g. OPES and harmonic walls)
#bias = colvar[['metad.bias','uwall.bias']].sum(axis=1).values

# calculate the weights
weights = np.exp( bias / kbt )
```

12.) After that, we use the compute_fes() function to plot the corrected free energy surface. This calculates the free energy with error bars according to block analysis. Here we use 2 blocks to calculate error.

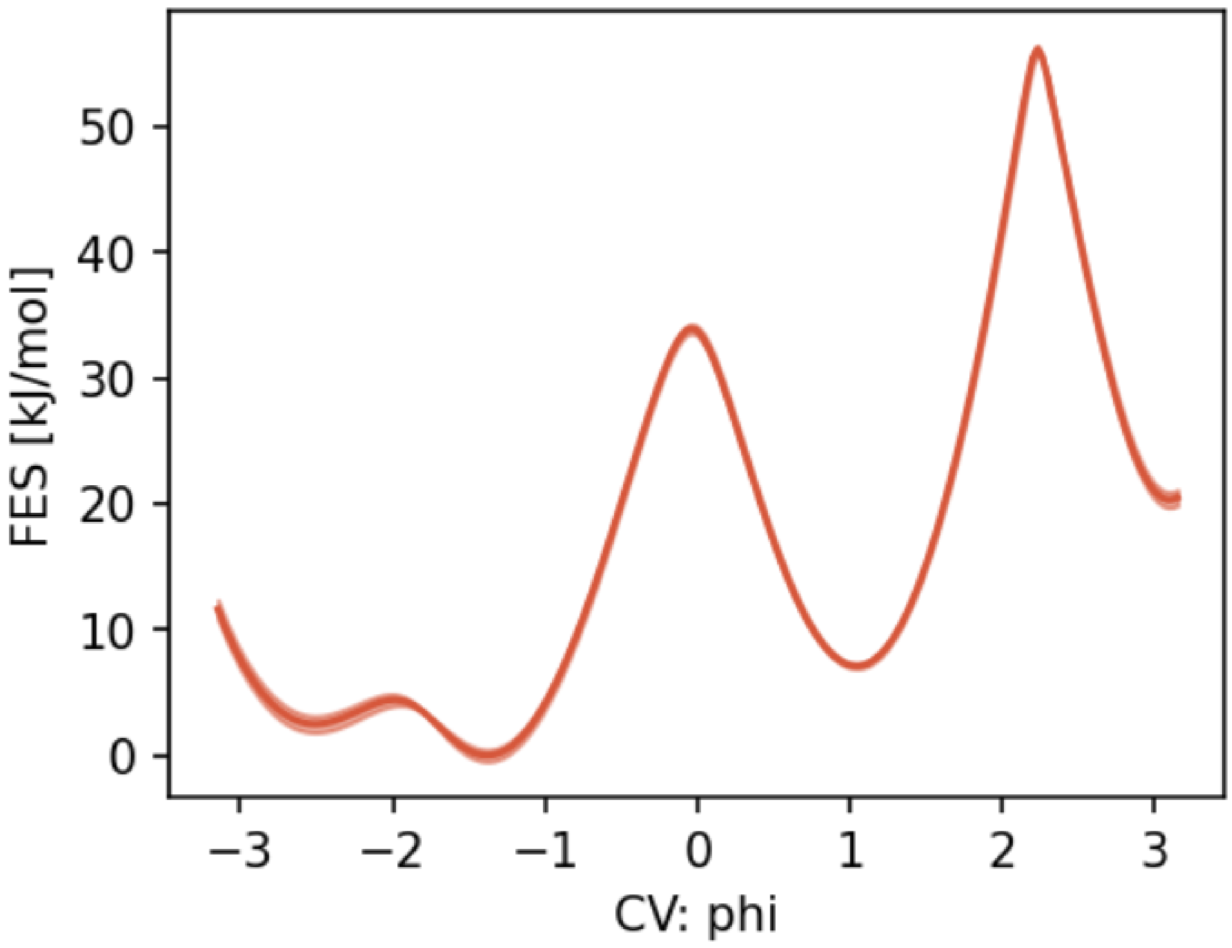
```
cv_name = 'phi'
cv1 = colvar[cv_name].values

