COWBOE

Construction Of Windows Based on Energy

What is cowboe

COWBOE is a python 3 tool for parameter selection in Umbrella Sampling which is a free energy calculation method used in Molecular dynamics simulations.

Features

- It is a Python 3 module hence easy to install.
- Parameter selection which tunes the total number of windows and force constants used with Umbrella sampling are done.
- Parameters are optimized using Nelder-mead / Restricted Nelder-Mead simplex optimization algorithm.
- Progress of the optimization can be tracked.
- Module includes functions to perform comparison and visualization of different PMFs (Potential of Mean Forces / Free energy files) and the NM algorithm results.

Installation

`cowboe requires Python 3.6> to run.`

Creating a virtual environment

It is best to install cowboe in a new environment

Using venv For Linux/macOS,

```
python3 -m venv /path/to/new/virtual/environment
source <venv>/bin/activate
```

For Windows,

```
c:\>python -m venv c:\path\to\myenv
c:\> <venv>\Scripts\activate.bat
```

Using conda

```
conda create -n cowboe python=3.7
conda activate cowboe
```

Installing pip

pip is already installed if you are using Python 3 >=3.4 downloaded from python.org or if you are working in a Virtual Environment created by virtualenv or venv. Just make sure to upgrade pip using,

```
python3 -m pip install --upgrade pip
```

Installing cowboe

cowboe is available on pypi and can be installed using pip as follows,

```
pip install cowboe
```

Building from source

Download the source as a zip file from GitHub and extract the files to a new folder. Before building cowboe install the dependencies using pip as shown below and it will resolve all dependencies conflicts.

```
cd <location of extracted files with setup.py file>
pip3 install numpy scipy matplotlib seaborn shapely imageio pandas
python setup.py install
```

After the installation cowboe module should be available and it can be checked by using,

```
python -c "import cowboe"
```

Any error means cowboe was not installed sucessfully.

