Supporting information for: Quantifying Asymmetry of Multimeric Proteins

Julian T. Brennecke and Bert L. de Groot*

Department of Theoretical and Computational Biophysics, Computational Biomolecular Dynamics Group, Max Planck Institute for Biophysical Chemistry, Göttingen, Germany

E-mail: bgroot@gwdg.de

User Manual Asympy

This manual provides detailed information on the usage of the provided tool. The package can be downloaded from *https://gitlab.gwdg.de/deGroot/asympy.git*. To run the programs, Python 2.7 with these packages is required:

- pylab
- mdtraj
- scipy
- optparse
- sklearn
- ipy_progressbar (optional)

First, the usage of the CSM calculation tool (*CSM.py*) with the individual subunit contributions is introduced. In the next part, the usage of the tool to calculate the symmetry measure according to the FAME algorithm (*FAME.py*) is demonstrated. To be able to calculate the FAME algorithm correctly, the trajectory and the functional input data have to be prepared to get a symmetric reference motion (see figure S5). The preparation is done using the *prepare_files.py* tool.

For all examples provided, various files are used. All tools require a PDB structure file (*struct.pdb*), which has to be matched with a trajectory file containing an ensemble of structures, e.g. a MD trajectory (*traj.xtc*). The CSM and the preparative tools expect the sorting of the subunits to be either clockwise or counter-clockwise. However, if the sorting differs from this convention, it can be changed by giving the atom ranges of each subunit in (counter-)clockwise order. A tetramer with a clockwise labeling of the subunits 1, 3, 2, 4 with 10 atoms each could be analyzed using the parameter setting *-c '[[0,10],[20,30],[10,20],[30,40]]'*.

CSM.py

By the CSM tool the asymmetry for each frame of a trajectory is calculated:

```
CSM.py -f traj.xtc -s struct.pdb -n 4 -o asym.txt -e asym.xtc. -p
```

This command takes a trajectory with a matching PDB file with four subunits (*-n 4*) and calculates the CSM measure. This measure in combination with the measure for the individual subunits is written to the file *asym.txt*. A typical output is shown in table S1. Here

Table S1: *asym.txt* Example output of CSM tool.

# Frame	CSM overall,	CSM for subunits			
0	0.95	0.24	0.45	0.12	0.14
1	1.2	0.32	0.47	0.2	0.21
⋮	⋮	⋮	⋮	⋮	⋮

the first column is the frame number. The second column gives the overall CSM score. The columns starting from the third column are the partial CSM scores for the different subunits.

An additional result from the CSM calculation is the fully symmetric structure for each frame. These structures are saved into the *asym.xtc* file. The flag *-p* does a permutation optimization. For the permutation optimization residues with chemically identical atoms are identified and the naming of these atoms are chosen to minimize the asymmetry measure. To be able to use this method the input pdb file has to contain chain/subunit labels (A,B,C,...) and the residues of the subunits have to be labeled identically NOT continuously. A script is provided which changes the residue labels *prep_pdb.py*. In order for it to relabel the residues correctly the chain/subunit labels have to be provided as (A,B,C,...).

prepare_files.py

The prepare_files program prepares the input for the FAME algorithm. As described in the main manuscript, a symmetrized input is needed to create a symmetric PLS-FMA vector:

prepare_files.py -f traj.xtc -s struct.pdb -n 4 -d data.dat -x out.xtc -p out.dat.

This command takes the data file *data.dat*, the format should be similar to the example input in table S2, and the trajectory file. As a result, the extended and rotated trajectory according to figure S5 is written to *-x out.xtc*. The input data is copied into the output file *-p out.dat* to match the trajectory for the FAME tool.

Table S2: *data.dat* format for the input data file. Alternatively, the frame number can be given before the functional value.

functional value
0.1
0.12
0.15
⋮

FAME.py

After preparing the input files using the prepare_files program, the output can be used to run the FAME algorithm. For the FAME algorithm, the optimal number of PLS components have to be determined first to avoid overfitting. The determination is done by running the FAME program with a negative value for the number of PLS components, that tests components starting at one to the absolute of the given number:

FAME.py -f out.xtc -s struct.pdb -n 4 -d out.dat -c -10 -o comp.dat.

The command tries components one to ten and evaluates the correlation. The program automatically uses the first half of the input as training data and the second half of the data as cross-validation data. An example output (*-o comp.dat*) is illustrated in figure S3. In this example, three components would be a good choice.
Consequently, the FAME program can be run with three PLS components:

FAME.py -f out.xtc -s struct.pdb -n 4 -d out.dat -c 3 -w model.dat -o contributions.dat -e extremes_ew.pdb -p extremes.pdb.

As an input for FAME, the output files of the prepare_files program are used. The output is the prediction for the functional value $\tilde{f}$ (*-w model.dat*). To get an idea on which parts are predicted well and which parts are not, the prediction $\tilde{f}$ can be visualized with the input data f (see figure S5). Note that for the final FAME algorithm the data set is not split into training and cross-validation data blocks but all data is used to build an optimal model. Furthermore, the contributions C_j for each of the subunits are written to *-o contributions.dat*. The PDB file *-p extremes.pdb* gives a graphical representation of the reference motion and *-e extremes_ ew.pdb* gives the ensemble weighted representation of the motion (refer to Hub and de Groot for further explanation).

The PLS-FMA algorithm is insensitive to the absolute values of the functional input data. However, for the decomposition of the PLS-FMA prediction into the subunit components and especially for the contribution calculation, the selecteion of the absolute values of the functional input is crucial. As $\underline{C}_j = \underline{\tilde{f}}_j / \underline{\tilde{f}}$ it is clear that changing the sign in $\underline{\tilde{f}}_j$ or $\underline{\tilde{f}}$ would require the other value to change its sign accordingly to keep $\underline{C}_j$ in the range of $0 \leq \underline{C}_j \leq 1$, which is required for this measure. Thus applying a linear transformation on the functional input data ($\underline{f}$) might be required to get a reasonable prediction for the contributions. The algorithm would apply no value to $\underline{C}_j$ if it is outside the range. Therefore, the number of missing values combined with the expected continuity of the contributions $\underline{C}_j$ (sudden changes in asymmetry are not expected for a continuous simulation) can be used to estimate the need for a linear transformation. If a linear transformation is applied, the missing values can be used to evaluate the improvement of the results through the transformation.

Note that the current implementation reassigns subunit labels and minimizes the RMSD to a given reference to achieve symmetry correction. However, an alternative (and mathematically more correct but computationally significantly more demanding) method is the calculation of the correct rotation axis followed by a rotation around this, similar to the

CSM approach. (See SI figure 4 as an example for TTR.)

Supplementary Figures

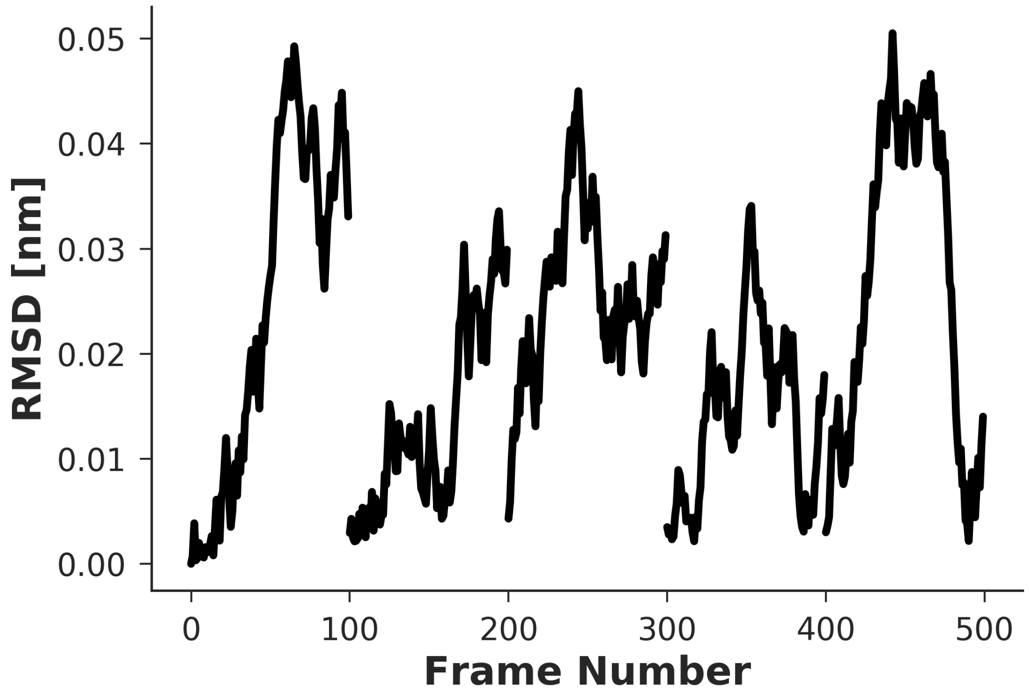

Figure S1: **RMSD KcsA** The RMSD of the trajectory of KcsA in reference to the symmetric crystal structure demonstrates how small the introduced asymmetric motions actually are.

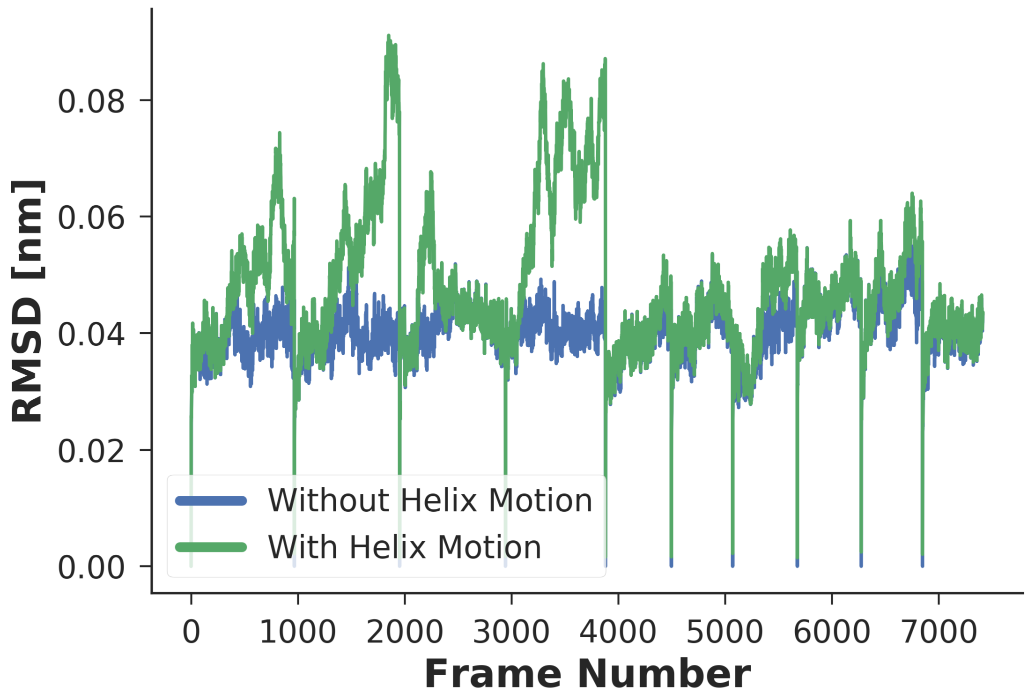

Figure S2: **RMSD TREK-2** The RMSD of the trajectory of TREK-2 comparing the structure of the original trajectory (blue line) and the trajectory after introducing the helix motion (green line).