fes_params = {
    'blocks': 2,              # Number of blocks for error analysis
    'bandwidth': 0.02,        # Kernel bandwidth (sigma) for density
estimation. if 'range', the bandwidth is expressed as ratio of the range of
values (e.g. here it is 1% of it)
    'scale_by': 'range',      # Method to scale the bandwidth
    'temp' : temperature,     # temperature for proper energy scaling
(alternative to kbt)
    'fes_units': 'kJ/mol',    # units of the free energy
    'weights': weights,       # Statistical weights from the bias
}

fig, ax = plt.subplots(figsize=(4,3),dpi=150)
fes1D, grid1D, bounds1D, error1D = compute_fes(
    cv1,
    plot=True,
    ax = ax,
    **fes_params
)
ax.set_xlabel(f'CV: {cv_name}')
plt.show()
```



13.) We can also calculate the error using more blocks.

```
cv_name = 'phi'
cv1 = colvar[cv_name].values

fes_params = {
    'blocks': 4,              # Number of blocks for error analysis
    'bandwidth': 0.02,        # Kernel bandwidth (sigma) for density
estimation. if 'range', the bandwidth is expressed as ratio of the range of
values (e.g. here it is 1% of it)
    'scale_by': 'range',      # Method to scale the bandwidth
    'temp' : temperature,     # temperature for proper energy scaling
(alternative to kbt)
    'fes_units': 'kJ/mol',    # units of the free energy
    'weights': weights,       # Statistical weights from the bias
}

fig, ax = plt.subplots(figsize=(4,3),dpi=150)
fes1D, grid1D, bounds1D, error1D = compute_fes(
    cv1,
    plot=True,
    ax = ax,
    **fes_params
)
ax.set_xlabel(f'CV: {cv_name}')
```

```
plt.show()
```



The error bars for these plots are very small because our system has converged well and it is relatively simple motion that we were trying to enhance sampling for. For more complex systems the error bars can be different sizes if you choose different block sizes.

Metadynamics Analysis: Free Energy Convergence

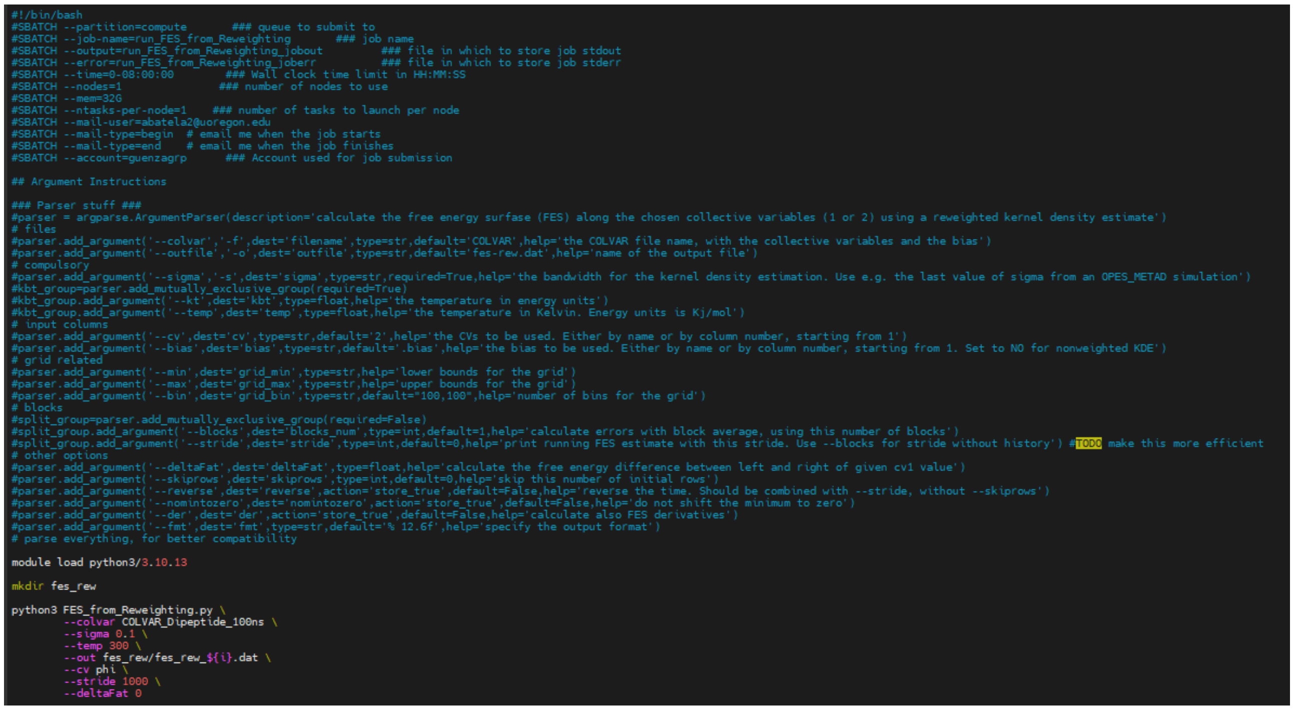
A good check to see if your system has converged free energy is to compare how the free energy difference between two minima changes vs time as the metadynamics simulation progresses.

$$\Delta F(t) = \int F_{minima_1}(t)ds - \int F_{minima_2}(t)ds$$

We will go back on Talapas for this step.

1.) Go into the folder containing your WT-Metadynamics run and copy the python code “FES_from_reweighting.py” from “ch447_547/shared/WT_metadynamics_analysis_code” into it.

2.) We also create the following bash script “run_FES_from_Reweighting.sh”



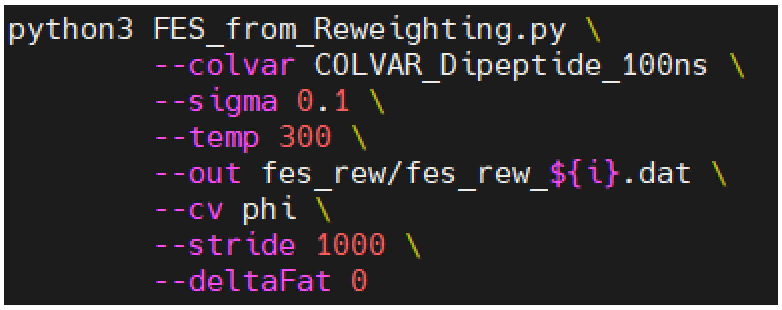
```
#!/bin/bash
#SBATCH --partition=compute       ### queue to submit to
#SBATCH --job-name=run_FES_from_Reweighting       ### job name
#SBATCH --output=run_FES_from_Reweighting_jobout         ### file in which to store job stdout
#SBATCH --error=run_FES_from_Reweighting_joberr          ### file in which to store job stderr
#SBATCH --time=0-08:00:00        ### Wall clock time limit in HH:MM:SS
#SBATCH --nodes=1               ### number of nodes to use
#SBATCH --mem=32G
#SBATCH --ntasks-per-node=1    ### number of tasks to launch per node
#SBATCH --mail-user=abatela2@uoregon.edu
#SBATCH --mail-type=begin  # email me when the job starts
#SBATCH --mail-type=end    # email me when the job finishes
#SBATCH --account=guenzagrp      ### Account used for job submission

## Argument Instructions

### Parser stuff ###
#parser = argparse.ArgumentParser(description='calculate the free energy surfase (FES) along the chosen collective variables (1 or 2) using a reweighted kernel density estimate')
# files
#parser.add_argument('--colvar','-f',dest='filename',type=str,default='COLVAR',help='the COLVAR file name, with the collective variables and the bias')
#parser.add_argument('--outfile','-o',dest='outfile',type=str,default='fes-rew.dat',help='name of the output file')
# compulsory
#parser.add_argument('--sigma','-s',dest='sigma',type=str,required=True,help='the bandwidth for the kernel density estimation. Use e.g. the last value of sigma from an OPES_METAD simulation')
#kbt_group=parser.add_mutually_exclusive_group(required=True)
#kbt_group.add_argument('--kt',dest='kbt',type=float,help='the temperature in energy units')
#kbt_group.add_argument('--temp',dest='temp',type=float,help='the temperature in Kelvin. Energy units is Kj/mol')
# input columns
#parser.add_argument('--cv',dest='cv',type=str,default='2',help='the CVs to be used. Either by name or by column number, starting from 1')
#parser.add_argument('--bias',dest='bias',type=str,default='.bias',help='the bias to be used. Either by name or by column number, starting from 1. Set to NO for nonweighted KDE')
# grid related
#parser.add_argument('--min',dest='grid_min',type=str,help='lower bounds for the grid')
#parser.add_argument('--max',dest='grid_max',type=str,help='upper bounds for the grid')
#parser.add_argument('--bin',dest='grid_bin',type=str,default="100,100",help='number of bins for the grid')
# blocks
#split_group=parser.add_mutually_exclusive_group(required=False)
#split_group.add_argument('--blocks',dest='blocks_num',type=int,default=1,help='calculate errors with block average, using this number of blocks')
#split_group.add_argument('--stride',dest='stride',type=int,default=0,help='print running FES estimate with this stride. Use --blocks for stride without history') #TODO make this more efficient
# other options
#parser.add_argument('--deltaFat',dest='deltaFat',type=float,help='calculate the free energy difference between left and right of given cv1 value')
#parser.add_argument('--skiprows',dest='skiprows',type=int,default=0,help='skip this number of initial rows')
#parser.add_argument('--reverse',dest='reverse',action='store_true',default=False,help='reverse the time. Should be combined with --stride, without --skiprows')
#parser.add_argument('--nomintozero',dest='nomintozero',action='store_true',default=False,help='do not shift the minimum to zero')
#parser.add_argument('--der',dest='der',action='store_true',default=False,help='calculate also FES derivatives')
#parser.add_argument('--fmt',dest='fmt',type=str,default='% 12.6f',help='specify the output format')
# parse everything, for better compatibility

module load python3/3.10.13

mkdir fes_rew

python3 FES_from_Reweighting.py \
        --colvar COLVAR_Dipeptide_100ns \
        --sigma 0.1 \
        --temp 300 \
        --out fes_rew/fes_rew_${i}.dat \
        --cv phi \
        --stride 1000 \
        --deltaFat 0
```

The first part contains the sbatch headers. This code can be run on a compute node.

The 2nd commented part contains information about the possible arguments that can be fed into the FES_from_Reweighting.py script.

The last part load python, creates a folder called few_rew, and makes a bunch of free energy calculations over a range of times.

Let's focus on what the python code does.