Usage

cowboe has different functions which perform individual task like running the cowboe algorithm, performing NM optimization, visualization of the pmf curves and summarizing the NM steps. A detailed explanation and examples with the required data files are provided in the examples directory. A comprehensive example is shown below,

```
from cowboe import pmftopoints, cowboe, cowboefit, settings_update
from cowboe import cowboeKS, cowboeRNM, cowboeNM, progressfile, NMprogress,
cowboe3Dsurface
from cowboe import cowboe_wham, pmfcompare, multi_pmfcompare, cowboe_settings,
wham_settings
from cowboe import cowboe_trajcut, cowboe_OVL, cowboe_pmfplot, pmfdiff
import numpy as np
import os
import warnings

os.chdir('location of the examples folder')

# uncomment the below two lines if you donot want to see the plot while running from
commandline
# matplotlib.use('Agg')
# warnings.filterwarnings("ignore")

# to update the cowboe_setting dictionary.
cowboe_settings.update({"param B" : 2.0})

# to convert the rest pmf into array for further processing.
'''The below function generates a pickle file called variable.pkl and it will be used
for further steps'''
pmftopoints(testpmf='test_pmf.txt')

# algorithm to get window distribution of the given parameters based on the test pmf
data.
'''
for the initial guess from pmftopoints() the below function runs the cowboe algorithm
to itertively
construct windows and generates the samplingtime and forceconstant for each window in
order to perform MD runs.
'''

'''
Here sc in the below function is the sampling considered in terms of ns
here 8ns corresponds to 8ns/window for a total of n number of windows
The defaults values like the conventional number of windows etc are used
to calculate how much computations the user would have used through the conventional
routine.

And these and other values can be adjusted using dict update

e.g. cowboe_settings.update({'conventional no of windows' : 24})
'''

cowboe(A=3.5, V = 0.8 , sc =8, name=3)

# to update the wham_setting dictionary
wham_settings.update({"tol" : 0.00015})

# calculates PMF using wham for the given trajectories # must update list.txt which is
the metadata file for the wham calculation.
# more information on the tool is available here,
'''
Grossfield, Alan, “WHAM: the weighted histogram analysis method”,
version 2.0.10, http://membrane.urmc.rochester.edu/wordpress/?page_id=126
'''

'''
All trajectory files must be given a '.traj' extension
list.txt file should also be present for identifying the trajectory files.
This file is same as the metadata file described in the wham documentation,
http://membrane.urmc.rochester.edu/wordpress/?page_id=126

'''
cowboe_wham(name = 'benchmark.txt', location ='</cowboe/examples/benchmark>', MCtrials
= 0)
cowboe_wham(name = '3.txt', location ='<cowboe/examples/3>', MCtrials = 0)

cowboe_pmfplot(pmf='1.txt', name='1_pmf', splice=0)

# calculates the fitness of a test case w.r.t to the benchmark case.
cowboefit(test='3.txt',bench='benchmark.txt')

'''
We need three initial parameter combinations to run the NM algorithm
Here the three initial guess are 1,2,3..

We can compare the pmf curves using the below functions
'''

pmfcompare(pmf1='1.txt', pmf2='3.txt', name='1-3-compare')
pmfdiff(pmf1='1.txt', pmf2='2.txt', name='1-2-compare')
multi_pmfcompare(pmfs=['1.txt', '2.txt', '3.txt'], name='multiple-compare', splices=
[0,0,0])

# A, V, max_vdist or fit() values for the three points
A = [2.0, 2.9, 3.5]
V = [0.75, 0.8700, 0.8000]
fit = [1.9834, 1.3844, 4.7587]

# Gives the possible moves with (R)NM algorithm for the above defined simplex.
cowboeNM(A = A, V = V, fit = fit)
cowboeRNM(A = A, V = V, fit = fit)

'''
After a series of (R)NM steps, the user will have a set of subsequent simplexes
each of which will be a 1*3*3 array, say if we have 6 steps, we will have a
6*3*3 array
'''

p = np.array(   [[[2.    , 0.75  , 1.9943],
                [2.9   , 0.87  , 1.8232],
                [3.5   , 0.8   , 4.7636]],

                [[2.    , 0.75  , 1.9943],
                 [2.9   , 0.87  , 1.8232],
                 [1.6571, 0.82  , 0.9899]],

                [[2.4028, 0.94  , 2.0045],
                 [2.9   , 0.87  , 1.8232],
                 [1.6571, 0.82  , 0.9899]],

                [[2.0939, 0.7975, 1.4695],
                 [2.9   , 0.87  , 1.8232],
                 [1.6571, 0.82  , 0.9899]],

                [[2.0939, 0.7975, 1.4695],
                 [1.1965, 0.7475, 3.5148],
                 [1.6571, 0.82  , 0.9899]],

                [[2.0939, 0.7975, 1.4695],
                 [2.3242, 0.8394, 1.8202],
                 [1.6571, 0.82  , 0.9899]]])

'''
The below function writes a text file called progress.txt which summarizes the steps
in a clean format.
'''
progressfile(points=p)

'''
The below function plots the different NM steps and summarizes them as a gif
'''
NMprogress(progressfile = 'progress.txt')

'''
The below function creates a 3d surface of the paramater space using the
different parameter evaluations, inorder the use this functoin ffmpeg must be
installed in the
path

More information on this is available here,

https://www.ffmpeg.org/
https://www.ffmpeg.org/download.html
https://anaconda.org/conda-forge/ffmpeg
'''
cowboe3Dsurface(progressfile = 'progress.txt')

'''
The following functions are all used for modifying or testing the trajectory files and
other sampling related properties.
'''
cowboe_trajcut(percentage=50.0, location='<cowboe/examples/benchmark>',\
            name='benchmark',listfile='list.txt',start=0)
cowboeKS(location='<cowboe/examples/benchmark>', \
         listfile='list.txt', percentage = 85)
cowboe_OVL(location='<cowboe/examples/benchmark>'\
           , listfile='list.txt', name = 'benchmark', distplot=False)
'''
On calling the below function, details on default setting and ways to modify them will
be shown
'''
settings_update()
```

Nelder-Mead simplex algorithm

The primary difference between the NM and restricted NM is that the RNM doesnt include the expansion step after reflection. The cowboeNM and cowboeRNM provide possible NM steps for a given simplex and the user would select a step based on the below pseudocode.

`Pseudocode of the Simplex Nelder-Mead Optimization`

```
Initialize the simplex with  n-1 random starting parameter value combinations e.g. [A,
V],
where n is the number of parameters being optimized.

Restricted Nelder-Mead algorithm:

while loop not done

    calculate centroid
    calculate reflected

    if reflected is better than best solution then
        calculate expanded
        replace worst solution with better of reflected and expanded

    else if reflected is worse than all but worst then
        calculate outward contracted

        if outward contracted is better than reflected
            replace worst solution with outward contracted
        end if

        else
            shrink the search area

    else if reflected is worse than all
        calculate inward contracted

        if inward contracted is better than worst
            replace worst solution with inward contracted
        end if

        else
            shrink the search area

    else
        replace worst solution with reflected

    end if

    if the solution is within tolerance, exit loop

end loop

return best solution found
```

cowboe and the WHAM wrapper have been given defaults values which can be modified by changing the cowboe.py file in the installation location or to make temporary changes dict update() of can be used as shown above. More information on this can be obtained by calling the settings_update() function in cowboe.

License