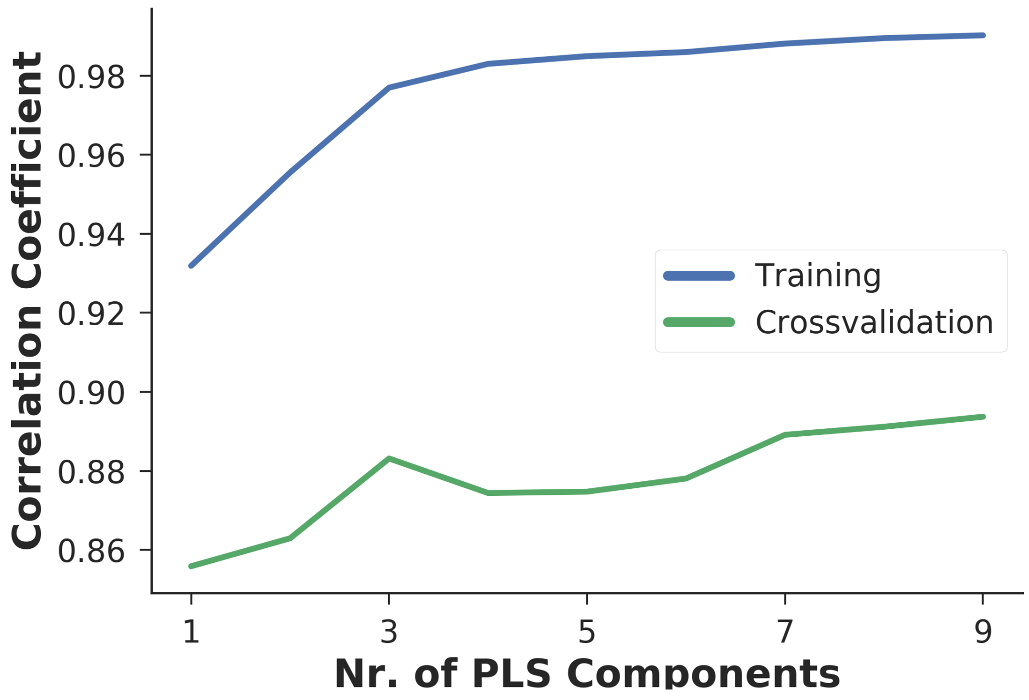

Figure S3: **PLS component estimation** To perform the PLS-FMA analysis, the correct number of PLS components has to be determined previously. Different PLS components have to be tested and the correlation between the functional input data and the model created with this number of components is calculated. The correlation for the training data is shown in blue and the correlation for the crossvalidation is shown in green.

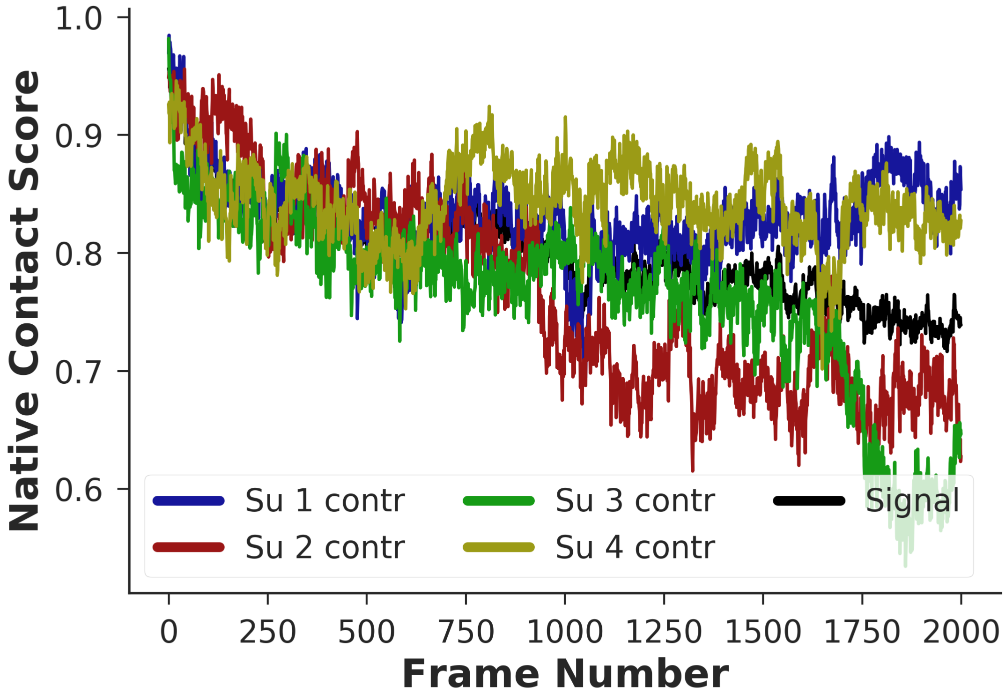

Figure S4: **TTR contacts** The black signal line in the background represents the overall contact score of the TTR simulations by calculating the native contact score based on the subunits independently (colored lines).

