DAFT tutorials

1: Association of two helices from sequence

This tutorial shows how DAFT can be used to study the association of two helices in a simple membrane. The helices are built from sequence, and this step uses PyMOL, which needs to be available. In addition, GROMACS needs to be available to run the simulations. All other components required are distributed together with DAFT, and should be available in the DAFT directory. The tutorial assumes a basic working knowledge of BASH.

Step 1: Setting up the environment

To make the commands easier to follow we set some variables prior to running DAFT. In particular, we specify the locations of GROMACS and of PyMOL. Note that the locations may differ on your machine, and the correct paths should be specified.

```
GMX=/usr/local/gromacs-4.6.1/bin/GMXRC
PYMOL=/usr/local/bin/pymol
```

Step 2: Setting up sequences

Now we define the amino acid sequences. The sequences used are those of glycophorin A wild type and the G83I mutant. We specify that they are helical by adding ":H" at the end. This specification is optional, as sequences are built helical by default, but it avoids using DSSP during topology generation.

```
wt=RASLIIFGVMAGVIGTILIN:H
mut=RASLIIFGVMAIVIGTILIN:H
```

These sequences correspond to the ones commonly used in TOXCAT assays, and differ from the biological ones in the anchor regions.

Step 3: Initializing the assays

From the sequences, we setup assays of 500 simulations (-n 500), of 512 ns each (-time 512), using an initial separation of 3.5 nm (-d 3.5) and a distance over z of 5 nm (-dz 5). We will use a POPC bilayer (-l POPC) and use regular Martini water (-sol W) to solvate the system. The bilayer and solvent are built by *insane*, which is called by DAFT.
To support the full range of features from insane, a specific syntax was developed, allowing specifying *insane* options on the command-line of DAFT. The basic syntax for this is `--insane-option=value`, but multiple options can be grouped together as `--insane{-option1=value1,-option2=value2}`. This yields the following command for setting up the simulations:

```
/path/to/DAFT/daft.sh -gmxrc $GMX -pymol $PYMOL \
    -tm WT=$wt -tm G83I=$mut -d 3.5 -dz 5 –n 500 -time 512 \
    --insane{-l=POPC,-sol=W} -c 2
```

The option –tm specifies a transmembrane sequence, which is defined as [name=]sequence[:structure]. If a name is given, it will be used for naming directories and it can be referred to when making combinations.

The option –c specifies the combinations to be made. Specifying '2' means making systems of dimers for each sequence. Heterodimers can be made by specifying '2!'. Alternatively, the specification –c WT,WT –c G83I,G83I could have been used to set up homodimers and –c WT,G83I could have been used for heterodimers. By default, if the option –c is not given, DAFT builds one type of system, combining all the single components.

The setup of 1000 simulation systems like this will take a few minutes (4m20 on a 2.30 GHz i7). The result will be two directories and several intermediate files, most notably the atomistic structures built, and the coarse grained structures and topologies:

```
daft-tutorial-1$ ls -l
total 136
drwxr-xr-x  3 tsjerk  staff    102 Feb 16 12:38 G83I-G83I
-rw-r--r--  1 tsjerk  staff    296 Feb 16 12:38 G83I-mart.ndx
-rw-r--r--  1 tsjerk  staff   2710 Feb 16 12:38 G83I-mart.pdb
-rw-r--r--  1 tsjerk  staff   7600 Feb 16 12:38 G83I.itp
-rw-r--r--  1 tsjerk  staff  12196 Feb 16 12:38 G83I.pdb
-rw-r--r--  1 tsjerk  staff    152 Feb 16 12:38 G83I.top
drwxr-xr-x  3 tsjerk  staff    102 Feb 16 12:38 WT-WT
-rw-r--r--  1 tsjerk  staff    289 Feb 16 12:38 WT-mart.ndx
-rw-r--r--  1 tsjerk  staff   2643 Feb 16 12:38 WT-mart.pdb
-rw-r--r--  1 tsjerk  staff   7447 Feb 16 12:38 WT.itp
-rw-r--r--  1 tsjerk  staff  11872 Feb 16 12:38 WT.pdb
-rw-r--r--  1 tsjerk  staff    146 Feb 16 12:38 WT.top
-rw-r--r--  1 tsjerk  staff    229 Feb 16 12:15 cmd.sh
```