```
python3 FES_from_Reweighting.py \
        --colvar COLVAR_Dipeptide_100ns \
        --sigma 0.1 \
        --temp 300 \
        --out fes_rew/fes_rew_${i}.dat \
        --cv phi \
        --stride 1000 \
        --deltaFat 0
```

--colvar: takes the COLVAR file as input

--sigma: need to specify the sigma which can be founded in the HILLS file at the end. For WT-Metadynamics it doesn't change from the value 0.1 which we set it as initially.

--temp: we ran our simulation at 300 K

-out: where the output free energy will be saved

--cv: the collective variable with respect we will calculate the free energy for

--stride: how many frames to skip before calculating new free energy

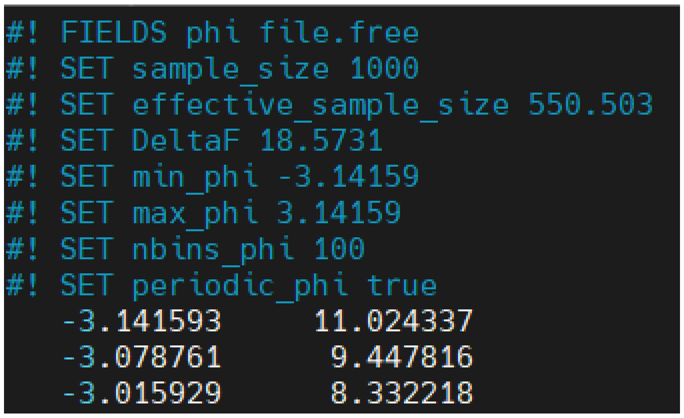
--deltaFat: location of the barrier between the two free energy minima. For our system according to our previous plots deltaFat=0 for this system.

3.) run the script using “sbatch run_FES_from_Reweighting.sh”

4.) copy “deltaFat_vs_time.py” from “ch447_547/shared/WT_metadynamics_analysis_code” to the “fes_rew” output folder.

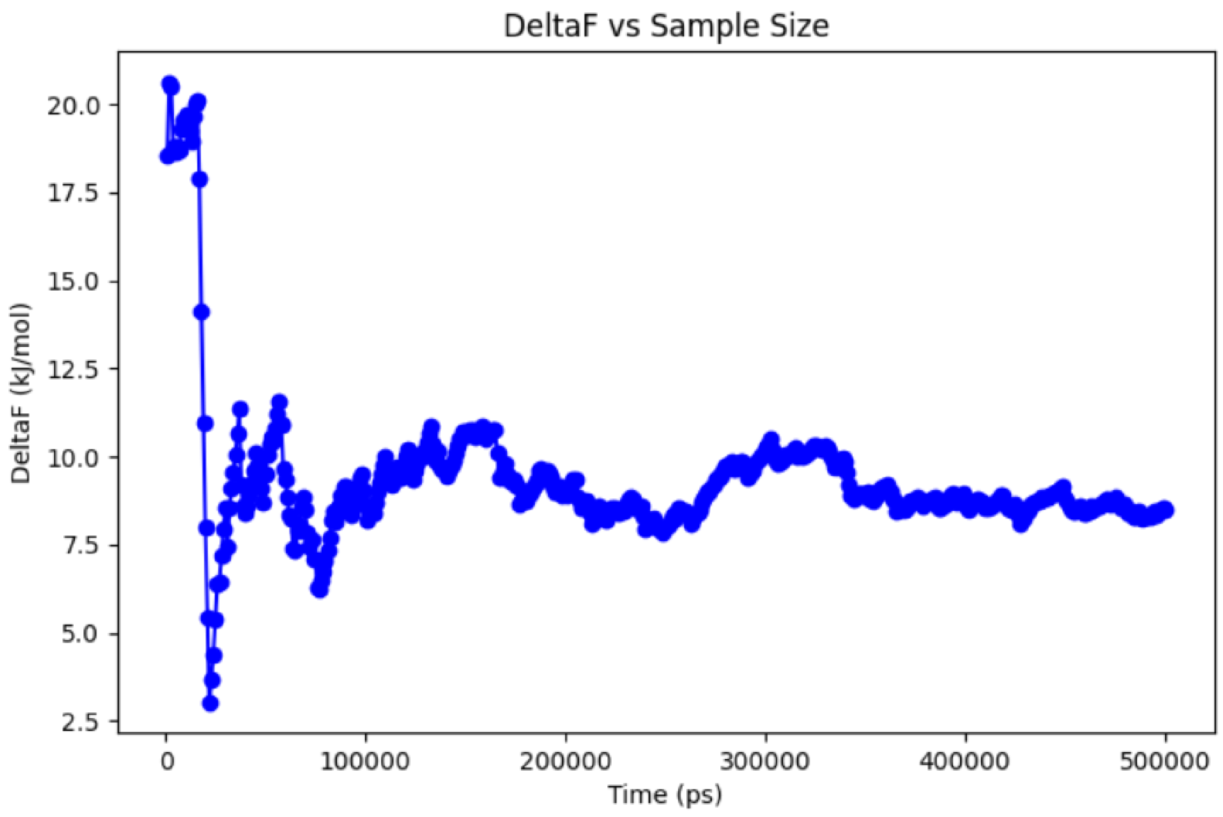
The “fes_rew" output folder contains a bunch of “fes_rew__${i}.dat” files. Each of these can be plotted as a free energy map along phi at various time-points in the simulation.

Here is what the top of one of the “fes_rew__${i}.dat” files looks like



```
#! FIELDS phi file.free
#! SET sample_size 1000
#! SET effective_sample_size 550.503
#! SET DeltaF 18.5731
#! SET min_phi -3.14159
#! SET max_phi 3.14159
#! SET nbins_phi 100
#! SET periodic_phi true
   -3.141593     11.024337
   -3.078761      9.447816
   -3.015929      8.332218
```

The “deltaFat_vs_time.py” will take the sample_size and deltaF output from each “fes_rew__${i}.dat” file and plot them on a plot. Sample_size is same as time in picoseconds since we are saving every 1 picosecond in our simulation. The DeltaF is the free energy difference between both minima at that specific point in the simulation.



You will obtain a plot which looks shows what the free energy difference looks like as a function of time. We can see that the free energy difference stabilizes at around 350,000 ps. So, we know that the free energy has converged and we can trust the shape of the free energy curve which we plotted before.