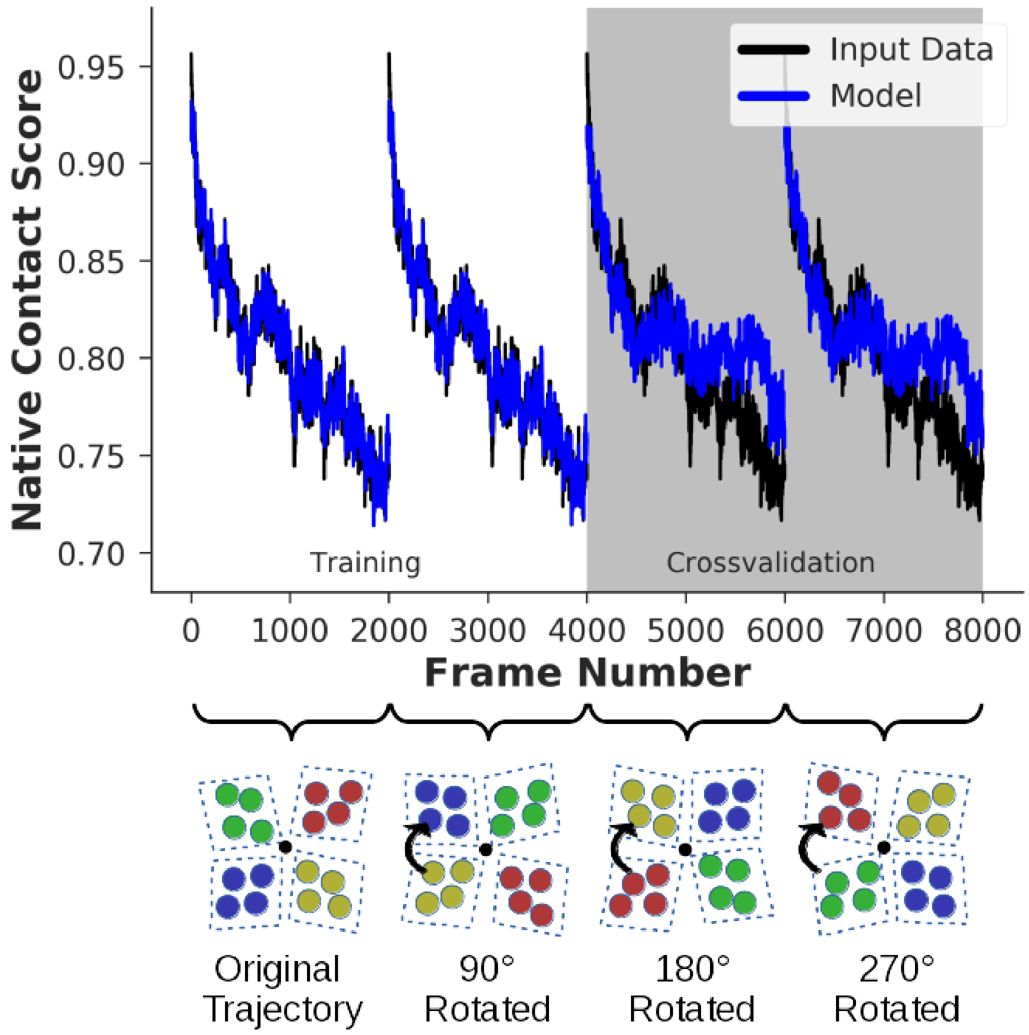

Figure S5: **TTR rotation scheme** To symmetrize the reference motion, the input ensemble is symmetrized by using all possible assignments of the labels.Thus the functional values are appended such that the overall dataset includes as many times the same functional value as subunits in the protein. Furthermore, the trajectory itself is duplicated by the same number. In the first repeat of the trajectory all subunits are rotated by 360 degrees divided by the number of subunits. The next repeat will be rotated by twice that and so on.
The input data (black line) is split in two halves where the first halve (white background) is used to train the model (blue line). For crossvalidation all input data are compared to the created model (gray background).