Both directories contain a subdirectory 'daft', which contains the solvated starting structures, each in its own folder. Each of these folders contains the data required for a single run:

```
daft-tutorial-1$ ls -l WT-WT/daft/daft-0001
total 440
-rw-r--r--  1 tsjerk  staff    7447 Feb 16 12:38 WT.itp
-rw-r--r--  1 tsjerk  staff  160681 Feb 16 12:38 daft.gro
-rw-r--r--  1 tsjerk  staff   33740 Feb 16 12:38 daft.ndx
-rw-r--r--  1 tsjerk  staff    5171 Feb 16 12:38 daft.pdb
-rw-r--r--  1 tsjerk  staff     620 Feb 16 12:38 daft.top
-rw-r--r--  1 tsjerk  staff     214 Feb 16 12:38 martinate-run.sh
```

Step 4: Running simulations

The script 'martinate-run.sh' contains the command to execute the run in its directory, and can be called directly, or the data can be transferred to a compute cluster, GRID, or cloud facility and the script can be run from a job. We note that the script relies on *martinate.sh*, which is distributed together with DAFT. A working version of GROMACS is also required.

Step 5: Analysis

The analysis focuses on interaction energy and on relative orientations and contacts of the components. *martinate.sh* writes energy statistics for each component and each combination of components, including membrane and/or solvent.

Energy analysis

At each time, the DAFT ensemble has a characteristic energy distribution. This distribution develops over time and, given sufficient time, will converge to the equilibrium ensemble. This equilibrium ensemble will probably not be reached though, as the time scales are too short for that.
The evolution of the energy distribution over time, and of the interaction energy distribution in particular, can be followed from the twenty-one 5% points (vigintiles) and the mean. To construct these distributions across the ensemble, first the energy terms need to be extracted from the binary energy files Gromacs writes, and for each term the data need to be gathered. To facilitate that, a bash script (*eneall.sh*) is provided with DAFT to process a series of energy files and extract all interaction energy terms:

```
cd WT-WT/daft
/path/to/DAFT/Analysis/eneall.sh */*.edr
```

This yields the following set of files:

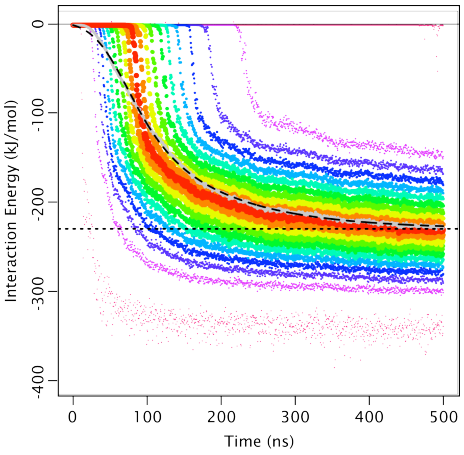
```
Membrane-Membrane.dat
Membrane-Solvent.dat
A_WT-Membrane.dat
A_WT-A_WT.dat
A_WT-B_WT.dat
A_WT-Solvent.dat
B_WT-Membrane.dat
B_WT-B_WT.dat
B_WT-Solvent.dat
Solvent-Solvent.dat
```

The first line in each file contains the time, while each following row corresponds to the interaction energy time series for a single run. Rows from time series shorter than the maximum run length are padded with 'NA', such that the whole data matrix

is filled. Further processing of these files can be done with the 'little-r' script *eneana.r*, which is a command-line operated R script to compute the vigintiles and graph them, and perform a non-linear least-squares fit of the mean interaction energy over time to a delayed decay model. Note that the latter is written specifically for the interaction energy between two components for which the binding assay is run.

```
/path/to/DAFT/Analysis/eneana.r A_WT-B_WT.dat
```

This writes a postscript file A_WT-B_WT.ps, which contains a graph of the vigintiles over time, like



The dashed line shows the fitted delayed decay model for the mean interaction energy, shown in grey. The plateau value of this model is indicated by the horizontal dotted line. The program also prints the summary of the fit of the average:

```
Formula: y ~ delayedDecay(tm, rate0, delay, plateau, rate)

Parameters:
          Estimate Std. Error t value Pr(>|t|)
rate0     3.300e-02  2.458e-03   13.43   <2e-16 ***
delay     4.025e+01  2.303e+00   17.48   <2e-16 ***
plateau -2.150e+02  7.025e-01 -305.97   <2e-16 ***
rate      1.539e-02  7.523e-04   20.46   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 15.19 on 1342 degrees of freedom

Number of iterations to convergence: 7
Achieved convergence tolerance: 9.091e-06
```

In addition, the summary of the fit for the median value (in red) is printed:

```
Formula: med ~ delayedDecay(tm, rate0, delay, plateau, rate)
```

```
Parameters:
          Estimate Std. Error  t value Pr(>|t|)
rate0    3.984e-01  5.045e-02    7.896 5.95e-15 ***
delay    6.195e+01  3.625e-01  170.882  < 2e-16 ***
plateau -2.179e+02  7.722e-01 -282.162  < 2e-16 ***
rate     1.537e-02  3.425e-04   44.888  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 19.68 on 1342 degrees of freedom

Number of iterations to convergence: 12
Achieved convergence tolerance: 6.885e-06
```

Orientation analysis

In addition to the analysis of the interaction energies, the relative orientations of the components are investigated. The first step for this is reordering all trajectories, such that the assemblies are clustered. This can be done in one go using the following command:

```
for trj in */*.xtc; do echo 1 | gmx trjconv -pbc cluster -s
${trj%%.*}.tpr -f $trj -o ${trj%.xtc}-clus.xtc; done
```

The next step is determining the relative orientations over time for each of the simulations. This is done with a Gromacs’ style C program, called *doriana* (domain orientation analysis). Doriana first determines the internal reference frame for each domain/component from a reference structure. It then calculates the change in orientation from the start by least-squares fitting. The resulting orientations are used to determine the relative orientations between domains.
In the case of homodimers or related components, it is advisable to use the same starting orientation, such that the internal frames at the start are identical, and a symmetric binding will yield a 180 degree rotation. This already applies to our case here, and we construct a doriana reference by copying the starting structure twice with sed.

```
sed -n -e '/^ATOM/{s/^\(.\{21\}\)./\1></;H;s/></A/p;}'
  -e '${x;s/></B/g;p;}' WT.pdb > WT-doriref.pdb
```

To understand this better, the different sed elements are explained here:

`-e ‘/^ATOM/{…}’`	Apply the statements between curly braces to lines starting with ‘ATOM’

`s/^\(.\{21\}\)./\1></`	Substitute the first 21+1 characters from the line by the first 21+'><'
`H`	Store the line in the (H)old space
`s/></A/p`	Substitute the characters '><' in the pattern space by 'A' and print.
`-e '${…}'`	Execute the statements between curly braces at the end of the file.
`x`	Exchange the pattern space and the hold space, i.e., retrieve the data stored in the hold space.
`s/></B/g`	Substitute '><' for 'B' globally in the pattern space, resetting the chain ID in all lines.
`p`	Print the pattern space.

Effectively, this takes a PDB file with one chain, prints each line with chain label 'A' and afterwards prints each line again, but with chain label 'B', yielding a PDB file with two copies of the same chain. This file we use as reference for the orientation analysis, but first we construct a matching index file (the lines can be copied):

```
N=$(grep -c "^ATOM" WT.pdb)

{ echo "[ AB ]"; seq $((2*N)); echo "[ A ]"; seq $N; echo "[ B ]"; seq $((N+1)) $((2*N)); } > doriref.ndx
```

To process the whole set of data, we use a for loop again (the line can be copied):

```
for xtc in */*-clus.xtc; do (cd ${i%/*}; echo 2 1 2 | doriana -s ../oriref.pdb -n ../oriref.ndx -f ${xtc#*/} -o doriana.pdb -ext doriana.dat); done
```

This yields a large amount of data that is processed further in R with the R script doriana.r:

```
/path/to/DAFT/Analysis/doriana.r 'daft-*/' doriana.dat
```

The script calculates 2D kernel density estimates for the different combinations of the COM-COM distance δ and the angles α, β, ϕ, θ, ψ, which are shown in the following figure: