homelette

Philipp Junk, Christina Kiel

Sep 27, 2023

CONTENTS

homelette is a Python package offering a unified interface to different software for generating and evaluating homology models. This enables users to easily assemble custom homology modelling pipelines. homelette is extensively documented, lightweight and easily extendable.

If you use homelette in your research, please cite the following article:

> Philipp Junk, Christina Kiel, HOMELETTE: a unified interface to homology modelling software, Bioinformatics, 2021;, btab866, https://doi.org/10.1093/bioinformatics/btab866

CHAPTER

ONE

SETTING UP HOMELETTE

This section explains how to set homelette up on your system. homelette is available on GitHub and PyPI. The easiest option to work with homelette is to use a docker container that has all dependencies already installed.

1.1 Installation

While installing the homelette base package is easy, some of its dependencies are quite complicated to install. If you just want to try out homelette, we would encourage you to start with our *Docker image* which has all these dependencies already installed.

1.1.1 homelette

homelette is easily available through our GitHub page (GitHub homelette) or through PyPI.

```
python3 -m pip install homelette
```

Please be aware that homelette requires Python 3.6.12 or newer.

1.1.2 Modelling and Evaluation Software

homelette doesn't have model generating or model evaluating capabilities on its own. Instead, it provides a unified interface to other software with these capabilities.

None of the tools and packages listed here are "hard" dependencies in the way that homelette won't work if you have them not installed. Actually, you can still use homelette without any of these packages. However, none of the pre-implemented building blocks would work that way. It is therefore strongly recommended that, in order to get the most out of homelette, to install as many of these tools and packages.

Again, we want to mention that we have prepared a *Docker image* that contains all of these dependencies, and we strongly recommend that you start there if you want to find out if homelette is useful for you.

MODELLER

Installation instructions for MODELLER can be found here: Installation MODELLER. Requires a license key (freely available for academic research) which can be requested here: License MODELLER.

altMOD

altMOD can be installed from here: GitHub altMOD. Please make sure that the altMOD directory is in your Python path.

ProMod3

ProMod3 has to be compiled from source, instructions can be found here: Installation ProMod3. Main dependencies are OpenMM (available through conda or from source) and OpenStructure (available here: Installation OpenStructure).

QMEAN

QMEAN has be compiled from source, instructions can be found here: GitLab QMEAN. Has the same dependencies as ProMod3.

SOAP potential

While the code for evaluation with SOAP is part of MODELLER, some files for SOAP are not included in the standard release and have to be downloaded separately. The files are available here Download SOAP.

Specifically, you need to have `soap_protein_od.hdf5` available in your modlib directory. The modlib directory is placed at `/usr/lib/modellerXX_XX/modlib/` if installed with `dpkg` or at `anaconda/envs/yourenv/lib/modellerXX-XX/modlib/` if installed with `conda`. These paths might be different on your system.

MolProbity

Installation instructures for MolProbity are available here: Github MolProbity. Please make sure that after installation, `phenix.molprobity` is in your path.

1.1.3 Alignment Software

homelette is, given a query sequence, to automatically search for potential templates and generate sequence alignments. This requires additional software.

Clustal Omega

Clustal Omega is a light and powerful multiple sequence alignment tool. It can be obtained as source code or precompiled from here: Clustal Omega webpage. Please make sure that after installation, `clustalo` is in your path.

HHSuite3

Installation instructions for HHSuite3 are available here: Github HHSuite. Please make sure that after installation, `hhblits` is in your path.

Databases for HHSuite3

Information on how to obtain the databases is available here: Github HHSuite. The PDB70 database (~25 GB download, ~65 GB extracted) is required for using HHSuite in homelette, while the UniRef30 database (50~ GB download, ~170 GB extracted) is optional. Please make sure that after downloading and extracting the databases that they are in one folder and are named `pdb70_*` and `UniRef30_*`, respectively.

1.2 Docker

One of the best ways to share software and software environments in a reproducible way is using Docker. We have prepared a way to set up a docker image containing *homelette* and all its dependencies.

Due to the way how MODELLER licenses need to be aquired for each individual user, a two step process to setting up the docker image is required:

1. The template for the docker image that contains everything except a MODELLER license key will be pulled from DockerHub.
2. With a valid MODELLER license key, a local image with all dependencies working will be generated.

Note: Due to the numerous dependencies installed in the Docker image, please be aware that the image is quite big (~10 GB).

Note: The databases required for using HHSuite3 are not included in the docker container due to their size.

The following sections will explain how to set up and use the docker image.

1.2.1 Setting up the docker image

A bash script (`construct_homelette_image.sh` found in `homelette/docker/`) has been provided which automatically pulls the latest version of the homelette_template image from DockerHub and then attempts to construct the local homelette image with the given MODELLER license key. After downloading the script from Github, run

```
./construct_homelette_image.sh "YOUR MODELLERKEY HERE"
```

Warning: The local image created by this contains your MODELLLER license key. Similarly, as you would not send your license key to others, please do not share this image with other people, including on DockerHub.

The script will fail and no local image will be constructed if the license key is not accepted by the MODELLER version in the container.

1.2.2 Accessing the docker image

After constructing the local *homelette* docker image, you can access the docker image as every other as well.

```
docker run --rm -it homelette
```

However, to make access a bit simpler, we have written a bash script (`homelete.sh` found in `homelette/docker/`) to provide different options and modes to access the docker image. There are four different modes available:

- `./homelette.sh -m tutorial`: This opens an interactive Jupyter Lab version of the tutorials.
- `./homelette.sh -m jupyterlab`: This opens an interactive Jupyter Lab session with access to homelette and all dependencies.
- `./homelette.sh -m interacive`: This opens an interactive Python interpreter session with access to homelette and all dependencies.
- `./homelette.sh -m script`: This allows the user to execute a Python script in the Docker container.

In addition, the script has the ability to make a number of directories from the host machine available to the container. Please check out `./homelette.sh -h` for more details. All containers generated by this script will be removed after termination.

CHAPTER

TWO

TUTORIALS

We have prepared a series of 7 tutorials which will teach the interested user everything about using the homelette package. This is a great place to get started with homelette.

For a more interactive experience, all tutorials are available as Jupyter Notebooks through our *Docker container*.

2.1 Tutorial 1: Basics

[1]:
```
import homelette as hm
```

2.1.1 Introduction

Welcome to the first tutorial on how to use the `homelette` package. In this example, we will generate homology models using both `modeller` [1,2] and `ProMod3` [3,4] and then evaluate them using the DOPE score [5].

`homelette` is a Python package that delivers a unified interface to various homology modelling and model evaluation software. It is also easily customizable and extendable. Through a series of 7 tutorials, you will learn how to work with `homelette` as well as how to extend and adapt it to your specific needs.

In tutorial 1, you will learn how to:

- Import an alignment.
- Generate homology models using a predefined routine with `modeller`.
- Generate homology models using a predefined routine with `ProMod3`.
- Evaluate these models.

In this example, we will generate a protein structure for the RBD domain of ARAF. ARAF is a RAF kinase important in MAPK signalling. As a template, we will choose a close relative of ARAF called BRAF, specifically the structure with the PDB code 3NY5.

All files necessary for running this tutorial are already prepared and deposited in the following directory: `homelette/example/data/`. If you execute this tutorial from `homelette/example/`, you don't have to adapt any of the paths.

`homelette` comes with an extensive documentation. You can either check out our online documentation, compile a local version of the documentation in `homelette/docs/` or use the `help()` function in Python.

2.1.2 Alignment

The basis for a good homology model is a good alignment between your target and your template(s). There are many ways to generate alignments. Depending on the scope of your project, you might want to generate extensive, high-quality multiple sequence alignments from annotated sequence libraries of your sequences of interest using specific software such as t-coffee [6,7], or get a web service such as HH-Pred [8,9] to search for potential templates and align them.

For this example, we have already provided an alignment for you.

`homelette` has its own `Alignment` class which is used to work with alignments. You can import alignments from different file types, write alignments to different file types, select a subset of sequences, calculate sequence identity and print the alignment to screen. For more information, please check out the documentation.

[2]:
```
# read in the alignment
aln = hm.Alignment('data/single/aln_1.fasta_aln')

# print to screen to check alignment
aln.print_clustal(line_wrap=70)
```

```
ARAF        ---GTVKVYLPNKQRTVVTVRDGMSVYDSLDKALKVRGLNQDCCVVYRLIKGRKTVTAWDTAIAPLDGEE
3NY5        HQKPIVRVFLPNKQRTVVPARCGVTVRDSLKKAL--RGLIPECCAVYRIQ---KKPIGWDTDISWLTGEE


ARAF        LIVEVL------
3NY5        LHVEVLENVPLT
```

The template aligns nicely to our target. We can also check how much sequence identity these two sequences share:

[3]:
```
# calculate identity
aln.calc_identity_target('ARAF')
```

[3]:
```
  sequence_1 sequence_2  identity
0       ARAF       3NY5     57.53
```

The two sequences share a high amount of sequence identity, which is a good sign that our homology model might be reliable.

`modeller` expects the sequences handed to it to be annotated to a minimal degree. It is usually a good idea to annotate any template given to `modeller` in addition to the required PDB identifier with beginning and end residues and chains. This can be done as such:

[4]:
```
# annotate the alignment
aln.get_sequence('ARAF').annotate(seq_type = 'sequence')
aln.get_sequence('3NY5').annotate(seq_type = 'structure',
                                  pdb_code = '3NY5',
                                  begin_res = '1',
                                  begin_chain = 'A',
                                  end_res = '81',
                                  end_chain = 'A')
```

For more information on the sequence annotation, please check the documentation.

2.1.3 Template Structures

For the sake of consistency, we recommend adjusting the residue count to start with residue 1 for each model and ignore missing residues. A good tool for handling PDB structures is `pdb-tools` (available here) [10].

2.1.4 Model Generation

After importing our alignment, checking it manually, calculating sequence identities and annotating the sequences, as well as taking about the templates we are using, we are now able to proceed with the model generation.

Before starting modelling and evaluation, we need to set up a `Task` object. The purpose of `Task` objects is to simplify the interface to modelling and evaluation methods. `Task` objects are alignment-specific and target-specific.

[5]:
```
# set up task object
t = hm.Task(
    task_name = 'Tutorial1',
    target = 'ARAF',
    alignment = aln,
    overwrite = True)
```

Upon initialization, the task object will check if there is a folder in the current working directory that corresponds to the given `task_name`. If no such folder is available, a new one will be created.

After initialization of the Task object, we can start with homology modelling. For this, we use the `execute_routine` function of the task object, which applies the chosen homology modelling method with the chosen target, alignment and template(s).

[6]:
```
# generate models with modeller
t.execute_routine(
    tag = 'example_modeller',
    routine = hm.routines.Routine_automodel_default,
    templates = ['3NY5'],
    template_location = './data/single')
```

It is possible to use the same `Task` object to create models from multiple different routine-template combinations.

[7]:
```
# generate models with promod3
t.execute_routine(
    tag = 'example_promod3',
    routine = hm.routines.Routine_promod3,
    templates = ['3NY5'],
    template_location = './data/single')
```

2.1.5 Model Evaluation

Similarly to modelling, model evaluation is performed through the `evaluate_models` function of the `Task` object. This function is an easy interface to perform one or more evaluation methods on the models deposited in the task object.

[8]:
```
# perform evaluation
t.evaluate_models(hm.evaluation.Evaluation_dope)
```

The `Task.get_evaluation` function retrieves the evaluation for all models in the `Task` object as a `pandas` data frame.

[9]:
```
t.get_evaluation()
```

[9]:
```
                    model               tag            routine         dope  \
0  example_modeller_1.pdb  example_modeller  automodel_default -7274.457520
1   example_promod3_1.pdb   example_promod3            promod3 -7642.868652

   dope_z_score
0     -1.576995
1     -1.934412
```

For more details on the available evaluation methods please check out the documentation and the **Tutorial 3**.

2.1.6 Further Reading

Congratulations, you are now familiar with the basic functionality of `homelette`. You can now load an alignment, are familiar with the `Task` object and can perform homology modelling and evaluate your models.

Please note that there are other, more advanced tutorials, which will teach you more about how to use `homelette`:

- **Tutorial 2**: Learn more about already implemented routines for homology modelling.
- **Tutorial 3**: Learn about the evaluation metrics available with `homelette`.
- **Tutorial 4**: Learn about extending `homelette`'s functionality by defining your own modelling routines and evaluation metrics.
- **Tutorial 5**: Learn about how to use parallelization in order to generate and evaluate models more efficiently.
- **Tutorial 6**: Learn about modelling protein complexes.
- **Tutorial 7**: Learn about assembling custom pipelines.
- **Tutorial 8**: Learn about automated template identification, alignment generation and template processing.

2.1.7 References

[1] Šali, A., & Blundell, T. L. (1993). Comparative protein modelling by satisfaction of spatial restraints. Journal of Molecular Biology, 234(3), 779–815. https://doi.org/10.1006/jmbi.1993.1626

[2] Webb, B., & Sali, A. (2016). Comparative Protein Structure Modeling Using MODELLER. Current Protocols in Bioinformatics, 54(1), 5.6.1-5.6.37. https://doi.org/10.1002/cpbi.3

[3] Biasini, M., Schmidt, T., Bienert, S., Mariani, V., Studer, G., Haas, J., Johner, N., Schenk, A. D., Philippsen, A., & Schwede, T. (2013). OpenStructure: An integrated software framework for computational structural biology. Acta Crystallographica Section D: Biological Crystallography, 69(5), 701–709. https://doi.org/10.1107/S0907444913007051

[4] Studer, G., Tauriello, G., Bienert, S., Biasini, M., Johner, N., & Schwede, T. (2021). ProMod3—A versatile homology modelling toolbox. PLOS Computational Biology, 17(1), e1008667. https://doi.org/10.1371/JOURNAL.PCBI.1008667

[5] Shen, M., & Sali, A. (2006). Statistical potential for assessment and prediction of protein structures. Protein Science, 15(11), 2507–2524. https://doi.org/10.1110/ps.062416606

[6] Notredame, C., Higgins, D. G., & Heringa, J. (2000). T-coffee: A novel method for fast and accurate multiple sequence alignment. Journal of Molecular Biology, 302(1), 205–217. https://doi.org/10.1006/jmbi.2000.4042

[7] Wallace, I. M., O'Sullivan, O., Higgins, D. G., & Notredame, C. (2006). M-Coffee: Combining multiple sequence alignment methods with T-Coffee. Nucleic Acids Research, 34(6), 1692–1699. https://doi.org/10.1093/nar/gkl091

[8] Söding, J., Biegert, A., & Lupas, A. N. (2005). The HHpred interactive server for protein homology detection and structure prediction. Nucleic Acids Research, 33(suppl_2), W244–W248. https://doi.org/10.1093/NAR/GKI408

[9] Zimmermann, L., Stephens, A., Nam, S. Z., Rau, D., Kübler, J., Lozajic, M., Gabler, F., Söding, J., Lupas, A. N., & Alva, V. (2018). A Completely Reimplemented MPI Bioinformatics Toolkit with a New HHpred Server at its Core. Journal of Molecular Biology, 430(15), 2237–2243. https://doi.org/10.1016/J.JMB.2017.12.007

[10] Rodrigues, J. P. G. L. M., Teixeira, J. M. C., Trellet, M., & Bonvin, A. M. J. J. (2018). pdb-tools: a swiss army knife for molecular structures. F1000Research 2018 7:1961, 7, 1961. https://doi.org/10.12688/f1000research.17456.1

2.1.8 Session Info

[10]:
```
# session info
import session_info
session_info.show(html = False, dependencies = True)
```

```
-----
homelette           1.4
pandas              1.5.3
session_info        1.0.0
-----
PIL                 7.0.0
altmod              NA
anyio               NA
asttokens           NA
attr                19.3.0
babel               2.12.1
backcall            0.2.0
certifi             2022.12.07
chardet             3.0.4
charset_normalizer  3.1.0
comm                0.1.2
cycler              0.10.0
cython_runtime      NA
dateutil            2.8.2
debugpy             1.6.6
decorator           4.4.2
executing           1.2.0
fastjsonschema      NA
idna                3.4
importlib_metadata  NA
importlib_resources NA
ipykernel           6.21.3
ipython_genutils    0.2.0
jedi                0.18.2
jinja2              3.1.2
json5               NA
jsonschema          4.17.3
jupyter_events      0.6.3
jupyter_server      2.4.0
jupyterlab_server   2.20.0
kiwisolver          1.0.1
markupsafe          2.1.2
matplotlib          3.1.2
```

(continues on next page)

(continued from previous page)

```
modeller            10.4
more_itertools      NA
mpl_toolkits        NA
nbformat            5.7.3
numexpr             2.8.4
numpy               1.24.2
ost                 2.3.1
packaging           20.3
parso               0.8.3
pexpect             4.8.0
pickleshare         0.7.5
pkg_resources       NA
platformdirs        3.1.1
prometheus_client   NA
promod3             3.2.1
prompt_toolkit      3.0.38
psutil              5.5.1
ptyprocess          0.7.0
pure_eval           0.2.2
pydev_ipython       NA
pydevconsole        NA
pydevd              2.9.5
pydevd_file_utils   NA
pydevd_plugins      NA
pydevd_tracing      NA
pygments            2.14.0
pyparsing           2.4.6
pyrsistent          NA
pythonjsonlogger    NA
pytz                2022.7.1
qmean               NA
requests            2.28.2
rfc3339_validator   0.1.4
rfc3986_validator   0.1.1
send2trash          NA
sitecustomize       NA
six                 1.12.0
sniffio             1.3.0
stack_data          0.6.2
swig_runtime_data4  NA
tornado             6.2
traitlets           5.9.0
urllib3             1.26.15
wcwidth             NA
websocket           1.5.1
yaml                6.0
zipp                NA
zmq                 25.0.1
-----
IPython             8.11.0
jupyter_client      8.0.3
jupyter_core        5.2.0
```

(continues on next page)

(continued from previous page)

```
jupyterlab          3.6.1
notebook            6.5.3
-----
Python 3.8.10 (default, Nov 14 2022, 12:59:47) [GCC 9.4.0]
Linux-4.15.0-206-generic-x86_64-with-glibc2.29
-----
Session information updated at 2023-03-15 23:34
```

2.2 Tutorial 2: Modelling

[1]:
```
import os

import homelette as hm
```

2.2.1 Introduction

Welcome to the second tutorial for `homelette`. In this tutorial, we will further explore the already implemented method to generate homology models.

Currently, the following software packages for generating homology models have been integrated in the `homelette` homology modelling interface:

- `modeller`: A robust package for homology modelling with a long history which is widely used [1,2]
- `altmod`: A modification to the standard `modeller` modelling procedure that has been reported to increase the quality of models [3]
- `ProMod3`: The modelling engine behind the popular SwissModel web platform [4,5]

Specifically, the following routines are implemented in `homelette`. For more details on the individual routines, please check the documentation or their respective docstring.

- `routines.Routine_automodel_default`
- `routines.Routine_automodel_slow`
- `routines.Routine_altmod_default`
- `routines.Routine_altmod_slow`
- `routines.Routine_promod3`

In this example, we will generate models for the RBD domain of ARAF. ARAF is a RAF kinase important in MAPK signalling. As a template, we will choose a close relative of ARAF called BRAF, specifically the structure with the PDB code 3NY5.

All files necessary for running this tutorial are already prepared and deposited in the following directory: `homelette/example/data/`. If you execute this tutorial from `homelette/example/`, you don't have to adapt any of the paths.

`homelette` comes with an extensive documentation. You can either check out our online documentation, compile a local version of the documentation in `homelette/docs/` with `sphinx` or use the `help()` function in Python.

2.2.2 Alignment

For this tutorial, we will use the same alignment and template as for **Tutorial 1**.

[2]:
```
# read in the alignment
aln = hm.Alignment('data/single/aln_1.fasta_aln')

# print to screen to check alignment
aln.print_clustal(line_wrap=70)
```

```
ARAF        ---GTVKVYLPNKQRTVVTVRDGMSVYDSLDKALKVRGLNQDCCVVYRLIKGRKTVTAWDTAIAPLDGEE
3NY5        HQKPIVRVFLPNKQRTVVPARCGVTVRDSLKKAL--RGLIPECCAVYRIQ---KKPIGWDTDISWLTGEE


ARAF        LIVEVL------
3NY5        LHVEVLENVPLT
```

[3]:
```
# annotate the alignment
aln.get_sequence('ARAF').annotate(seq_type = 'sequence')
aln.get_sequence('3NY5').annotate(seq_type = 'structure',
                                  pdb_code = '3NY5',
                                  begin_res = '1',
                                  begin_chain = 'A',
                                  end_res = '81',
                                  end_chain = 'A')
```

2.2.3 Model Generation using `routines`

The building blocks in `homelette` that take care of model generation are called Routines. There is a number of predefined routines, and it is also possible to construct custom routines (see **Tutorial 4**). Every routine in `homelette` expects a number of identical arguments, while some can have a few optional ones as well.

[4]:
```
?hm.routines.Routine_automodel_default
```

```
Init signature:
hm.routines.Routine_automodel_default(
    alignment: Type[ForwardRef('Alignment')],
    target: str,
    templates: Iterable,
    tag: str,
    n_threads: int = 1,
    n_models: int = 1,
) -> None
Docstring:
Class for performing homology modelling using the automodel class from
modeller with a default parameter set.

Parameters
----------
alignment : Alignment
    The alignment object that will be used for modelling
```

(continues on next page)

(continued from previous page)

```
target : str
    The identifier of the protein to model
templates : Iterable
    The iterable containing the identifier(s) of the template(s) used
    for the modelling
tag : str
    The identifier associated with a specific execution of the routine
n_threads : int
    Number of threads used in model generation (default 1)
n_models : int
    Number of models generated (default 1)

Attributes
----------
alignment : Alignment
    The alignment object that will be used for modelling
target : str
    The identifier of the protein to model
templates : Iterable
    The iterable containing the identifier(s) of the template(s) used for
    the modelling
tag : str
    The identifier associated with a specific execution of the routine
n_threads : int
    Number of threads used for model generation
n_models : int
    Number of models generated
routine : str
    The identifier associated with a specific routine
models : list
    List of models generated by the execution of this routine

Raises
------
ImportError
    Unable to import dependencies

Notes
-----
The following modelling parameters can be set when initializing this
Routine object:

* n_models
* n_threads

The following modelling parameters are set for this class:

+----------------------+----------------------------------------+
| modelling            | value                                  |
| parameter            |                                        |
+======================+========================================+
| model_class          | modeller.automodel.automodel           |
```

(continues on next page)

(continued from previous page)

```
+----------------------+----------------------------------------+
| library_schedule     | modeller.automodel.autosched.normal    |
+----------------------+----------------------------------------+
| md_level             | modeller.automodel.refine.very_fast    |
+----------------------+----------------------------------------+
| max_var_iterations   | 200                                    |
+----------------------+----------------------------------------+
| repeat_optmization   | 1                                      |
+----------------------+----------------------------------------+
File:           /usr/local/src/homelette-1.4/homelette/routines.py
Type:           type
Subclasses:
```

The following arguments are required for all pre-defined routines:

- `alignment`: The alignment object used for modelling.
- `target`: The identifier of the target sequence in the alignment object
- `templates`: An iterable containing the identifier(s) of the templates for this modelling routine. `homelette` expects that templates are uniquely identified by their identifier in the alignment and in the template PDB file(s). Routines based on `modeller` work with one or multiple templates, whereas `Routine_promod3` only accepts a single template per run.
- `tag`: Each executed routine is given a tag which will be used to name the generated models.

In addition, pre-defined routines expect the template PDBs to be present in the current working directory.

The routine `Routine_automodel_default` has two optional arguments:

- `n_models`: the number of models that should be produced on this run, as routines based on `modeller` are able to produce an arbitary number of models.
- `n_threads`: enable mulit-threading for the execution of this routine. For more information on parallelization in `homelette`, please check out **Tutorial 5**.

While it is generally recommended to execute routines using `Task` objects (see next section), it is also possible to execute them directly. For doing this, since the template file has to be in the curent working directory, we quickly change working directory to a prepared directory where we can execute the routine (this code assumes that your working directory is `homelette/examples`.

[5]:
```
# change directory
os.chdir('data/single')
# print content of directory to screen
print('Files before modelling:\n' + ' '.join(os.listdir()) + '\n\n')

# perform modelling
routine = hm.routines.Routine_automodel_default(
    alignment=aln,
    target='ARAF',
    templates=['3NY5'],
    tag='model')
routine.generate_models()

print('Files after modelling:\n' + ' '.join(os.listdir()) + '\n')
```

(continues on next page)

(continued from previous page)

```
# remove model
os.remove('model_1.pdb')

# change back to tutorial directory
os.chdir('../..')
```

```
Files before modelling:
3NY5.pdb aln_1.fasta_aln 4G0N.pdb


Files after modelling:
model_1.pdb 3NY5.pdb aln_1.fasta_aln 4G0N.pdb
```

2.2.4 Model Generation using `Task` and `routines`

`homelette` has `Task` objects that allow for easier use of Routines and Evaluations (see also **Tutorial 3**). `Task` objects help to direct and organize modelling pipelines. It is strongly recommended to use `Task` objects to execute routines and evaluations.

For more information on `Task` objects, please check out the documentation or **Tutorial 1**.

[6]:
```
# set up task object
t = hm.Task(
    task_name = 'Tutorial2',
    target = 'ARAF',
    alignment = aln,
    overwrite = True)
```

Using the `Task` object, we can now begin to generate our models with different routines using the `Task.execute_routine` method.

[7]:
```
?hm.Task.execute_routine
```

```
Signature:
hm.Task.execute_routine(
    self,
    tag: str,
    routine: Type[ForwardRef('routines.Routine')],
    templates: Iterable,
    template_location: str = '.',
    **kwargs,
) -> None
Docstring:
Generates homology models using a specified modelling routine

Parameters
----------
tag : str
    The identifier associated with this combination of routine and
    template(s). Has to be unique between all routines executed by the
```

(continues on next page)

(continued from previous page)

```
    same task object
routine : Routine
    The routine object used to generate the models
templates : list
    The iterable containing the identifier(s) of the template(s) used
    for model generation
template_location : str, optional
    The location of the template PDB files. They should be named
    according to their identifiers in the alignment (i.e. for a
    sequence named "1WXN" to be used as a template, it is expected that
    there will be a PDB file named "1WXN.pdb" in the specified template
    location (default is current working directory)
**kwargs
    Named parameters passed directly on to the Routine object when the
    modelling is performed. Please check the documentation in order to
    make sure that the parameters passed on are available with the
    Routine object you intend to use

Returns
-------
None
File:      /usr/local/src/homelette-1.4/homelette/organization.py
Type:      function
```

As we can see, `Task.execute_routine` expects a number of arguments from the user:

- `tag`: Each executed routine is given a tag which will be used to name the generated models. This is useful for differentiating between different routines executed by the same `Task`, for example if different templates are used.
- `routine`: Here the user can set which routine will be used for generating the homology model(s), arguably the most important setting.
- `templates`: An iterable containing the identifier(s) of the templates for this modelling routine. `homelette` expects that templates are uniquely identified by their identifier(s) in the alignment and in the template location.
- `template_location`: The folder where the PDB file(s) used as template(s) are found.

We are generating some models with the pre-defined routines of `homelette`:

[8]:
```
# model generation with modeller
t.execute_routine(
    tag = 'example_modeller',
    routine = hm.routines.Routine_automodel_default,
    templates = ['3NY5'],
    template_location = './data/single')

# model generation with altmod
t.execute_routine(
    tag = 'example_altmod',
    routine = hm.routines.Routine_altmod_default,
    templates = ['3NY5'],
    template_location = './data/single')

# model generation with promod3
t.execute_routine(
```

(continues on next page)

(continued from previous page)

```
    tag = 'example_promod3',
    routine = hm.routines.Routine_promod3,
    templates = ['3NY5'],
    template_location = './data/single')
```

As mentioned before, some modelling routines have optional arguments, such as `n_models` for `Routine_autmodel_default`. We can pass these optional arguments to `Task.execute_routine` which passes them on the routine selected:

[9]:
```
# multiple model generation with altmod
t.execute_routine(
    tag = 'example_modeller_more_models',
    routine = hm.routines.Routine_automodel_default,
    templates = ['3NY5'],
    template_location = './data/single',
    n_models = 10)
```

Models generated using `Task` objects are stored as `Model` objects in the `Task`:

[10]:
```
t.models
```

[10]:
```
[<homelette.organization.Model at 0x7f421f7f9280>,
 <homelette.organization.Model at 0x7f421f7cf7f0>,
 <homelette.organization.Model at 0x7f421f8f4370>,
 <homelette.organization.Model at 0x7f421f8dfca0>,
 <homelette.organization.Model at 0x7f421f8df2e0>,
 <homelette.organization.Model at 0x7f421f8da2b0>,
 <homelette.organization.Model at 0x7f421f8da400>,
 <homelette.organization.Model at 0x7f421f8da370>,
 <homelette.organization.Model at 0x7f421f806220>,
 <homelette.organization.Model at 0x7f421f806cd0>,
 <homelette.organization.Model at 0x7f421f806a00>,
 <homelette.organization.Model at 0x7f421f806f10>,
 <homelette.organization.Model at 0x7f421f806280>]
```

In conclusion, we have learned how to use a single `Task` object to generate models with different modelling routines. We have also learned how to pass optional arguments on to the executed routines.

In this example, the target, the alignment and the templates were kept identical. Varying the templates would be straight forward, under the condition that other templates are included in the alignment. For varying alignments and targets, new `Task` objects would need to be created. This is a design choice that is meant to encourage users to try out different routines or templates/template combinations. It is recommended when using different routines or multiple templates to indicate this using the `tag` argument of `Task.execute_routine` (i.e. `tag='automodel_3NY5'`). Similarly, using a single `Task` object for multiple targets or alignments is discouraged and we recommend to utilize multiple `Task` objects for these modelling approaches.

2.2.5 Further Reading

You are now familiar with model generation in `homelette`.

Please note that there are other tutorials, which will teach you more about how to use `homelette`:

- **Tutorial 1**: Learn about the basics of `homelette`.
- **Tutorial 3**: Learn about the evaluation metrics available with `homelette`.
- **Tutorial 4**: Learn about extending `homelette`'s functionality by defining your own modelling routines and evaluation metrics.
- **Tutorial 5**: Learn about how to use parallelization in order to generate and evaluate models more efficiently.
- **Tutorial 6**: Learn about modelling protein complexes.
- **Tutorial 7**: Learn about assembling custom pipelines.
- **Tutorial 8**: Learn about automated template identification, alignment generation and template processing.

2.2.6 References

[1] Šali, A., & Blundell, T. L. (1993). Comparative protein modelling by satisfaction of spatial restraints. Journal of Molecular Biology, 234(3), 779–815. https://doi.org/10.1006/jmbi.1993.1626

[2] Webb, B., & Sali, A. (2016). Comparative Protein Structure Modeling Using MODELLER. Current Protocols in Bioinformatics, 54(1), 5.6.1-5.6.37. https://doi.org/10.1002/cpbi.3

[3] Janson, G., Grottesi, A., Pietrosanto, M., Ausiello, G., Guarguaglini, G., & Paiardini, A. (2019). Revisiting the "satisfaction of spatial restraints" approach of MODELLER for protein homology modeling. PLoS Computational Biology, 15(12), e1007219. https://doi.org/10.1371/journal.pcbi.1007219

[4] Biasini, M., Schmidt, T., Bienert, S., Mariani, V., Studer, G., Haas, J., Johner, N., Schenk, A. D., Philippsen, A., & Schwede, T. (2013). OpenStructure: An integrated software framework for computational structural biology. Acta Crystallographica Section D: Biological Crystallography, 69(5), 701–709. https://doi.org/10.1107/S0907444913007051

[5] Studer, G., Tauriello, G., Bienert, S., Biasini, M., Johner, N., & Schwede, T. (2021). ProMod3—A versatile homology modelling toolbox. PLOS Computational Biology, 17(1), e1008667. https://doi.org/10.1371/JOURNAL.PCBI.1008667

2.2.7 Session Info

[11]:
```
# session info
import session_info
session_info.show(html = False, dependencies = True)
```

```
-----
homelette           1.4
session_info        1.0.0
-----
PIL                 7.0.0
altmod              NA
anyio               NA
asttokens           NA
attr                19.3.0
```

(continues on next page)

(continued from previous page)

```
babel                2.12.1
backcall             0.2.0
certifi              2022.12.07
chardet              3.0.4
charset_normalizer   3.1.0
comm                 0.1.2
cycler               0.10.0
cython_runtime       NA
dateutil             2.8.2
debugpy              1.6.6
decorator            4.4.2
executing            1.2.0
fastjsonschema       NA
idna                 3.4
importlib_metadata   NA
importlib_resources  NA
ipykernel            6.21.3
ipython_genutils     0.2.0
jedi                 0.18.2
jinja2               3.1.2
json5                NA
jsonschema           4.17.3
jupyter_events       0.6.3
jupyter_server       2.4.0
jupyterlab_server    2.20.0
kiwisolver           1.0.1
markupsafe           2.1.2
matplotlib           3.1.2
modeller             10.4
more_itertools       NA
mpl_toolkits         NA
nbformat             5.7.3
numexpr              2.8.4
numpy                1.24.2
ost                  2.3.1
packaging            20.3
pandas               1.5.3
parso                0.8.3
pexpect              4.8.0
pickleshare          0.7.5
pkg_resources        NA
platformdirs         3.1.1
prometheus_client    NA
promod3              3.2.1
prompt_toolkit       3.0.38
psutil               5.5.1
ptyprocess           0.7.0
pure_eval            0.2.2
pydev_ipython        NA
pydevconsole         NA
pydevd               2.9.5
pydevd_file_utils    NA
```

(continues on next page)

(continued from previous page)

```
pydevd_plugins      NA
pydevd_tracing      NA
pygments            2.14.0
pyparsing           2.4.6
pyrsistent          NA
pythonjsonlogger    NA
pytz                2022.7.1
qmean               NA
requests            2.28.2
rfc3339_validator   0.1.4
rfc3986_validator   0.1.1
send2trash          NA
sitecustomize       NA
six                 1.12.0
sniffio             1.3.0
stack_data          0.6.2
swig_runtime_data4  NA
tornado             6.2
traitlets           5.9.0
urllib3             1.26.15
wcwidth             NA
websocket           1.5.1
yaml                6.0
zipp                NA
zmq                 25.0.1
-----
IPython             8.11.0
jupyter_client      8.0.3
jupyter_core        5.2.0
jupyterlab          3.6.1
notebook            6.5.3
-----
Python 3.8.10 (default, Nov 14 2022, 12:59:47) [GCC 9.4.0]
Linux-4.15.0-206-generic-x86_64-with-glibc2.29
-----
Session information updated at 2023-03-15 23:35
```

2.3 Tutorial 3: Evaluation

[1]:
```
import homelette as hm
```

2.3.1 Introduction

Welcome to the third tutorial for `homelette`. In this tutorial, we will explore which evaluation metrics are implemented in `homelette` and how to use them.

Model evaluation is an important step in any homology modelling procedure. In most practical scenarios, you will end up with more than one possible model and have to decide which one is "best". Obtaining multiple models can be the result of trying out different templates or combinations of templates, different algorithms generating the models, or due to using an algorithm which can generate multiple models.

The following evaluation metrics are implemented in `homelette`:

- `evaluation.Evaluation_dope`: DOPE score from `modeller` [1]
- `evaluation.Evaluation_soap_protein`: SOAP score from `modeller` for the evaluation of single proteins [2]
- `evaluation.Evaluation_soap_pp`: SOAP score from `modeller` for the evaluation of protein complexes [2]
- `evaluation.Evaluation_qmean4`: QMEAN4 score [3,4]
- `evaluation.Evaluation_qmean6`: QMEAN6 score [3,4]
- `evaluation.Evaluation_qmeandisco`: QMEAN DisCo score [3,4,5]
- `evaluation.Evaluation_mol_probity`: MolProbity score for the structural evaluation of proteins [6,7,8]

All files necessary for running this tutorial are already prepared and deposited in the following directory: `homelette/example/data/`. If you execute this tutorial from `homelette/example/`, you don't have to adapt any of the paths.

`homelette` comes with an extensive documentation. You can either check out our online documentation, compile a local version of the documentation in `homelette/docs/` with `sphinx` or use the `help()` function in Python.

2.3.2 Model Generation

In order to have a few models to evaluate, we will briefly generate some models of ARAF as we have done in previous tutorials (please check **Tutorial 1** and **Tutorial 2** for more information on this part).

[2]:
```
# get alignment
aln = hm.Alignment('data/single/aln_1.fasta_aln')

# annotate the alignment
aln.get_sequence('ARAF').annotate(
    seq_type = 'sequence')
aln.get_sequence('3NY5').annotate(
    seq_type = 'structure',
    pdb_code = '3NY5',
    begin_res = '1',
    begin_chain = 'A',
    end_res = '81',
    end_chain = 'A')

# initialize task object
t = hm.Task(
    task_name = 'Tutorial3',
    target = 'ARAF',
    alignment = aln,
    overwrite = True)
```

(continues on next page)

(continued from previous page)

```
# generate models with modeller
t.execute_routine(
    tag = 'modeller',
    routine = hm.routines.Routine_automodel_default,
    templates = ['3NY5'],
    template_location = './data/single',
    n_models = 5)

# generate models with altmod
t.execute_routine(
    tag = 'altmod',
    routine = hm.routines.Routine_altmod_default,
    templates = ['3NY5'],
    template_location = './data/single',
    n_models = 5)
```

We now have generated 10 models, 5 generated with `modeller` and another 5 generated with `altmod`.

2.3.3 Model Evaluation using `evaluation`

Similar to routines, evaluations can be executed on their own, although it is recommended to use an interface through the `Task` object (see next section). For showcasing how an evaluation can be executed on its own, we will take one of the previously generated models as an example:

[3]:
```
# example model
model = t.models[0]
model
```

[3]:
```
<homelette.organization.Model at 0x7f7f681ae250>
```

Every `Model` object has an `Model.evaluation` attribute where information about the model and its evaluations are collected:

[4]:
```
model.evaluation
```

[4]:
```
{'model': 'modeller_1.pdb', 'tag': 'modeller', 'routine': 'automodel_default'}
```

After performing an evaluation, this dictionary will be updated with the results of the evaluation:

[5]:
```
hm.evaluation.Evaluation_dope(model, quiet=True)
model.evaluation
```

[5]:
```
{'model': 'modeller_1.pdb',
 'tag': 'modeller',
 'routine': 'automodel_default',
 'dope': -7216.8564453125,
 'dope_z_score': -1.5211129532811163}
```

The interface to evaluations is relatively simple:

[6]:
```
?hm.evaluation.Evaluation_dope
```

```
Init signature:
hm.evaluation.Evaluation_dope(
    model: Type[ForwardRef('Model')],
    quiet: bool = False,
) -> None
Docstring:
Class for evaluating a model with DOPE score.

Will dump the following entries to the model.evaluation dictionary:

* dope
* dope_z_score

Parameters
----------
model : Model
    The model object to evaluate
quiet : bool
    If True, will perform evaluation with suppressing stdout (default
    False). Needs to be False for running it asynchronously, as done
    when running Task.evaluate_models with multple cores

Attributes
----------
model : Model
    The model object to evaluate
output : dict
    Dictionary that all outputs will be dumped into

Raises
------
ImportError
    Unable to import dependencies

Notes
-----
DOPE is a staticial potential for the evaluation of homology models [1]_.
For further information, please check the modeller documentation or the
associated publication.

References
----------
.. [1] Shen, M., & Sali, A. (2006). Statistical potential for assessment
   and prediction of protein structures. Protein Science, 15(11),
   2507-2524. https://doi.org/10.1110/ps.062416606
File:           /usr/local/src/homelette-1.4/homelette/evaluation.py
Type:           type
Subclasses:
```

Evaluations take only two arguments: - `model`: A `Model` object - `quiet`: A boolean value determining whether any output to the console should be suppressed.

Unlike routines, evaluations are executed as soon as the object is initialized.

2.3.4 Model Evaluation using `Task` and `evaluation`

Using the interface to evaluations that is implemented in `Task` objects has several advantages: it is possible to evaluate multiple models with multiple evaluation metrics in one command. In addition, multi-threading can be enabled (see **Tutorial 5** for more details). The method to run evaluations with a `Task` object is called `evaluate_models`.

[7]:
```
?hm.Task.evaluate_models
```

```
Signature:
hm.Task.evaluate_models(
    self,
    *args: Type[ForwardRef('evaluation.Evaluation')],
    n_threads: int = 1,
) -> None
Docstring:
Evaluates models using one or multiple evaluation metrics

Parameters
----------
*args: Evaluation
    Evaluation objects that will be applied to the models
n_threads : int, optional
    Number of threads used for model evaluation (default is 1, which
    deactivates parallelization)

Returns
-------
None
File:      /usr/local/src/homelette-1.4/homelette/organization.py
Type:      function
```

[8]:
```
# running dope and soap at the same time
t.evaluate_models(hm.evaluation.Evaluation_dope,
                  hm.evaluation.Evaluation_soap_protein)
```

After running evaluations, output of all `Model.evaluation` can be compiled to a `pandas` data frame as such:

[9]:
```
t.get_evaluation()
```

[9]:
```
            model       tag            routine         dope  dope_z_score  \
0  modeller_1.pdb  modeller  automodel_default -7216.856445     -1.521113
1  modeller_2.pdb  modeller  automodel_default -7274.457520     -1.576995
2  modeller_3.pdb  modeller  automodel_default -7126.735352     -1.433681
3  modeller_4.pdb  modeller  automodel_default -7225.522461     -1.529520
4  modeller_5.pdb  modeller  automodel_default -7128.661621     -1.435550
5    altmod_1.pdb    altmod     altmod_default -8148.456055     -2.424912
6    altmod_2.pdb    altmod     altmod_default -8187.364258     -2.462659
7    altmod_3.pdb    altmod     altmod_default -8202.568359     -2.477409
8    altmod_4.pdb    altmod     altmod_default -8170.016602     -2.445829
9    altmod_5.pdb    altmod     altmod_default -8145.944336     -2.422475

   soap_protein
0 -44167.968750
1 -45681.269531
```

(continues on next page)

(continued from previous page)

```
2 -43398.992188
3 -42942.808594
4 -41418.894531
5 -53440.839844
6 -49991.304688
7 -53909.824219
8 -52208.402344
9 -50776.855469
```

2.3.5 On the combination of different evaluation metrics

Oftentimes it is useful to use different metrics to evaluate models. However, that produces the problem of having multiple metrics to base a decision on. There are multiple solutions to this problem, all of them with their own advantages and disadvantes. We want to mention the combination of z-scores of the different metrics and the combination of metrics by borda count.

In the following, we show how to combine multiple scores to one borda score. In short, borda count is an agglomeration of ranks in the different individual metrics to one score.

Note

Be careful because, for some metrics, lower values are better (DOPE, SOAP, MolProbity), but for others higher values are better (QMEAN).

[10]:
```
df = t.get_evaluation()
df = df.drop(labels=['routine', 'tag'], axis=1)

# rank by dope and soap
df['rank_dope'] = df['dope'].rank()
df['rank_soap'] = df['soap_protein'].rank()

# calculate points based on rank
n = df.shape[0]
df['points_dope'] = n - df['rank_dope']
df['points_soap'] = n - df['rank_soap']

df
```

[10]:

	model	dope	dope_z_score	soap_protein	rank_dope	rank_soap	points_dope	points_soap
0	modeller_1.pdb	-7216.856445	-1.521113	-44167.968750	8.0	8.0	2.0	2.0
1	modeller_2.pdb	-7274.457520	-1.576995	-45681.269531	6.0	6.0	4.0	4.0
2	modeller_3.pdb	-7126.735352	-1.433681	-43398.992188	10.0	10.0	0.0	0.0
3	modeller_4.pdb	-7225.522461	-1.529520	-42942.808594	7.0	7.0	3.0	3.0
4	modeller_5.pdb	-7128.661621	-1.435550	-41418.894531	9.0	9.0	1.0	1.0
5	altmod_1.pdb	-8148.456055	-2.424912	-53440.839844	4.0	4.0	5.0	5.0
6	altmod_2.pdb	-8187.364258	-2.462659	-49991.304688	2.0	2.0	7.0	7.0
7	altmod_3.pdb	-8202.568359	-2.477409	-53909.824219	1.0	1.0	8.0	8.0
8	altmod_4.pdb	-8170.016602	-2.445829	-52208.402344	3.0	3.0	6.0	6.0
9	altmod_5.pdb	-8145.944336	-2.422475	-50776.855469	5.0	5.0	4.0	4.0

(continues on next page)

(continued from previous page)

```
0        7.0          2.0          3.0
1        6.0          4.0          4.0
2        8.0          0.0          2.0
3        9.0          3.0          1.0
4       10.0          1.0          0.0
5        2.0          6.0          8.0
6        5.0          8.0          5.0
7        1.0          9.0          9.0
8        3.0          7.0          7.0
9        4.0          5.0          6.0
```

[11]:
```
# calculate borda score and borda rank
df['borda_score'] = df['points_dope'] + df['points_soap']
df['borda_rank'] = df['borda_score'].rank(ascending=False)

df = df.drop(labels=['rank_dope', 'rank_soap', 'points_dope', 'points_soap'], axis=1)
df.sort_values(by='borda_rank')
```

[11]:
```
            model         dope  dope_z_score  soap_protein  borda_score  \
7    altmod_3.pdb -8202.568359     -2.477409 -53909.824219         18.0
5    altmod_1.pdb -8148.456055     -2.424912 -53440.839844         14.0
8    altmod_4.pdb -8170.016602     -2.445829 -52208.402344         14.0
6    altmod_2.pdb -8187.364258     -2.462659 -49991.304688         13.0
9    altmod_5.pdb -8145.944336     -2.422475 -50776.855469         11.0
1  modeller_2.pdb -7274.457520     -1.576995 -45681.269531          8.0
0  modeller_1.pdb -7216.856445     -1.521113 -44167.968750          5.0
3  modeller_4.pdb -7225.522461     -1.529520 -42942.808594          4.0
2  modeller_3.pdb -7126.735352     -1.433681 -43398.992188          2.0
4  modeller_5.pdb -7128.661621     -1.435550 -41418.894531          1.0

   borda_rank
7         1.0
5         2.5
8         2.5
6         4.0
9         5.0
1         6.0
0         7.0
3         8.0
2         9.0
4        10.0
```

The model with the highest borda score or the lowest borda count is the best model according to the combination of DOPE and SOAP scores.

2.3.6 Further reading

You are now familiar with using the implemented evaluation features of `homelette`. For further reading, please consider checking out the other tutorials:

- **Tutorial 1**: Learn about the basics of `homelette`.
- **Tutorial 2**: Learn more about already implemented routines for homology modelling.
- **Tutorial 4**: Learn about extending `homelette`'s functionality by defining your own modelling routines and evaluation metrics.
- **Tutorial 5**: Learn about how to use parallelization in order to generate and evaluate models more efficiently.
- **Tutorial 6**: Learn about modelling protein complexes.
- **Tutorial 7**: Learn about assembling custom pipelines.
- **Tutorial 8**: Learn about automated template identification, alignment generation and template processing.

2.3.7 References

[1] Shen, M., & Sali, A. (2006). Statistical potential for assessment and prediction of protein structures. Protein Science, 15(11), 2507–2524. https://doi.org/10.1110/ps.062416606

[2] Dong, G. Q., Fan, H., Schneidman-Duhovny, D., Webb, B., Sali, A., & Tramontano, A. (2013). Optimized atomic statistical potentials: Assessment of protein interfaces and loops. Bioinformatics, 29(24), 3158–3166. https://doi.org/10.1093/bioinformatics/btt560

[3] Benkert, P., Tosatto, S. C. E., & Schomburg, D. (2008). QMEAN: A comprehensive scoring function for model quality assessment. Proteins: Structure, Function and Genetics, 71(1), 261–277. https://doi.org/10.1002/prot.21715

[4] Benkert, P., Biasini, M., & Schwede, T. (2011). Toward the estimation of the absolute quality of individual protein structure models. Bioinformatics, 27(3), 343–350. https://doi.org/10.1093/bioinformatics/btq662

[5] Studer, G., Rempfer, C., Waterhouse, A. M., Gumienny, R., Haas, J., & Schwede, T. (2020). QMEANDisCo-distance constraints applied on model quality estimation. Bioinformatics, 36(6), 1765–1771. https://doi.org/10.1093/bioinformatics/btz828

[6] Davis, I. W., Leaver-Fay, A., Chen, V. B., Block, J. N., Kapral, G. J., Wang, X., Murray, L. W., Arendall, W. B., Snoeyink, J., Richardson, J. S., & Richardson, D. C. (2007). MolProbity: all-atom contacts and structure validation for proteins and nucleic acids. Nucleic Acids Research, 35(suppl_2), W375–W383. https://doi.org/10.1093/NAR/GKM216

[7] Chen, V. B., Arendall, W. B., Headd, J. J., Keedy, D. A., Immormino, R. M., Kapral, G. J., Murray, L. W., Richardson, J. S., & Richardson, D. C. (2010). MolProbity: All-atom structure validation for macromolecular crystallography. Acta Crystallographica Section D: Biological Crystallography, 66(1), 12–21. https://doi.org/10.1107/S0907444909042073

[8] Williams, C. J., Headd, J. J., Moriarty, N. W., Prisant, M. G., Videau, L. L., Deis, L. N., Verma, V., Keedy, D. A., Hintze, B. J., Chen, V. B., Jain, S., Lewis, S. M., Arendall, W. B., Snoeyink, J., Adams, P. D., Lovell, S. C., Richardson, J. S., & Richardson, D. C. (2018). MolProbity: More and better reference data for improved all-atom structure validation. Protein Science, 27(1), 293–315. https://doi.org/10.1002/pro.3330

2.3.8 Session Info

[12]:
```
# session info
import session_info
session_info.show(html = False, dependencies = True)
```

```
-----
homelette           1.4
pandas              1.5.3
session_info        1.0.0
-----
PIL                 7.0.0
altmod              NA
anyio               NA
asttokens           NA
attr                19.3.0
babel               2.12.1
backcall            0.2.0
certifi             2022.12.07
chardet             3.0.4
charset_normalizer  3.1.0
comm                0.1.2
cycler              0.10.0
cython_runtime      NA
dateutil            2.8.2
debugpy             1.6.6
decorator           4.4.2
executing           1.2.0
fastjsonschema      NA
idna                3.4
importlib_metadata  NA
importlib_resources NA
ipykernel           6.21.3
ipython_genutils    0.2.0
jedi                0.18.2
jinja2              3.1.2
json5               NA
jsonschema          4.17.3
jupyter_events      0.6.3
jupyter_server      2.4.0
jupyterlab_server   2.20.0
kiwisolver          1.0.1
markupsafe          2.1.2
matplotlib          3.1.2
modeller            10.4
more_itertools      NA
mpl_toolkits        NA
nbformat            5.7.3
numexpr             2.8.4
numpy               1.24.2
ost                 2.3.1
packaging           20.3
parso               0.8.3
```

(continues on next page)

(continued from previous page)

```
pexpect             4.8.0
pickleshare         0.7.5
pkg_resources       NA
platformdirs        3.1.1
prometheus_client   NA
promod3             3.2.1
prompt_toolkit      3.0.38
psutil              5.5.1
ptyprocess          0.7.0
pure_eval           0.2.2
pydev_ipython       NA
pydevconsole        NA
pydevd              2.9.5
pydevd_file_utils   NA
pydevd_plugins      NA
pydevd_tracing      NA
pygments            2.14.0
pyparsing           2.4.6
pyrsistent          NA
pythonjsonlogger    NA
pytz                2022.7.1
qmean               NA
requests            2.28.2
rfc3339_validator   0.1.4
rfc3986_validator   0.1.1
send2trash          NA
sitecustomize       NA
six                 1.12.0
sniffio             1.3.0
stack_data          0.6.2
swig_runtime_data4  NA
tornado             6.2
traitlets           5.9.0
urllib3             1.26.15
wcwidth             NA
websocket           1.5.1
yaml                6.0
zipp                NA
zmq                 25.0.1
-----
IPython             8.11.0
jupyter_client      8.0.3
jupyter_core        5.2.0
jupyterlab          3.6.1
notebook            6.5.3
-----
Python 3.8.10 (default, Nov 14 2022, 12:59:47) [GCC 9.4.0]
Linux-4.15.0-206-generic-x86_64-with-glibc2.29
-----
Session information updated at 2023-03-15 23:37
```

2.4 Tutorial 4: Extending homelette

[1]:
```
import homelette as hm

import contextlib
import glob
import os.path
import sys

from modeller import environ, Selection
from modeller.automodel import LoopModel
```

2.4.1 Introduction

Welcome to the forth tutorial on `homelette`. In this tutorial, we will discuss how to implement custom building blocks, either for generating or for evaluating models. These custom building blocks can be integrated in homology modelling pipelines.

This is probably the most important tutorial in the series. After this tutorial, you will be able to implement your own routines into the `homelette` framework, which gives you complete control over the homology modelling pipelines you want to establish!

Please note that we encourage users to share custom routines and evaluation metrics if they think they might be useful for the community. In our online documentation, there is a dedicated section for these contributions. If you are interested, please contact us on GitHub or via email.

2.4.2 Alignment

For this tutorial, we are using the same alignment as in **Tutorial 1**. Identical to **Tutorial 1**, the alignment is imported and annotated and a `Task` object is created.

[2]:
```
# read in the alignment
aln = hm.Alignment('data/single/aln_1.fasta_aln')

# annotate the alignment
aln.get_sequence('ARAF').annotate(
    seq_type = 'sequence')
aln.get_sequence('3NY5').annotate(
    seq_type = 'structure',
    pdb_code = '3NY5',
    begin_res = '1',
    begin_chain = 'A',
    end_res = '81',
    end_chain = 'A')

# initialize task object
t = hm.Task(
    task_name = 'Tutorial4',
    target = 'ARAF',
    alignment = aln,
    overwrite = True)
```

2.4.3 Defining custom routines

As an example for a custom routine, we will implement a `LoopModel` class from `modeller` [1,2] loosely following this tutorial on the `modeller` web page (in the section **Loop Refining**).

[3]:
```
class Routine_loopmodel(hm.routines.Routine): # (1)
    '''
    Custom routine for modeller loop modelling.
    '''
    def __init__(self, alignment, target, templates, tag, n_models=1, n_loop_models=1):
↪# (2)
        hm.routines.Routine.__init__(self, alignment, target, templates, tag)
        self.routine = 'loopmodel' # string identifier of routine

        self.n_models = n_models
        self.n_loop_models = n_loop_models

    def generate_models(self): # (3)
        # (4) process alignment
        self.alignment.select_sequences([self.target] + self.templates)
        self.alignment.remove_redundant_gaps()
        # write alignemnt to temporary file
        self.alignment.write_pir('.tmp.pir')

        # (5) define custom loop model class
        class MyLoop(LoopModel):
            # set residues that will be refined by loop modelling
            def select_loop_atoms(self):
                return Selection(self.residue_range('18:A', '22:A'))

        with contextlib.redirect_stdout(None): # (6) suppress modeller output to stdout
            # (7) set up modeller environment
            env = environ()
            env.io.hetatm = True

            # initialize model
            m = MyLoop(env,
                       alnfile='.tmp.pir',
                       knowns=self.templates,
                       sequence=self.target)

            # set modelling parameters
            m.blank_single_chain = False
            m.starting_model = 1
            m.ending_model = self.n_models
            m.loop.starting_model = 1
            m.loop.ending_model = self.n_loop_models

            # make models
            m.make()

        # (8) capture output
        for pdb in glob.glob('{}.BL*.pdb'.format(self.target)):
```

(continues on next page)

(continued from previous page)

```
            self.models.append(
                hm.Model(os.path.realpath(os.path.expanduser(pdb)),
                         self.tag, self.routine))

        # (9) rename files with method from hm.routines.Routine
        self._rename_models()

        # (10) clean up
        self._remove_files(
            '{}.B99*.pdb'.format(self.target),
            '{}.D00*'.format(self.target),
            '{}.DL*'.format(self.target),
            '{}.IL*'.format(self.target),
            '{}.ini'.format(self.target),
            '{}.lrsr'.format(self.target),
            '{}.rsr'.format(self.target),
            '{}.sch'.format(self.target),
            '.tmp*')
```

The lines of code in the definition of the custom routine above that are marked with numbers get special comments here:

1. Our custom routine in this example inherits from a parent class `Routine` defined in `homelette`. This is not strictly necessary, however, the parent class has a few useful functions already implemented that we will make use of (see steps 2, 9, 10)
2. Every routine needs to accept these arguments: `alignment`, `target`, `templates`, `tag`. In our case, we just hand them through to the parent method `Routine.__init__` that saves them as attributes, as well as introduces the attribute `self.models` where models will be deposited after generation.
3. Every routine needs a `generate_models` method. Usually, functionality for, you guessed it, model generation is packed in there.
4. `modeller` requires the aligment as a file in PIR format. The following few lines of code format the alignment and then produce the required file.
5. The following lines follow closely the `modeller` tutorial for loop modelling. This part implements a custom `LoopModel` class that defines a specific set of residue to be considered for loop modelling.
6. `modeller` writes a lot of output to stdout, and using `contextlib` is a way to suppress this output. If you want to see all the output from `modeller`, either delete the `with` statement or write `with contextlib.redirect_stdout(sys.stdout):` instead.
7. The following lines follow closely the `modeller` tutorial for loop modelling. This part initializes the model and generates the models requested.
8. The final models generated will be called `ARAF.BL00010001.pdb` and so on. These lines of code find these PDB files and add them to the `Routine_loopmodel.models` list as `Model`s. After execution by a `Task` objects, `Model` objects in this list will be added to the `Task.models` list.
9. Models generated will be renamed according to the tag given using the parent class method `Routine._rename_models`.
10. Temporary files from modeller as well as the temporary alignment file are removed from the folder using the parent class method `Routine._remove_files`.

Now, after implementing the routine, let's try it out in practice. As explained in **Tutorial 2**, we will be using the `Task.execute_routine` interface for that:

[4]:
```
# perform modelling
t.execute_routine(
    tag = 'custom_loop',
    routine = Routine_loopmodel,
    templates = ['3NY5'],
    template_location = './data/single',
    n_models = 2,
    n_loop_models = 2)
```

[5]:
```
# check generated models
t.models
```

[5]:
```
[<homelette.organization.Model at 0x7f211ff3fa30>,
 <homelette.organization.Model at 0x7f211ff54a30>,
 <homelette.organization.Model at 0x7f211ff54d90>,
 <homelette.organization.Model at 0x7f211ff54e20>]
```

In practice, a valid routine only needs to adhere to a small number of formal criteria to fit in the `homelette` framework:

- It needs to be an object.
- It needs to have an `__init__` method that can handle the named arguments `alignment`, `target`, `templates` and `tag`.
- It needs a `generate_models` method.
- It needs an attribute `models` in which generated models are stored as `Model` objects in list.

Any object that satisfies these criteria can be used in the framework.

2.4.4 Defining custom evaluations

As an example for a custom evaluation, we will implement a sample evaluation that counts the number of residues in the models.

[6]:
```
class Evaluation_countresidues():
    '''
    Custom evaluation: counting CA atoms
    '''
    def __init__(self, model, quiet=True): # (1)
        self.model = model
        self.output = dict()
        # (2) perform evaluation
        self.evaluate()
        # (3) update model.evaluation
        self.model.evaluation.update(self.output)

    def evaluate(self): # (4)
        # (5) parse model pdb
        pdb = self.model.parse_pdb()
```

(continues on next page)

(continued from previous page)

```
        # count number of CA atoms in PDB
        n_residues = pdb['name'].eq('CA').sum()

        # append to output
        self.output['n_residues'] = n_residues
```

The lines of code marked with numbers in the definiton of the custom evaluation get special comments here:

1. The `__init__` function takes exactly 2 arguments: `model` and `quiet`. `quiet` is a boolean value indicating whether output to stdout should be suppressed (not applicable in this case).
2. All evaluation metrics are executed upon initialization.
3. The `custom_evaluation.output` dictionary is merged with the `Model.evaluation` dictionary to make the output of our evaluation metrics available to the model.
4. Here we define the function where the actual evaluation takes place.
5. For the actual evaluation, we make use of the `Model.parse_pdb` method, which parses the PDB file associated to a specific model object to a `pandas` data frame. This can be useful for a number of evaluations (access residues, coordinates, etc.)

Note

If more arguments are required for a custom evaluation, we recomment to store them as attributes in the `Model` objects and then access these attributes while running the evaluation.

Now we apply our custom evaluation to our previously generated models using the `Task.evaluate_models` interface (for more details, see **Tutorial 3**):

[7]:
```
t.evaluate_models(Evaluation_countresidues)
t.get_evaluation()
```

[7]:

	model	tag	routine	n_residues
0	custom_loop_1.pdb	custom_loop	loopmodel	73
1	custom_loop_2.pdb	custom_loop	loopmodel	73
2	custom_loop_3.pdb	custom_loop	loopmodel	73
3	custom_loop_4.pdb	custom_loop	loopmodel	73

In practice, the formal requirements for a custom evaluation are the following:

- It has to be an object.
- `__init__` has the two arguments `model` and `quiet`. More arguments would work in conjunction with `Task.evaluate_models` only if defaults are set and used. We recommend storing more arguments as attributes in the `Model` object and then accessing them during the evaluation.
- It executes evaluation on initialization.
- On finishing the evaluation, it updates the `Model.evaluation` dictionary with the results of the evaluation.

2.4.5 Further reading

Congratulations on finishing the tutorial on extending `homelette`.

Please take again notice that on our online documentation, there is a page collecting user-submitted custom routines and evaluation metrics. User are encouraged to share if they implemented something which they might think could be useful for the community. If you are interested, please contact us on GitHub or via email.

There are more tutorials which might interest you:

- **Tutorial 1**: Learn about the basics of `homelette`.
- **Tutorial 2**: Learn more about already implemented routines for homology modelling.
- **Tutorial 3**: Learn about the evaluation metrics available with `homelette`.
- **Tutorial 5**: Learn about how to use parallelization in order to generate and evaluate models more efficiently.
- **Tutorial 6**: Learn about modelling protein complexes.
- **Tutorial 7**: Learn about assembling custom pipelines.
- **Tutorial 8**: Learn about automated template identification, alignment generation and template processing.

2.4.6 References

[1] Šali, A., & Blundell, T. L. (1993). Comparative protein modelling by satisfaction of spatial restraints. Journal of Molecular Biology, 234(3), 779–815. https://doi.org/10.1006/jmbi.1993.1626

[2] Webb, B., & Sali, A. (2016). Comparative Protein Structure Modeling Using MODELLER. Current Protocols in Bioinformatics, 54(1), 5.6.1-5.6.37. https://doi.org/10.1002/cpbi.3

2.4.7 Session Info

[8]:
```
# session info
import session_info
session_info.show(html = False, dependencies = True)
```

```
-----
homelette           1.4
modeller            10.4
pandas              1.5.3
session_info        1.0.0
-----
PIL                 7.0.0
altmod              NA
anyio               NA
asttokens           NA
attr                19.3.0
babel               2.12.1
backcall            0.2.0
certifi             2022.12.07
chardet             3.0.4
charset_normalizer  3.1.0
comm                0.1.2
cycler              0.10.0
```

(continues on next page)

(continued from previous page)

```
cython_runtime      NA
dateutil            2.8.2
debugpy             1.6.6
decorator           4.4.2
executing           1.2.0
fastjsonschema      NA
idna                3.4
importlib_metadata  NA
importlib_resources NA
ipykernel           6.21.3
ipython_genutils    0.2.0
jedi                0.18.2
jinja2              3.1.2
json5               NA
jsonschema          4.17.3
jupyter_events      0.6.3
jupyter_server      2.4.0
jupyterlab_server   2.20.0
kiwisolver          1.0.1
markupsafe          2.1.2
matplotlib          3.1.2
more_itertools      NA
mpl_toolkits        NA
nbformat            5.7.3
numexpr             2.8.4
numpy               1.24.2
ost                 2.3.1
packaging           20.3
parso               0.8.3
pexpect             4.8.0
pickleshare         0.7.5
pkg_resources       NA
platformdirs        3.1.1
prometheus_client   NA
promod3             3.2.1
prompt_toolkit      3.0.38
psutil              5.5.1
ptyprocess          0.7.0
pure_eval           0.2.2
pydev_ipython       NA
pydevconsole        NA
pydevd              2.9.5
pydevd_file_utils   NA
pydevd_plugins      NA
pydevd_tracing      NA
pygments            2.14.0
pyparsing           2.4.6
pyrsistent          NA
pythonjsonlogger    NA
pytz                2022.7.1
qmean               NA
requests            2.28.2
```

(continues on next page)

(continued from previous page)

```
rfc3339_validator   0.1.4
rfc3986_validator   0.1.1
send2trash          NA
sitecustomize       NA
six                 1.12.0
sniffio             1.3.0
stack_data          0.6.2
swig_runtime_data4  NA
tornado             6.2
traitlets           5.9.0
urllib3             1.26.15
wcwidth             NA
websocket           1.5.1
yaml                6.0
zipp                NA
zmq                 25.0.1
-----
IPython             8.11.0
jupyter_client      8.0.3
jupyter_core        5.2.0
jupyterlab          3.6.1
notebook            6.5.3
-----
Python 3.8.10 (default, Nov 14 2022, 12:59:47) [GCC 9.4.0]
Linux-4.15.0-206-generic-x86_64-with-glibc2.29
-----
Session information updated at 2023-03-15 23:36
```

2.5 Tutorial 5: Parallelization

[1]:
```
import homelette as hm

import time
```

2.5.1 Introduction

Welcome to the fifth tutorial on `homelette`. This tutorial is about parallelization in `homelette`. When modelling hundreds or thousands of models, some processes can be significantly sped up by dividing the workload on multiple processes in parallel (supported by appropriate hardware).

There are possibilities to parallelize both model generation and evaluation in `homelette`.

2.5.2 Alignment and Task setup

For this tutorial, we are using the same alignment as in **Tutorial 1**. Identical to previous tutorials, the alignment is imported and annotated, and a `Task` object is set up.

[2]:
```
# read in the alignment
aln = hm.Alignment('data/single/aln_1.fasta_aln')

# annotate the alignment
aln.get_sequence('ARAF').annotate(
    seq_type = 'sequence')
aln.get_sequence('3NY5').annotate(
    seq_type = 'structure',
    pdb_code = '3NY5',
    begin_res = '1',
    begin_chain = 'A',
    end_res = '81',
    end_chain = 'A')

# initialize task object
t = hm.Task(
    task_name = 'Tutorial5',
    target = 'ARAF',
    alignment = aln,
    overwrite = True)
```

2.5.3 Parallel model generation

When trying to parallelize model generation, `homelette` makes use of the parallelization methods implemented in the packages that `homelette` uses, if they are available. Model generation with `modeller` can be parallized and is available in `homelette` through a simple handler [1,2].

All `modeller` based, pre-implemented routines have the argument `n_threads` which can be used to use parallelization. The default is `n_threads = 1` which does not activate parallelization, but any number > 1 will distribute the workload on the number of threads requested using the `modeller.parallel` submodule.

[3]:
```
# use only 1 thread to generate 20 models
start = time.perf_counter()
t.execute_routine(
    tag = '1_thread',
    routine = hm.routines.Routine_automodel_default,
    templates = ['3NY5'],
    template_location = './data/single/',
    n_models = 20)
print(f'Elapsed time: {time.perf_counter() - start:.2f}')
```

```
Elapsed time: 47.84
```

[4]:
```
# use 4 threads to generate 20 models faster
start  = time.perf_counter()
t.execute_routine(
    tag = '4_threads',
    routine = hm.routines.Routine_automodel_default,
```

(continues on next page)

(continued from previous page)

```
    templates = ['3NY5'],
    template_location = './data/single/',
    n_models = 20,
    n_threads = 4)
print(f'Elapsed time: {time.perf_counter() - start:.2f}')
```

```
Elapsed time: 15.44
```

Using multiple threads can significantly speed up model generation, especially if a large number of models is generated.

Note

Please be aware that the `modeller.parallel` submodule uses the Python module `pickle`, which requires objects to be pickled to be saved in a separate file. In practical terms, if you want to run parallelization in modeller with a custom object (i.e. a custom defined routine, see **Tutorial 4**), you cannot make use of parallelization unless you have imported it from a separate file. Therefore we recommend that custom routines and evaluation are saved in a separate file and then imported from there.

The following code block shows how custom building blocks could be put in an external file (`data/extension.py`) and then imported for modelling and analysis.

[5]:
```
# import from custom file
from data.extension import Custom_Routine, Custom_Evaluation

?Custom_Routine
```

```
Init signature: Custom_Routine()
Docstring:      Custom routine waiting to be implemented.
File:           ~/workdir/data/extension.py
Type:           type
Subclasses:
```

[6]:
```
!cat data/extension.py
```

```
'''
Examples of custom objects for homelette in a external file.
'''


class Custom_Routine():
    '''
    Custom routine waiting to be implemented.
    '''
    def __init__(self):
        print('TODO: implement this')


class Custom_Evaluation():
    '''
    Custom evaluation waiting to be implemented.
    '''
    def __init__(self):
        print('TODO: implement this')
```

Alternatively, you could use the `/homelette/extension/` folder in which extensions are stored. See our comments on extensions in our documentation for more details.

2.5.4 Parallel model evaluation

`homelette` can also use parallelization to speed up model evaluation. This is internally archieved by using `concurrent.futures.ThreadPoolExecutor`.

In order to use parallelization when performing evaluations, use the `n_threads` argument in `Task.evaluate_models`.

[7]:
```
# use 1 thread for model evaluation
start  = time.perf_counter()
t.evaluate_models(hm.evaluation.Evaluation_mol_probity, n_threads=1)
print(f'Elapsed time: {time.perf_counter() - start:.2f}')
```

```
Elapsed time: 468.37
```

[8]:
```
# use 4 threads for model evaluation
start  = time.perf_counter()
t.evaluate_models(hm.evaluation.Evaluation_mol_probity, n_threads=4)
print(f'Elapsed time: {time.perf_counter() - start:.2f}')
```

```
Elapsed time: 128.37
```

For some evaluation schemes, using parallelization can lead to a significant speedup.

Note

Please be advised that for some (very fast) evaluation methods, the time investment of spawning new child processes might not compensate for the speedup gained by parallelization. Test your usecase on your system in a small setting and use at your own discretion.

---

[9]:
```
# use 1 thread for model evaluation
start  = time.perf_counter()
t.evaluate_models(hm.evaluation.Evaluation_dope, n_threads=1)
print(f'Elapsed time: {time.perf_counter() - start:.2f}')
```

```
Elapsed time: 10.34
```

[10]:
```
# use 4 threads for model evaluation
start  = time.perf_counter()
t.evaluate_models(hm.evaluation.Evaluation_dope, n_threads=4)
print(f'Elapsed time: {time.perf_counter() - start:.2f}')
```

```
Elapsed time: 15.95
```

Note

When creating and using custom evaluation metrics, please make sure to avoid race conditions. `Task.evaluate_models` is implemented with a protection against race conditions, but this is not bulletproof. Also, if you need to create temporary files, make sure to create file names with model-specific names (i.e. by using the model name in the file name). Defining custom evaluations in a separate file is not necessary, as parallelization of evaluation methods does not rely on `pickle`.

Note

In case some custom evaluation metrics are very memory-demanding, running it in parallel can easily overwhelm the system. Again, we encourage you to test your usecase on your system in a small setting.

2.5.5 Further reading

Congratulation on completing **Tutorial 5** about parallelization in `homelette`. Please note that there are other tutorials, which will teach you more about how to use `homelette`:

- **Tutorial 1**: Learn about the basics of `homelette`.
- **Tutorial 2**: Learn more about already implemented routines for homology modelling.
- **Tutorial 3**: Learn about the evaluation metrics available with `homelette`.
- **Tutorial 4**: Learn about extending `homelette`'s functionality by defining your own modelling routines and evaluation metrics.
- **Tutorial 6**: Learn about modelling protein complexes.
- **Tutorial 7**: Learn about assembling custom pipelines.
- **Tutorial 8**: Learn about automated template identification, alignment generation and template processing.

2.5.6 References

[1] Šali, A., & Blundell, T. L. (1993). Comparative protein modelling by satisfaction of spatial restraints. Journal of Molecular Biology, 234(3), 779–815. https://doi.org/10.1006/jmbi.1993.1626

[2] Webb, B., & Sali, A. (2016). Comparative Protein Structure Modeling Using MODELLER. Current Protocols in Bioinformatics, 54(1), 5.6.1-5.6.37. https://doi.org/10.1002/cpbi.3

2.5.7 Session Info

[11]:
```
# session info
import session_info
session_info.show(html = False, dependencies = True)
```

```
-----
data                NA
homelette           1.4
session_info        1.0.0
-----
PIL                 7.0.0
altmod              NA
anyio               NA
asttokens           NA
attr                19.3.0
babel               2.12.1
backcall            0.2.0
certifi             2022.12.07
chardet             3.0.4
```

(continues on next page)

(continued from previous page)

```
charset_normalizer  3.1.0
comm                0.1.2
cycler              0.10.0
cython_runtime      NA
dateutil            2.8.2
debugpy             1.6.6
decorator           4.4.2
executing           1.2.0
fastjsonschema      NA
idna                3.4
importlib_metadata  NA
importlib_resources NA
ipykernel           6.21.3
ipython_genutils    0.2.0
jedi                0.18.2
jinja2              3.1.2
json5               NA
jsonschema          4.17.3
jupyter_events      0.6.3
jupyter_server      2.4.0
jupyterlab_server   2.20.0
kiwisolver          1.0.1
markupsafe          2.1.2
matplotlib          3.1.2
modeller            10.4
more_itertools      NA
mpl_toolkits        NA
nbformat            5.7.3
numexpr             2.8.4
numpy               1.24.2
ost                 2.3.1
packaging           20.3
pandas              1.5.3
parso               0.8.3
pexpect             4.8.0
pickleshare         0.7.5
pkg_resources       NA
platformdirs        3.1.1
prometheus_client   NA
promod3             3.2.1
prompt_toolkit      3.0.38
psutil              5.5.1
ptyprocess          0.7.0
pure_eval           0.2.2
pydev_ipython       NA
pydevconsole        NA
pydevd              2.9.5
pydevd_file_utils   NA
pydevd_plugins      NA
pydevd_tracing      NA
pygments            2.14.0
pyparsing           2.4.6
```

(continues on next page)

(continued from previous page)

```
pyrsistent          NA
pythonjsonlogger    NA
pytz                2022.7.1
qmean               NA
requests            2.28.2
rfc3339_validator   0.1.4
rfc3986_validator   0.1.1
send2trash          NA
sitecustomize       NA
six                 1.12.0
sniffio             1.3.0
stack_data          0.6.2
swig_runtime_data4  NA
tornado             6.2
traitlets           5.9.0
urllib3             1.26.15
wcwidth             NA
websocket           1.5.1
yaml                6.0
zipp                NA
zmq                 25.0.1
-----
IPython             8.11.0
jupyter_client      8.0.3
jupyter_core        5.2.0
jupyterlab          3.6.1
notebook            6.5.3
-----
Python 3.8.10 (default, Nov 14 2022, 12:59:47) [GCC 9.4.0]
Linux-4.15.0-206-generic-x86_64-with-glibc2.29
-----
Session information updated at 2023-03-15 23:56
```

2.6 Tutorial 6: Complex Modelling

[1]:
```
import homelette as hm
```

2.6.1 Introduction

Welcome to the 6th tutorial on `homelette` about homology modelling of complex structures.

There are multiple issues about modelling protein complexes that make it a separate topic from the homology modelling of single structures:

- Usually, a complex structure is required as a template.
- Not all modelling programs can perform complex modelling.
- Not all evaluation metrics developed for homology modelling are applicable to complex structures.
- You need multiple alignments.

homelette is able to use modeller based modelling routines for complex modelling [1,2], and has some specific classes in place that make complex modelling easier to the user: - A function to assemble appropriate complex alignments - Special modelling classes for complex modelling - Special evaluation metrics for complex modelling

For this tutorial, we will build models for ARAF in complex with HRAS. As a template, we will use the structures [4G0N] (https://www.rcsb.org/structure/4G0N)(RAF1 in complex with HRAS) and 3NY5 (BRAF).

2.6.2 Alignment

Since all current modelling routines for protein complexes are modeller based, an alignment according to the modeller specification has to be constructed. homelette has the helper function assemble_complex_aln in the homelette.alignment submodule that is able to do that:

[2]:
```
?hm.alignment.assemble_complex_aln
```

```
Signature:
hm.alignment.assemble_complex_aln(
    *args: Type[ForwardRef('Alignment')],
    names: dict,
) -> Type[ForwardRef('Alignment')]
Docstring:
Assemble complex alignments compatible with MODELLER from individual
alignments.

Parameters
----------
*args : Alignment
    The input alignments
names : dict
    Dictionary instructing how sequences in the different alignment objects
    are supposed to be arranged in the complex alignment. The keys are the
    names of the sequences in the output alignments. The values are
    iterables of the sequence names from the input alignments in the order
    they are supposed to appaer in the output alignment. Any value that can
    not be found in the alignment signals that this position in the complex
    alignment should be filled with gaps.

Returns
-------
Alignment
    Assembled complex alignment

Examples
--------
>>> aln1 = hm.Alignment(None)
>>> aln1.sequences = {
...     'seq1_1': hm.alignment.Sequence('seq1_1', 'HELLO'),
...     'seq2_1': hm.alignment.Sequence('seq2_1', 'H---I'),
...     'seq3_1': hm.alignment.Sequence('seq3_1', '-HI--')
...     }
>>> aln2 = hm.Alignment(None)
>>> aln2.sequences = {
...     'seq2_2': hm.alignment.Sequence('seq2_2', 'KITTY'),
...     'seq1_2': hm.alignment.Sequence('seq1_2', 'WORLD')
```

(continues on next page)

(continued from previous page)

```
...     }
>>> names = {'seq1': ('seq1_1', 'seq1_2'),
...          'seq2': ('seq2_1', 'seq2_2'),
...          'seq3': ('seq3_1', 'gaps')
...     }
>>> aln_assembled = hm.alignment.assemble_complex_aln(
...     aln1, aln2, names=names)
>>> aln_assembled.print_clustal()
seq1        HELLO/WORLD
seq2        H---I/KITTY
seq3        -HI--/-----
File:      /usr/local/src/homelette-1.4/homelette/alignment.py
Type:      function
```

In our case, we assemble an alignment from two different alignments, `aln_1` which contains ARAF, RAF1 (*4G0N*) and BRAF (*3NY5*) and `aln_2` which contains an HRAS sequence and the HRAS sequence from *4G0N*.

[3]:
```
# import single alignments
aln1_file = 'data/complex/aln_eff.fasta_aln'
aln2_file = 'data/complex/aln_ras.fasta_aln'

aln_1 = hm.Alignment(aln1_file)
aln_2 = hm.Alignment(aln2_file)

# build dictionary that indicates how sequences should be assembled
names = {
    'ARAF': ('ARAF', 'HRAS'),
    '4G0N': ('4G0N', '4G0N'),
    '3NY5': ('3NY5', ''),
}

# assemble alignment
aln = hm.alignment.assemble_complex_aln(aln_1, aln_2, names=names)
aln.remove_redundant_gaps()
aln.print_clustal(line_wrap=70)
```

```
ARAF        ---GTVKVYLPNKQRTVVTVRDGMSVYDSLDKALKVRGLNQDCCVVYRLI---KGRKTVTAWDTAIAPLD
4G0N        -TSNTIRVFLPNKQRTVVNVRNGMSLHDCLMKALKVRGLQPECCAVFRLLHEHKGKKARLDWNTDAASLI
3NY5        HQKPIVRVFLPNKQRTVVPARCGVTVRDSLKKAL--RGLIPECCAVYRIQ------KKPIGWDTDISWLT


ARAF        GEELIVEVL------/MTEYKLVVVGAGGVGKSALTIQLIQNHFVDEYDPTIEDSYRKQVVIDGETCLLD
4G0N        GEELQVDFL------/MTEYKLVVVGAGGVGKSALTIQLIQNHFVDEYDPTIEDSYRKQVVIDGETCLLD
3NY5        GEELHVEVLENVPLT/------------------------------------------------------


ARAF        ILDTAGQEEYSAMRDQYMRTGEGFLCVFAINNTKSFEDIHQYREQIKRVKDSDDVPMVLVGNKCDLAART
4G0N        ILDTAGQEE--AMRDQYMRTGEGFLCVFAINNTKSFEDIHQYREQIKRVKDSDDVPMVLVGNKCDLAART
3NY5        ------------------------------------------------------------------------


ARAF        VESRQAQDLARSYGIPYIETSAKTRQGVEDAFYTLVREIRQ-
4G0N        VESRQAQDLARSYGIPYIETSAKTRQGVEDAFYTLVREIRQH
```

(continues on next page)

(continued from previous page)

```
3NY5        ----------------------------------------
```

After assembling the complex alignment, we annotate it as usual:

[4]:
```
# annotate alignment
aln.get_sequence('ARAF').annotate(seq_type='sequence')
aln.get_sequence('4G0N').annotate(seq_type = 'structure',
                                  pdb_code = '4G0N',
                                  begin_res = '1',
                                  begin_chain = 'A')
aln.get_sequence('3NY5').annotate(seq_type = 'structure',
                                  pdb_code = '3NY5',
                                  begin_res = '1',
                                  begin_chain = 'A')
```

2.6.3 Modelling

There are 4 routines available specifically for complex modelling based on `modeller` [1,2] and `altmod` [3]. They run with the same parameters as their counterparts for single structure modelling, except that they handle naming of new chains and residue numbers a bit differently.

The following routines are available for complex modelling:

- `Routine_complex_automodel_default`
- `Routine_complex_automodel_slow`
- `Routine_complex_altmod_default`
- `Routine_complex_altmod_slow`

[5]:
```
# initialize task object
t = hm.Task(task_name='Tutorial6',
            alignment=aln,
            target='ARAF',
            overwrite=True)
```

Modelling can be performed with `Task.execute_routine` as usual.

[6]:
```
# generate models based on a complex template
t.execute_routine(tag='automodel_' + '4G0N',
                  routine=hm.routines.Routine_complex_automodel_default,
                  templates = ['4G0N'],
                  template_location='data/complex/',
                  n_models=20,
                  n_threads=5)
```

Not all templates have to be complex templates, it is perfectly applicable to mix complex templates and single templates. However, at least one complex template should be used in order to convey information about the orientation of the proteins to each other.

[7]:
```
# generate models based on a complex and a single template
t.execute_routine(tag='automodel_' + '_'.join(['4G0N', '3NY5']),
                  routine=hm.routines.Routine_complex_automodel_default,
                  templates = ['4G0N', '3NY5'],
                  template_location='data/complex',
                  n_models=20,
                  n_threads=5)
```

2.6.4 Evaluation

Not all evaluation metrics are designed to evaluate complex structures. For example, the SOAP score has different statistical potentials for single proteins (`Evaluation_soap_protein`) and for protein complexes (`Evaluation_soap_pp`) [4].

[8]:
```
# perform evaluation
t.evaluate_models(hm.evaluation.Evaluation_mol_probity,
                  hm.evaluation.Evaluation_soap_pp,
                  n_threads=5)
```

[9]:
```
# show a bit of the evaluation
t.get_evaluation().sort_values(by='soap_pp_all').head()
```

[9]:

	model	tag	routine	mp_score	soap_pp_all	soap_pp_atom	soap_pp_pair
32	automodel_4G0N_3NY5_13.pdb	automodel_4G0N_3NY5	complex_automodel_default	2.25	-9502.636719	-7770.577637	-1732.059326
39	automodel_4G0N_3NY5_20.pdb	automodel_4G0N_3NY5	complex_automodel_default	2.15	-9486.243164	-7656.946777	-1829.296143
28	automodel_4G0N_3NY5_9.pdb	automodel_4G0N_3NY5	complex_automodel_default	2.46	-9475.368164	-7769.337891	-1706.030396
29	automodel_4G0N_3NY5_10.pdb	automodel_4G0N_3NY5	complex_automodel_default	2.72	-9458.609375	-7647.797852	-1810.811646
9	automodel_4G0N_10.pdb	automodel_4G0N	complex_automodel_default	2.39	-9405.662109	-7718.845215	-1686.817139

2.6.5 Further reading

Congratulation on finishing the tutorial about complex modelling in `homelette`. The following tutorials might also be of interest to you:

- **Tutorial 1**: Learn about the basics of `homelette`.
- **Tutorial 2**: Learn more about already implemented routines for homology modelling.
- **Tutorial 3**: Learn about the evaluation metrics available with `homelette`.
- **Tutorial 4**: Learn about extending `homelette`'s functionality by defining your own modelling routines and evaluation metrics.
- **Tutorial 5**: Learn about how to use parallelization in order to generate and evaluate models more efficiently.
- **Tutorial 7**: Learn about assembling custom pipelines.
- **Tutorial 8**: Learn about automated template identification, alignment generation and template processing.

2.6.6 References

[1] Šali, A., & Blundell, T. L. (1993). Comparative protein modelling by satisfaction of spatial restraints. Journal of Molecular Biology, 234(3), 779–815. https://doi.org/10.1006/jmbi.1993.1626

[2] Webb, B., & Sali, A. (2016). Comparative Protein Structure Modeling Using MODELLER. Current Protocols in Bioinformatics, 54(1), 5.6.1-5.6.37. https://doi.org/10.1002/cpbi.3

[3] Janson, G., Grottesi, A., Pietrosanto, M., Ausiello, G., Guarguaglini, G., & Paiardini, A. (2019). Revisiting the "satisfaction of spatial restraints" approach of MODELLER for protein homology modeling. PLoS Computational Biology, 15(12), e1007219. https://doi.org/10.1371/journal.pcbi.1007219

[4] Dong, G. Q., Fan, H., Schneidman-Duhovny, D., Webb, B., Sali, A., & Tramontano, A. (2013). Optimized atomic statistical potentials: Assessment of protein interfaces and loops. Bioinformatics, 29(24), 3158–3166. https://doi.org/10.1093/bioinformatics/btt560

2.6.7 Session Info

[10]:
```
# session info
import session_info
session_info.show(html = False, dependencies = True)
```

```
-----
homelette           1.4
pandas              1.5.3
session_info        1.0.0
-----
PIL                 7.0.0
altmod              NA
anyio               NA
asttokens           NA
attr                19.3.0
babel               2.12.1
backcall            0.2.0
certifi             2022.12.07
chardet             3.0.4
```

(continues on next page)

(continued from previous page)

```
charset_normalizer  3.1.0
comm                0.1.2
cycler              0.10.0
cython_runtime      NA
dateutil            2.8.2
debugpy             1.6.6
decorator           4.4.2
executing           1.2.0
fastjsonschema      NA
idna                3.4
importlib_metadata  NA
importlib_resources NA
ipykernel           6.21.3
ipython_genutils    0.2.0
jedi                0.18.2
jinja2              3.1.2
json5               NA
jsonschema          4.17.3
jupyter_events      0.6.3
jupyter_server      2.4.0
jupyterlab_server   2.20.0
kiwisolver          1.0.1
markupsafe          2.1.2
matplotlib          3.1.2
modeller            10.4
more_itertools      NA
mpl_toolkits        NA
nbformat            5.7.3
numexpr             2.8.4
numpy               1.24.2
ost                 2.3.1
packaging           20.3
parso               0.8.3
pexpect             4.8.0
pickleshare         0.7.5
pkg_resources       NA
platformdirs        3.1.1
prometheus_client   NA
promod3             3.2.1
prompt_toolkit      3.0.38
psutil              5.5.1
ptyprocess          0.7.0
pure_eval           0.2.2
pydev_ipython       NA
pydevconsole        NA
pydevd              2.9.5
pydevd_file_utils   NA
pydevd_plugins      NA
pydevd_tracing      NA
pygments            2.14.0
pyparsing           2.4.6
pyrsistent          NA
```

(continues on next page)

(continued from previous page)

```
pythonjsonlogger    NA
pytz                2022.7.1
qmean               NA
requests            2.28.2
rfc3339_validator   0.1.4
rfc3986_validator   0.1.1
send2trash          NA
sitecustomize       NA
six                 1.12.0
sniffio             1.3.0
stack_data          0.6.2
swig_runtime_data4  NA
tornado             6.2
traitlets           5.9.0
urllib3             1.26.15
wcwidth             NA
websocket           1.5.1
yaml                6.0
zipp                NA
zmq                 25.0.1
-----
IPython             8.11.0
jupyter_client      8.0.3
jupyter_core        5.2.0
jupyterlab          3.6.1
notebook            6.5.3
-----
Python 3.8.10 (default, Nov 14 2022, 12:59:47) [GCC 9.4.0]
Linux-4.15.0-206-generic-x86_64-with-glibc2.29
-----
Session information updated at 2023-03-15 23:40
```

2.7 Tutorial 7: Assembling custom pipelines

[1]:
```
import homelette as hm

import matplotlib as plt
import seaborn as sns
```

2.7.1 Introduction

Welcome to the final tutorial on `homelette`. This tutorial is about combining what we learnt in the previous tutorials about model generating and model evaluating building blocks.

The strength of `homelette` lies in its ability to *A)* be almost freely extendable by the user (see **Tutorial 4**) and *B)* in the ease with which pre-defined or custom-made building blocks for model generation and evaluation can be assembled into custom pipelines. This tutorial showcases *B)*.

For our target sequence, ARAF, we will identify templates and generate alignments with the `AlignmentGenerator_pdb` building block [1,2,3,4]. We will select two templates, BRAF (3NY5) and RAF1

(4G0N). We will build models for ARAF with two different routines, `Routine_automodel_default` and `Routine_automodel_slow` [5,6], and from the different templates. The generated models will be evaluated by SOAP scores and MolProbity and a combined score will be calculated using Borda Count [7,8,9,10].

2.7.2 Alignment

Consistent with the other tutorials, we will be modelling the protein ARAF. For this tutorial, we will use the `AlignmentGenerator_pdb` in order to search for templates, create an alignment, and process both the templates as well as the alignment:

[2]:
```
gen = hm.alignment.AlignmentGenerator_pdb.from_fasta('data/alignments/ARAF.fa')
```

[3]:
```
# search for templates and generate first alignment
gen.get_suggestion()
gen.show_suggestion()
```

```
Querying PDB...
Query successful, 16 found!

Retrieving sequences...
Sequences succefully retrieved!

Generating alignment...
Alignment generated!
```

[3]:

	template	coverage	identity
0	6XI7_2	100.0	60.27
1	1C1Y_2	100.0	60.27
2	1GUA_2	100.0	60.27
3	4G0N_2	100.0	60.27
4	4G3X_2	100.0	60.27
5	6VJJ_2	100.0	60.27
6	6XGU_2	100.0	60.27
7	6XGV_2	100.0	60.27
8	6XHA_2	100.0	60.27
9	6XHB_2	100.0	60.27
10	7JHP_2	100.0	60.27
11	3KUC_2	100.0	58.90
12	3KUD_2	100.0	58.90
13	3NY5_1	100.0	58.90
14	6NTD_2	100.0	53.42
15	6NTC_2	100.0	52.05

For this example, we will choose one template of BRAF (3NY5) and one template from RAF1 (4G0N):

[4]:
```
# select templates and show alignment
gen.select_templates(['3NY5_1', '4G0N_2'])
gen.alignment.print_clustal(70)
```

```
ARAF        -------------GTVKVYLPNKQRTVVTVRDGMSVYDSLDKALKVRGLNQDCCVVYRLI---KGRKTVT
3NY5_1      MGHHHHHHSHMQKPIVRVFLPNKQRTVVPARCGVTVRDSLKKALMMRGLIPECCAVYRIQ---DGEKKPI
4G0N_2      -----------TSNTIRVFLPNKQRTVVNVRNGMSLHDCLMKALKVRGLQPECCAVFRLLHEHKGKKARL
```

(continues on next page)

(continued from previous page)

```
ARAF        AWDTAIAPLDGEELIVEVL----------
3NY5_1      GWDTDISWLTGEELHVEVLENVPLTTHNF
4G0N_2      DWNTDAASLIGEELQVDFL----------
```

Next, we download the template structures and process both the alignment and the structures:

[5]:
```
# download structures, process alignment and structures
gen.get_pdbs()
gen.show_suggestion()
```

```
Guessing template naming format...
Template naming format guessed: polymer_entity!

Checking template dir...
Template dir not found...
New template dir created at
"/home/homelette/workdir/templates"!

Processing templates:

3NY5 downloading from PDB...
3NY5 downloaded!
3NY5_A: Chain extracted!
3NY5_A: Alignment updated!
3NY5_A: PDB processed!
3NY5_B: Chain extracted!
3NY5_B: Alignment updated!
3NY5_B: PDB processed!
3NY5_C: Chain extracted!
3NY5_C: Alignment updated!
3NY5_C: PDB processed!
3NY5_D: Chain extracted!
3NY5_D: Alignment updated!
3NY5_D: PDB processed!
4G0N downloading from PDB...
4G0N downloaded!
4G0N_B: Chain extracted!
4G0N_B: Alignment updated!
4G0N_B: PDB processed!

Finishing... All templates successfully
downloaded and processed!
Templates can be found in
"/home/homelette/workdir/templates".
```

[5]:

	template	coverage	identity
0	4G0N_B	100.00	60.27
1	3NY5_B	94.52	57.53
2	3NY5_A	93.15	57.53
3	3NY5_C	93.15	57.53

(continues on next page)

(continued from previous page)

```
4   3NY5_D     91.78     57.53
```

We can see that there are multiple chains of 3NY5 that fit our alignment. One of the chains has less missing residues than the other ones, so we are choosing this one:

[6]:
```
# select templates
gen.select_templates(['4G0N_B', '3NY5_B'])
gen.alignment.print_clustal(70)
gen.show_suggestion()
```

```
ARAF        ----GTVKVYLPNKQRTVVTVRDGMSVYDSLDKALKVRGLNQDCCVVYRLI---KGRKTVTAWDTAIAPL
4G0N_B      --TSNTIRVFLPNKQRTVVNVRNGMSLHDCLMKALKVRGLQPECCAVFRLLHEHKGKKARLDWNTDAASL
3NY5_B      SHQKPIVRVFLPNKQRTVVPARCGVTVRDSLKKAL--RGLIPECCAVYRIQ-----EKKPIGWDTDISWL


ARAF        DGEELIVEVL--------
4G0N_B      IGEELQVDFL--------
3NY5_B      TGEELHVEVLENVPLTTH
```

[6]:
```
  template  coverage  identity
0   4G0N_B    100.00     60.27
1   3NY5_B     94.52     57.53
```

Now that we have our templates prepared and aligned, we can now define a custom `Task` object in order to assemble `homelette` building blocks into a pipeline:

2.7.3 Custom pipeline

The easiest way to formulate custom pipelines by assembling the `homelette` building blocks of model building and evaluation is to construct custom `Task` objects:

[7]:
```
class CustomPipeline(hm.Task):
    '''
    Example for a cumstom pipeline
    '''
    def model_generation(self, templates):
        # model generation with automodel_default
        self.execute_routine(tag='automodel_def_' + '-'.join(templates),
                             routine = hm.routines.Routine_automodel_default,
                             templates = templates,
                             template_location = './templates/',
                             n_models = 20,
                             n_threads = 5)
        # model generation with autmodel_slow
        self.execute_routine(tag='autmodel_slow_' + '-'.join(templates),
                             routine = hm.routines.Routine_automodel_slow,
                             templates = templates,
                             template_location = './templates/',
                             n_models = 20,
                             n_threads = 5)
```

(continues on next page)

(continued from previous page)

```
    def model_evaluation(self):
        # perform evaluation
        self.evaluate_models(hm.evaluation.Evaluation_mol_probity,
                             n_threads=5)
        self.evaluate_models(hm.evaluation.Evaluation_soap_protein,
                             n_threads=5)
        self.evaluate_models(hm.evaluation.Evaluation_qmean4,
                            n_threads=5)
        ev = self.get_evaluation()
        # borda count for best models
        ev['points_soap'] = ev.shape[0] - ev['soap_protein'].rank()
        ev['points_mol_probity'] = ev.shape[0] - ev['mp_score'].rank()
        ev['borda_score'] = ev['points_soap'] + ev['points_mol_probity']
        ev['borda_rank'] = ev['borda_score'].rank(ascending=False)
        ev = ev.drop(labels=['points_soap', 'points_mol_probity'], axis=1)
        return ev
```

We have constructed a custom `Task` object (more specifically, a custom objects that inherits all methods and attributes from `Task`) and added two more functions: `model_generation` and `model_evaluation`.

In `CustomPipeline.model_generation` we are using two routines (`Routine_automodel_default` and `Routine_automodel_slow`) to generate 20 models each. In `CustomPipeline.model_generation` we evaluate the models using `Evaluation_mol_probity` and `Evaluation_soap_protein` and then rank the generated models based on both evaluation metrics using Borda Count.

After constructing our pipeline, let's execute it with two different templates as an example:

After having a custom `Task` object defined, we can initialize it from the `AlignmentGenerator` in order to do the modelling and evaluation:

[8]:
```
# initialize task from alignment generator
t = gen.initialize_task(
    task_name = 'Tutorial7',
    overwrite = True,
    task_class = CustomPipeline)
```

[9]:
```
# execute pipeline for different templates
t.model_generation(['3NY5_B'])
t.model_generation(['4G0N_B'])

df_eval = t.model_evaluation()
```

We have successfully generated and evaluated 80 models.

[10]:
```
# get template from tag
df_eval['template'] = df_eval['tag'].str.contains('3NY5').map({True: '3NY5', False: '4G0N
↪'})
```

[11]:
```
df_eval.sort_values(by = 'borda_rank').head(10)
```

[11]:
```
                         model                   tag            routine  \
64   autmodel_slow_4G0N_B_5.pdb  autmodel_slow_4G0N_B     automodel_slow
77  autmodel_slow_4G0N_B_18.pdb  autmodel_slow_4G0N_B     automodel_slow
```

(continues on next page)

(continued from previous page)

```
38  autmodel_slow_3NY5_B_19.pdb  autmodel_slow_3NY5_B     automodel_slow
69  autmodel_slow_4G0N_B_10.pdb  autmodel_slow_4G0N_B     automodel_slow
63   autmodel_slow_4G0N_B_4.pdb  autmodel_slow_4G0N_B     automodel_slow
79  autmodel_slow_4G0N_B_20.pdb  autmodel_slow_4G0N_B     automodel_slow
72  autmodel_slow_4G0N_B_13.pdb  autmodel_slow_4G0N_B     automodel_slow
73  autmodel_slow_4G0N_B_14.pdb  autmodel_slow_4G0N_B     automodel_slow
49  automodel_def_4G0N_B_10.pdb  automodel_def_4G0N_B  automodel_default
34  autmodel_slow_3NY5_B_15.pdb  autmodel_slow_3NY5_B     automodel_slow

    mp_score  soap_protein    qmean4  qmean4_z_score  borda_score  borda_rank  \
64      2.21 -45545.746094  0.814469        0.255860        149.5         1.0
77      2.17 -45043.023438  0.775498       -0.340560        143.0         2.0
38      2.42 -48817.878906  0.769190       -0.437096        141.0         3.0
69      2.30 -45205.257812  0.805243        0.114666        138.0         4.0
63      2.26 -44921.707031  0.771055       -0.408556        134.0         5.0
79      2.24 -44596.234375  0.787342       -0.159296        131.5         6.0
72      2.21 -44206.707031  0.796167       -0.024243        128.5         7.0
73      2.39 -44924.730469  0.754554       -0.661071        126.0         8.0
49      2.47 -45311.910156  0.767716       -0.459645        125.0         9.0
34      2.33 -44530.144531  0.720679       -1.179500        124.0        10.0

   template
64     4G0N
77     4G0N
38     3NY5
69     4G0N
63     4G0N
79     4G0N
72     4G0N
73     4G0N
49     4G0N
34     3NY5
```

We can see that most of the best 10 models were generated with the slower routine `Routine_autmodel_slow`. This is to be expected, as this routine spends more time on model refinement and should therefore produce "better" models.

Next, we visualize the results of our evaluation with `seaborn`.

2.7.4 Visualization

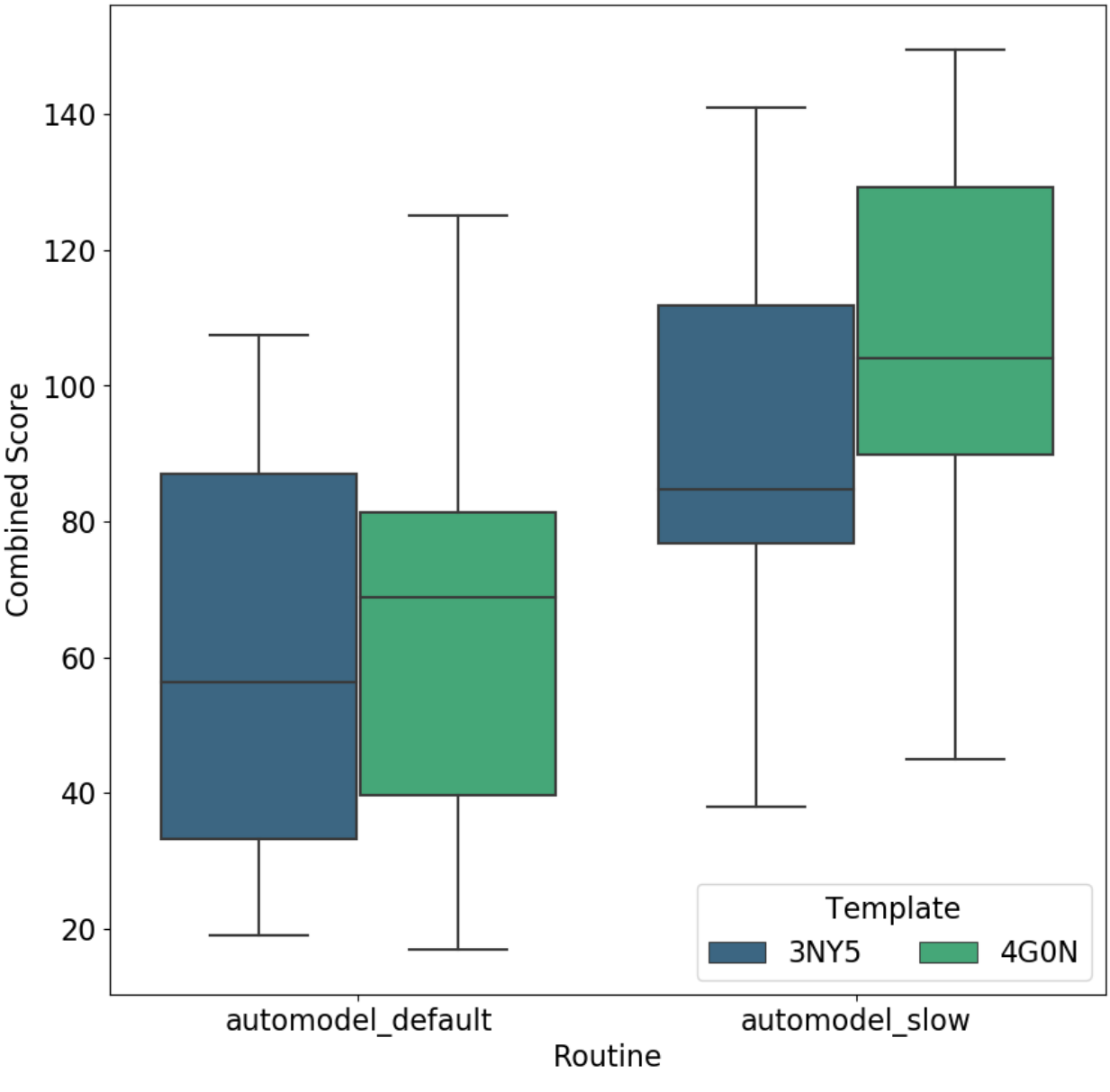
[12]:
```
# visualize combined score with seaborn
%matplotlib inline

# set font size
plt.rcParams.update({'font.size': 16})

plot = sns.boxplot(x = 'routine', y = 'borda_score', hue='template', data=df_eval,
                   palette='viridis')
plot.set(xlabel = 'Routine')
plot.set(ylabel = 'Combined Score')
plot.figure.set_size_inches(10, 10)
```

(continues on next page)

(continued from previous page)

```
plot.legend(title = 'Template', loc = 'lower right', ncol = 2, fancybox = True)
#plot.figure.savefig('tutorial7.png', dpi=300)
```

[12]:
```
<matplotlib.legend.Legend at 0x7f23799902e0>
```



As expected, the routine which spends more time on model refinement (`Routine_automodel_slow`) produces on average better results. Also, there are interesting differences between the templates used.

2.7.5 Further Reading

Congratulations on finishing the final tutorial about `homelette`. You might also be interested in the other tutorials:

- **Tutorial 1**: Learn about the basics of `homelette`.
- **Tutorial 2**: Learn more about already implemented routines for homology modelling.
- **Tutorial 3**: Learn about the evaluation metrics available with `homelette`.
- **Tutorial 4**: Learn about extending `homelette`'s functionality by defining your own modelling routines and evaluation metrics.
- **Tutorial 5**: Learn about how to use parallelization in order to generate and evaluate models more efficiently.
- **Tutorial 6**: Learn about modelling protein complexes.
- **Tutorial 8**: Learn about automated template identification, alignment generation and template processing.

2.7.6 References

[1] Rose, Y., Duarte, J. M., Lowe, R., Segura, J., Bi, C., Bhikadiya, C., Chen, L., Rose, A. S., Bittrich, S., Burley, S. K., & Westbrook, J. D. (2021). RCSB Protein Data Bank: Architectural Advances Towards Integrated Searching and Efficient Access to Macromolecular Structure Data from the PDB Archive. Journal of Molecular Biology, 433(11), 166704. https://doi.org/10.1016/J.JMB.2020.11.003

[2] Steinegger, M., & Söding, J. (2017). MMseqs2 enables sensitive protein sequence searching for the analysis of massive data sets. Nature Biotechnology 2017 35:11, 35(11), 1026–1028. https://doi.org/10.1038/nbt.3988

[3] Sievers, F., Wilm, A., Dineen, D., Gibson, T. J., Karplus, K., Li, W., Lopez, R., McWilliam, H., Remmert, M., Söding, J., Thompson, J. D., & Higgins, D. G. (2011). Fast, scalable generation of high-quality protein multiple sequence alignments using Clustal Omega. Molecular Systems Biology, 7(1), 539. https://doi.org/10.1038/MSB.2011.75

[4] Sievers, F., & Higgins, D. G. (2018). Clustal Omega for making accurate alignments of many protein sequences. Protein Science, 27(1), 135–145. https://doi.org/10.1002/PRO.3290

[5] Šali, A., & Blundell, T. L. (1993). Comparative protein modelling by satisfaction of spatial restraints. Journal of Molecular Biology, 234(3), 779–815. https://doi.org/10.1006/jmbi.1993.1626

[6] Webb, B., & Sali, A. (2016). Comparative Protein Structure Modeling Using MODELLER. Current Protocols in Bioinformatics, 54(1), 5.6.1-5.6.37. https://doi.org/10.1002/cpbi.3

[7] Dong, G. Q., Fan, H., Schneidman-Duhovny, D., Webb, B., Sali, A., & Tramontano, A. (2013). Optimized atomic statistical potentials: Assessment of protein interfaces and loops. Bioinformatics, 29(24), 3158–3166. https://doi.org/10.1093/bioinformatics/btt560

[8] Davis, I. W., Leaver-Fay, A., Chen, V. B., Block, J. N., Kapral, G. J., Wang, X., Murray, L. W., Arendall, W. B., Snoeyink, J., Richardson, J. S., & Richardson, D. C. (2007). MolProbity: all-atom contacts and structure validation for proteins and nucleic acids. Nucleic Acids Research, 35(suppl_2), W375–W383. https://doi.org/10.1093/NAR/GKM216

[9] Chen, V. B., Arendall, W. B., Headd, J. J., Keedy, D. A., Immormino, R. M., Kapral, G. J., Murray, L. W., Richardson, J. S., & Richardson, D. C. (2010). MolProbity: All-atom structure validation for macromolecular crystallography. Acta Crystallographica Section D: Biological Crystallography, 66(1), 12–21. https://doi.org/10.1107/S0907444909042073

[10] Williams, C. J., Headd, J. J., Moriarty, N. W., Prisant, M. G., Videau, L. L., Deis, L. N., Verma, V., Keedy, D. A., Hintze, B. J., Chen, V. B., Jain, S., Lewis, S. M., Arendall, W. B., Snoeyink, J., Adams, P. D., Lovell, S. C., Richardson, J. S., & Richardson, D. C. (2018). MolProbity: More and better reference data for improved all-atom structure validation. Protein Science, 27(1), 293–315. https://doi.org/10.1002/pro.3330

2.7.7 Session Info

[13]:
```
# session info
import session_info
session_info.show(html = False, dependencies = True)
```

```
-----
homelette           1.4
matplotlib          3.1.2
pandas              1.5.3
seaborn             0.12.2
session_info        1.0.0
-----
PIL                 7.0.0
altmod              NA
anyio               NA
asttokens           NA
attr                19.3.0
babel               2.12.1
backcall            0.2.0
certifi             2022.12.07
chardet             3.0.4
charset_normalizer  3.1.0
comm                0.1.2
cycler              0.10.0
cython_runtime      NA
dateutil            2.8.2
debugpy             1.6.6
decorator           4.4.2
executing           1.2.0
fastjsonschema      NA
idna                3.4
importlib_metadata  NA
importlib_resources NA
ipykernel           6.21.3
ipython_genutils    0.2.0
jedi                0.18.2
jinja2              3.1.2
json5               NA
jsonschema          4.17.3
jupyter_events      0.6.3
jupyter_server      2.4.0
jupyterlab_server   2.20.0
kiwisolver          1.0.1
markupsafe          2.1.2
matplotlib_inline   0.1.6
modeller            10.4
more_itertools      NA
mpl_toolkits        NA
nbformat            5.7.3
numexpr             2.8.4
numpy               1.24.2
ost                 2.3.1
```

(continues on next page)

(continued from previous page)

```
packaging           20.3
parso               0.8.3
pexpect             4.8.0
pickleshare         0.7.5
pkg_resources       NA
platformdirs        3.1.1
prometheus_client   NA
promod3             3.2.1
prompt_toolkit      3.0.38
psutil              5.5.1
ptyprocess          0.7.0
pure_eval           0.2.2
pydev_ipython       NA
pydevconsole        NA
pydevd              2.9.5
pydevd_file_utils   NA
pydevd_plugins      NA
pydevd_tracing      NA
pygments            2.14.0
pyparsing           2.4.6
pyrsistent          NA
pythonjsonlogger    NA
pytz                2022.7.1
qmean               NA
requests            2.28.2
rfc3339_validator   0.1.4
rfc3986_validator   0.1.1
scipy               1.10.1
send2trash          NA
sitecustomize       NA
six                 1.12.0
sniffio             1.3.0
stack_data          0.6.2
swig_runtime_data4  NA
tornado             6.2
traitlets           5.9.0
urllib3             1.26.15
wcwidth             NA
websocket           1.5.1
yaml                6.0
zipp                NA
zmq                 25.0.1
-----
IPython             8.11.0
jupyter_client      8.0.3
jupyter_core        5.2.0
jupyterlab          3.6.1
notebook            6.5.3
-----
Python 3.8.10 (default, Nov 14 2022, 12:59:47) [GCC 9.4.0]
Linux-4.15.0-206-generic-x86_64-with-glibc2.29
-----
```

(continues on next page)

(continued from previous page)

```
Session information updated at 2023-03-15 23:50
```

2.8 Tutorial 8: Automatic Alignment Generation

[1]:
```
import homelette as hm
```

2.8.1 Introduction

Welcome to the eighth tutorial for `homelette`, in which we will explore `homelette`'s tool for automated alignment generation.

The alignment is a central step in homology modelling, and the quality of the alignment used for modelling has a lot of influence on the final models. In general, the challenge of creating solid sequence alignments is mainly dependent how closely the target and template are. If they share a high sequence identity, the alignments are easy to construct and the modelling process will most likely be successful.

Note

As a rule of thumb, it is said that everything above 50-60% sequence identity is well approachable, while everything below 30% sequence identity is very challenging to model.

`homelette` has methods that can automatically generate an alignment given a query sequence. However, these methods hide some of the complexity of generating good alignments. Use them at your own discretion, especially for target sequences with low sequence identity to any template.

Note

Be careful with automatically generated alignments if your protein of interest has no closely related templates

---

After these words of caution, let's look at the implemented methods:

- `alignment.AlignmentGenerator_pdb`: Query the PDB and local alignment with Clustal Omega
- `alignment.AlignmentGenerator_hhblits`: Local database search against PDB70 database.
- `alignment.AlignmentGenerator_from_aln`: For if you already have an alignment ready, but want to make use of `homelette`'s processing of templates and alignments.

2.8.2 Method 1: Querying RCSB and Realignment of template sequences with Clusta Omega

This class performs a three step process:

- Template Identification: Query the RCSB using a sequence (interally, MMseq2 is used by RCSB) [1, 2] (`get_suggestion`)
- Then the sequences of identified templates are aligned locally using Clustal Omega [3, 4]. (`get_suggesion`)
- Finally, the template structures are downloaded and processed together with the alignment (`get_pdbs`)

Afterwards, the templates schould be ready for performing homology modelling.

For a practical demonstration, let's find some templates for ARAF:

[2]:
```
gen = hm.alignment.AlignmentGenerator_pdb.from_fasta('data/alignments/ARAF.fa')
# gen = hm.alignment.AlignmentGenerator_pdb(
#     sequence =
↪'GTVKVYLPNKQRTVVTVRDGMSVYDSLDKALKVRGLNQDCCVVYRLIKGRKTVTAWDTAIAPLDGEELIVEVL',
#     target = 'ARAF')
```

There are two ways how `AlignmentGenerator` can be initialized: either with a sequence, or from a fasta file. Both ways are shown above.

In the next step we use this sequence to generate an initial alignment:

[3]:
```
gen.get_suggestion()
```

```
Querying PDB...
Query successful, 16 found!

Retrieving sequences...
Sequences succefully retrieved!

Generating alignment...
Alignment generated!
```

As we can see from the output, we are querying the PDB and extracting potential templates. Then, an alignment is generated.

We can have a first look at the suggested templates as such:

[4]:
```
gen.show_suggestion()
```

[4]:

	template	coverage	identity
0	1C1Y_2	100.0	60.27
1	1GUA_2	100.0	60.27
2	4G0N_2	100.0	60.27
3	4G3X_2	100.0	60.27
4	6VJJ_2	100.0	60.27
5	6XGU_2	100.0	60.27
6	6XGV_2	100.0	60.27
7	6XHA_2	100.0	60.27
8	6XHB_2	100.0	60.27
9	6XI7_2	100.0	60.27
10	7JHP_2	100.0	60.27
11	3KUC_2	100.0	58.90
12	3KUD_2	100.0	58.90
13	3NY5_1	100.0	58.90
14	6NTD_2	100.0	53.42
15	6NTC_2	100.0	52.05

[5]:
```
gen.alignment.print_clustal(70)
```

```
ARAF        -------------GTVKVYLPNKQRTVVTVRDGMSVYDSLDKALKVRGLNQDCCVVYRLI---KGRKTVT
1C1Y_2      ------------SNTIRVFLPNKQRTVVNVRNGMSLHDCLMKALKVRGLQPECCAVFRLLHEHKGKKARL
```

(continues on next page)

(continued from previous page)

```
1GUA_2      --------PSKTSNTIRVFLPNKQRTVVNVRNGMSLHDCLMKALKVRGLQPECCAVFRLLHEHKGKKARL
3KUC_2      --------PSKTSNTIRVFLPNKQRTVVRVRNGMSLHDCLMKKLKVRGLQPECCAVFRLLHEHKGKKARL
3KUD_2      --------PSKTSNTIRVFLPNKQRTVVNVRNGMSLHDCLMKKLKVRGLQPECCAVFRLLHEHKGKKARL
3NY5_1      MGHHHHHHSHMQKPIVRVFLPNKQRTVVPARCGVTVRDSLKKALMMRGLIPECCAVYRIQ---DGEKKPI
4G0N_2      -----------TSNTIRVFLPNKQRTVVNVRNGMSLHDCLMKALKVRGLQPECCAVFRLLHEHKGKKARL
4G3X_2      ------------SNTIRVFLPNKQRTVVNVRNGMSLHDCLMKALKVRGLQPECCAVFRLLHEHKGKKARL
6NTC_2      --------GAMDSNTIRVLLPNQEWTVVKVRNGMSLHDSLMKALKRHGLQPESSAVFRLLHEHKGKKARL
6NTD_2      --------GAMDSNTIRVLLPNHERTVVKVRNGMSLHDSLMKALKRHGLQPESSAVFRLLHEHKGKKARL
6VJJ_2      ---------SKTSNTIRVFLPNKQRTVVNVRNGMSLHDCLMKALKVRGLQPECCAVFRLLHEHKGKKARL
6XGU_2      ---------SKTSNTIRVFLPNKQRTVVNVRNGMSLHDCLMKALKVRGLQPECCAVFRLLHEHKGKKARL
6XGV_2      ---------SKTSNTIRVFLPNKQRTVVNVRNGMSLHDCLMKALKVRGLQPECCAVFRLLHEHKGKKARL
6XHA_2      ---------SKTSNTIRVFLPNKQRTVVNVRNGMSLHDCLMKALKVRGLQPECCAVFRLLHEHKGKKARL
6XHB_2      ---------SKTSNTIRVFLPNKQRTVVNVRNGMSLHDCLMKALKVRGLQPECCAVFRLLHEHKGKKARL
6XI7_2      ---------SKTSNTIRVFLPNKQRTVVNVRNGMSLHDCLMKALKVRGLQPECCAVFRLLHEHKGKKARL
7JHP_2      ------------SNTIRVFLPNKQRTVVNVRNGMSLHDCLMKALKVRGLQPECCAVFRLLHEHKGKKARL


ARAF        AWDTAIAPLDGEELIVEVL----------------------------------------------------
1C1Y_2      DWNTDAASLIGEELQVDFL----------------------------------------------------
1GUA_2      DWNTDAASLIGEELQVDFL----------------------------------------------------
3KUC_2      DWNTDAASLIGEELQVDFL----------------------------------------------------
3KUD_2      DWNTDAASLIGEELQVDFL----------------------------------------------------
3NY5_1      GWDTDISWLTGEELHVEVLENVPLTTHNF------------------------------------------
4G0N_2      DWNTDAASLIGEELQVDFL----------------------------------------------------
4G3X_2      DWNTDAASLIGEELQVDFL----------------------------------------------------
6NTC_2      DWNTDAASLIGEELQVDFL----------------------------------------------------
6NTD_2      DWNTDAASLIGEELQVDFL----------------------------------------------------
6VJJ_2      DWNTDAASLIGEELQVDFL----------------------------------------------------
6XGU_2      DWNTDAASLIGEELQVDFLDHVPLTTHNFARKTFLKLAFCDICQKFLLNGFRCQTCGYKFHEHCSTKVPT
6XGV_2      DWNTDAASLIGEELQVDFLDHVPLTTHNFARKTFLKLAFCDICQKFLLNGFRCQTCGYKFHEHCSTKVPT
6XHA_2      DWNTDAASLIGEELQVDFLDHVPLTTHNFARKTFLKLAFCDICQKFLLNGFRCQTCGYKFHEHCSTKVPT
6XHB_2      DWNTDAASLIGEELQVDFLDHVPLTTHNFARKTFLKLAFCDICQKFLLNGFRCQTCGYKFHEHCSTKVPT
6XI7_2      DWNTDAASLIGEELQVDFLDHVPLTTHNFARKTFLKLAFCDICQKFLLNGFRCQTCGYKFHEHCSTKVPT
7JHP_2      DWNTDAASLIGEELQVDFLDHVPLTTHNFARKTFLKLAFCDICQKFLLNGFRCQTCGYKFHEHCSTKVPT


ARAF        ------
1C1Y_2      ------
1GUA_2      ------
3KUC_2      ------
3KUD_2      ------
3NY5_1      ------
4G0N_2      ------
4G3X_2      ------
6NTC_2      ------
6NTD_2      ------
6VJJ_2      ------
6XGU_2      MCVDWS
6XGV_2      MCVDWS
6XHA_2      MCVDWS
6XHB_2      MCVDWS
6XI7_2      MCVDWS
```

(continues on next page)

(continued from previous page)

```
7JHP_2      MCVDW-
```

After potentially filtering out some sequences, we can proceed with the next step: downloading the structures for our templates, comparing the sequences of the templates with the residues present in the template structure and make adjustments to both the structure and the alignment if necessary.

[6]:
```
gen.get_pdbs()
```

```
Guessing template naming format...
Template naming format guessed: polymer_entity!

Checking template dir...
Template dir found!

Processing templates:

1C1Y downloading from PDB...
1C1Y downloaded!
1C1Y_B: Chain extracted!
1C1Y_B: Alignment updated!
1C1Y_B: PDB processed!
1GUA downloading from PDB...
1GUA downloaded!
1GUA_B: Chain extracted!
1GUA_B: Alignment updated!
1GUA_B: PDB processed!
3KUC downloading from PDB...
3KUC downloaded!
3KUC_B: Chain extracted!
3KUC_B: Alignment updated!
3KUC_B: PDB processed!
3KUD downloading from PDB...
3KUD downloaded!
3KUD_B: Chain extracted!
3KUD_B: Alignment updated!
3KUD_B: PDB processed!
3NY5 downloading from PDB...
3NY5 downloaded!
3NY5_A: Chain extracted!
3NY5_A: Alignment updated!
3NY5_A: PDB processed!
3NY5_B: Chain extracted!
3NY5_B: Alignment updated!
3NY5_B: PDB processed!
3NY5_C: Chain extracted!
3NY5_C: Alignment updated!
3NY5_C: PDB processed!
3NY5_D: Chain extracted!
3NY5_D: Alignment updated!
3NY5_D: PDB processed!
4G0N downloading from PDB...
```

(continues on next page)

(continued from previous page)

```
4G0N downloaded!
4G0N_B: Chain extracted!
4G0N_B: Alignment updated!
4G0N_B: PDB processed!
4G3X downloading from PDB...
4G3X downloaded!
4G3X_B: Chain extracted!
4G3X_B: Alignment updated!
4G3X_B: PDB processed!
6NTC downloading from PDB...
6NTC downloaded!
6NTC_B: Chain extracted!
6NTC_B: Alignment updated!
6NTC_B: PDB processed!
6NTD downloading from PDB...
6NTD downloaded!
6NTD_B: Chain extracted!
6NTD_B: Alignment updated!
6NTD_B: PDB processed!
6VJJ downloading from PDB...
6VJJ downloaded!
6VJJ_B: Chain extracted!
6VJJ_B: Alignment updated!
6VJJ_B: PDB processed!
6XGU downloading from PDB...
6XGU downloaded!
6XGU_B: Chain extracted!
6XGU_B: Alignment updated!
6XGU_B: PDB processed!
6XGV downloading from PDB...
6XGV downloaded!
6XGV_B: Chain extracted!
6XGV_B: Alignment updated!
6XGV_B: PDB processed!
6XHA downloading from PDB...
6XHA downloaded!
6XHA_B: Chain extracted!
6XHA_B: Alignment updated!
6XHA_B: PDB processed!
6XHB downloading from PDB...
6XHB downloaded!
6XHB_B: Chain extracted!
6XHB_B: Alignment updated!
6XHB_B: PDB processed!
6XI7 downloading from PDB...
6XI7 downloaded!
6XI7_B: Chain extracted!
6XI7_B: Alignment updated!
6XI7_B: PDB processed!
7JHP downloading from PDB...
7JHP downloaded!
7JHP_C: Chain extracted!
```

(continues on next page)

(continued from previous page)

```
7JHP_C: Alignment updated!
7JHP_C: PDB processed!

Finishing... All templates successfully
downloaded and processed!
Templates can be found in
"/home/homelette/workdir/templates".
```

`get_pdbs` will check all chains of a template and download those with the correct sequence.

[7]:
```
gen.alignment.print_clustal(70)
```

```
ARAF        -------------GTVKVYLPNKQRTVVTVRDGMSVYDSLDKALKVRGLNQDCCVVYRLI---KGRKTVT
1C1Y_B      ------------SNTIRVFLPNKQRTVVNVRNGMSLHDCLMKALKVRGLQPECCAVFRLLHEHKGKKARL
1GUA_B      ------------NTIRVFLPNKQRTVVNVRNGMSLHDCLMKALKVRGLQPECCAVFRLLHEHKGKKARL
3KUC_B      ------------NTIRVFLPNKQRTVVRVRNGMSLHDCLMKKLKVRGLQPECCAVFRLLHEHKGKKARL
3KUD_B      ------------NTIRVFLPNKQRTVVNVRNGMSLHDCLMKKLKVRGLQPECCAVFRLLHEHKGKKARL
3NY5_A      ---------H-QKPIVRVFLPNKQRTVVPARCGVTVRDSLKKAL--RGLIPECCAVYRIQ------KKPI
3NY5_B      --------SH-QKPIVRVFLPNKQRTVVPARCGVTVRDSLKKAL--RGLIPECCAVYRIQ-----EKKPI
3NY5_C      -----------QKPIVRVFLPNKQRTVVPARCGVTVRDSLKKAL--RGLIPECCAVYRIQ------KKPI
3NY5_D      ---------H-QKPIVRVFLPNKQRTVVPARCGVTVRDSLKKAL--RGLIPECCAVYRI-------KKPI
4G0N_B      -----------TSNTIRVFLPNKQRTVVNVRNGMSLHDCLMKALKVRGLQPECCAVFRLLHEHKGKKARL
4G3X_B      ------------SNTIRVFLPNKQRTVVNVRNGMSLHDCLMKALKVRGLQPECCAVFRLLHEHKGKKARL
6NTC_B      -------------NTIRVLLPNQEWTVVKV---MSLHDSLMKALKRHGLQPESSAVF---------KARL
6NTD_B      ------------SNTIRVLLPNHERTVVKVRNGMSLHDSLMKALKRHGLQPESSAVF-----------RL
6VJJ_B      ------------SNTIRVFLPNKQRTVVNVRNGMSLHDCLMKALKVRGLQPECCAVFRLLHEHKGKKARL
6XGU_B      ------------SNTIRVFLPNKQRTVVNVRNGMSLHDCLMKALKVRGLQPE-CAVFRLLHEHKGKKARL
6XGV_B      ------------SNTIRVFLPNKQRTVVNVRNGMSLHDCLMKALKVRGLQPE-CAVFRLLHEHKGKKARL
6XHA_B      ------------SNTIRVFLPNKQRTVVNVRNGMSLHDCLMKALKVRGLQPE-CAVFRLLHEHKGKKARL
6XHB_B      ------------SNTIRVFLPNKQRTVVNVRNGMSLHDCLMKALKVRGLQPE-CAVFRLLHEHKGKKARL
6XI7_B      -------------NTIRVFLPNKQRTVVNVRNGMSLHDCLMKALKVRGLQPECCAVFRLLH----KKARL
7JHP_C      ------------SNTIRVFLPNKQRTVVNVRNGMSLHDCLMKALKVRGLQPECCAVFRLL-----KKARL


ARAF        AWDTAIAPLDGEELIVEVL-----------------------------------------------------
1C1Y_B      DWNTDAASLIGEELQVDFL-----------------------------------------------------
1GUA_B      DWNTDAASLIGEELQVDFL-----------------------------------------------------
3KUC_B      DWNTDAASLIGEELQVDFL-----------------------------------------------------
3KUD_B      DWNTDAASLIGEELQVDFL-----------------------------------------------------
3NY5_A      GWDTDISWLTGEELHVEVLENVPLT-----------------------------------------------
3NY5_B      GWDTDISWLTGEELHVEVLENVPLTTH---------------------------------------------
3NY5_C      GWDTDISWLTGEELHVEVLENVPLTTH---------------------------------------------
3NY5_D      GWDTDISWLTGEELHVEVLENVPL------------------------------------------------
4G0N_B      DWNTDAASLIGEELQVDFL-----------------------------------------------------
4G3X_B      DWNTDAASLIGEELQVDFL-----------------------------------------------------
6NTC_B      DWNTDAASLIGEELQVDF------------------------------------------------------
6NTD_B      DWNTDAASLIGEELQVD-------------------------------------------------------
6VJJ_B      DWNTDAASLIGEELQVDFL-----------------------------------------------------
6XGU_B      DWNTDAASLIGEELQVDFLDHVPLTTHNFARKTFLKLAFCDICQKFLLNGFRCQTCGYKFHEHCSTKVPT
6XGV_B      DWNTDAASLIGEELQVDFLDHVPLTTHNFARKTFLKLAFCDICQKFLLNGFRCQTCGYKFHEHCSTKVPT
6XHA_B      DWNTDAASLIGEELQVDFLDHVPLTTHNFARKTFLKLAFCDICQKFLLNGFRCQTCGYKFHEHCSTKVPT
6XHB_B      DWNTDAASLIGEELQVDFLDHVPLTTHNFARKTFLKLAFCDICQKFLLNGFRCQTCGYKFHEHCSTKVPT
6XI7_B      DWNTDAASLIGEELQVDFLDHVPLTTHNFARKTFLKLAFCDICQKFLLNGFRCQTCGYKFHEHCSTKVPT
```

(continues on next page)

(continued from previous page)

```
7JHP_C      DWNTDAASLIGEELQVDFLDH--LTTHNFARKTFLKLAFCDICQKFLLNGFRCQTCGYKFHEHCSTKVPT


ARAF        ------
1C1Y_B      ------
1GUA_B      ------
3KUC_B      ------
3KUD_B      ------
3NY5_A      ------
3NY5_B      ------
3NY5_C      ------
3NY5_D      ------
4G0N_B      ------
4G3X_B      ------
6NTC_B      ------
6NTD_B      ------
6VJJ_B      ------
6XGU_B      MCVDWS
6XGV_B      MCVDWS
6XHA_B      MCVDWS
6XHB_B      MCVDWS
6XI7_B      MCV---
7JHP_C      MCVDW-
```

Now we can directly use these template for homology modelling:

[8]:
```
# initialize task
t = gen.initialize_task(task_name = 'Tutorial8', overwrite = True)

# create a model per template
templates = [temp for temp in t.alignment.sequences.keys() if temp != 'ARAF']
for template in templates:
    t.execute_routine(
        tag = f'test_{template}',
        routine = hm.routines.Routine_automodel_default,
        templates = [template],
        template_location = './templates/'
    )
```

[9]:
```
# inspect models
t.models
```

[9]:
```
[<homelette.organization.Model at 0x7f22492f4340>,
 <homelette.organization.Model at 0x7f22492f45b0>,
 <homelette.organization.Model at 0x7f229829a610>,
 <homelette.organization.Model at 0x7f2273b6afa0>,
 <homelette.organization.Model at 0x7f2273b38ee0>,
 <homelette.organization.Model at 0x7f22491c0e50>,
 <homelette.organization.Model at 0x7f22491bf070>,
 <homelette.organization.Model at 0x7f22491bf880>,
```

(continues on next page)

(continued from previous page)

```
 <homelette.organization.Model at 0x7f22491c5760>,
 <homelette.organization.Model at 0x7f22491c5a00>,
 <homelette.organization.Model at 0x7f22491c8310>,
 <homelette.organization.Model at 0x7f22491c8820>,
 <homelette.organization.Model at 0x7f22491b0f10>,
 <homelette.organization.Model at 0x7f22491c96a0>,
 <homelette.organization.Model at 0x7f22491c9b80>,
 <homelette.organization.Model at 0x7f22491c8af0>,
 <homelette.organization.Model at 0x7f22492f49d0>,
 <homelette.organization.Model at 0x7f22491bfbe0>,
 <homelette.organization.Model at 0x7f2273b38040>]
```

2.8.3 Method 2: HHSuite

This class is build on the `hhblits` query function of the HHSuite3 [5].

This has the same interface as `AlignmentGenerator_pdb`, except some different settings for the alignment generation with `get_pdbs`.

It should also be noted that technically, this approach does not generate a multiple sequence alignment, but rather a combined alignment of lots of pairwise alignments of query to template. These pairwise alignments are combined on the common sequence they are all aligned to.

(This code is commented out since it requires big databases to run, which are not part of the docker container.)

[10]:
```
# gen = hm.alignment.AlignmentGenerator_hhblits.from_fasta('data/alignments/ARAF.fa')
# gen.get_suggestion(database_dir='/home/philipp/Downloads/hhsuite_dbs/')
# gen.get_pdbs()
# gen.show_suggestion()
# t = gen.initialize_task()
```

2.8.4 Method 3: Using pre-computed alignments

If you already have an alignment computed, but want to make use of `get_pdbs` in order to download the templates and process the alignment and the template structures, there is also the possibility to load your alignment into an `AlignmentGenerator` object:

[11]:
```
# initialize an alignment generator from a pre-computed alignemnt
gen = hm.alignment.AlignmentGenerator_from_aln(
    alignment_file = 'data/alignments/unprocessed.fasta_aln',
    target = 'ARAF')

gen.show_suggestion()
gen.alignment.print_clustal(70)
gen.get_pdbs()
gen.alignment.print_clustal(70)
```

```
ARAF        -------------GTVKVYLPNKQRTVVTVRDGMSVYDSLDKALKVRGLNQDCCVVYRLI---KGRKTVT
3NY5        MGHHHHHHSHMQKPIVRVFLPNKQRTVVPARCGVTVRDSLKKALMMRGLIPECCAVYRIQ---DGEKKPI
4G0N        -----------TSNTIRVFLPNKQRTVVNVRNGMSLHDCLMKALKVRGLQPECCAVFRLLHEHKGKKARL
```

(continues on next page)

(continued from previous page)

```
ARAF        AWDTAIAPLDGEELIVEVL----------
3NY5        GWDTDISWLTGEELHVEVLENVPLTTHNF
4G0N        DWNTDAASLIGEELQVDFL----------


Guessing template naming format...
Template naming format guessed: entry!

Checking template dir...
Template dir found!

Processing templates:

3NY5 downloading from PDB...
3NY5 downloaded!
3NY5_A: Chain extracted!
3NY5_A: Alignment updated!
3NY5_A: PDB processed!
3NY5_B: Chain extracted!
3NY5_B: Alignment updated!
3NY5_B: PDB processed!
3NY5_C: Chain extracted!
3NY5_C: Alignment updated!
3NY5_C: PDB processed!
3NY5_D: Chain extracted!
3NY5_D: Alignment updated!
3NY5_D: PDB processed!
4G0N downloading from PDB...
4G0N downloaded!
4G0N_B: Chain extracted!
4G0N_B: Alignment updated!
4G0N_B: PDB processed!

Finishing... All templates successfully
downloaded and processed!
Templates can be found in
"./templates/".
ARAF        -------------GTVKVYLPNKQRTVVTVRDGMSVYDSLDKALKVRGLNQDCCVVYRLI---KGRKTVT
3NY5_A      ---------H-QKPIVRVFLPNKQRTVVPARCGVTVRDSLKKAL--RGLIPECCAVYRIQ------KKPI
3NY5_B      --------SH-QKPIVRVFLPNKQRTVVPARCGVTVRDSLKKAL--RGLIPECCAVYRIQ-----EKKPI
3NY5_C      -----------QKPIVRVFLPNKQRTVVPARCGVTVRDSLKKAL--RGLIPECCAVYRIQ------KKPI
3NY5_D      ---------H-QKPIVRVFLPNKQRTVVPARCGVTVRDSLKKAL--RGLIPECCAVYRI-------KKPI
4G0N_B      -----------TSNTIRVFLPNKQRTVVNVRNGMSLHDCLMKALKVRGLQPECCAVFRLLHEHKGKKARL


ARAF        AWDTAIAPLDGEELIVEVL----------
3NY5_A      GWDTDISWLTGEELHVEVLENVPLT----
3NY5_B      GWDTDISWLTGEELHVEVLENVPLTTH--
3NY5_C      GWDTDISWLTGEELHVEVLENVPLTTH--
3NY5_D      GWDTDISWLTGEELHVEVLENVPL-----
4G0N_B      DWNTDAASLIGEELQVDFL----------
```

(continues on next page)

(continued from previous page)

Again, for every template structure, `homelette` is finding which chains fit to the sequence and then extract all of them.

Of course, if your alignment and template(s) are already processed, it is perfectly fine to use the `Alignment` class directly as we have done in the previous tutorials.

2.8.5 Implementing own methods

While not discussed in **Tutorial 4**, `AlignmentGenerator` object are also building blocks in the `homelette` framework and custom versions can be implemented. All `AlignmentGenerator` children classes so far inherit from the `AlignmentGenerator` abstract base class, which contains some useful functionality for writing your own alignment generations, in particular the `get_pdbs` function.

2.8.6 Further Reading

Congratulation on finishing the tutorial about alignment generation in `homelette`.

Please note that there are other tutorials, which will teach you more about how to use `homelette`.

- **Tutorial 1**: Learn about the basics of `homelette`.
- **Tutorial 2**: Learn more about already implemented routines for homology modelling.
- **Tutorial 3**: Learn about the evaluation metrics available with `homelette`.
- **Tutorial 4**: Learn about extending `homelette`'s functionality by defining your own modelling routines and evaluation metrics.
- **Tutorial 5**: Learn about how to use parallelization in order to generate and evaluate models more efficiently.
- **Tutorial 6**: Learn about modelling protein complexes.
- **Tutorial 7**: Learn about assembling custom pipelines.

2.8.7 References

[1] Rose, Y., Duarte, J. M., Lowe, R., Segura, J., Bi, C., Bhikadiya, C., Chen, L., Rose, A. S., Bittrich, S., Burley, S. K., & Westbrook, J. D. (2021). RCSB Protein Data Bank: Architectural Advances Towards Integrated Searching and Efficient Access to Macromolecular Structure Data from the PDB Archive. Journal of Molecular Biology, 433(11), 166704. https://doi.org/10.1016/J.JMB.2020.11.003

[2] Steinegger, M., & Söding, J. (2017). MMseqs2 enables sensitive protein sequence searching for the analysis of massive data sets. Nature Biotechnology 2017 35:11, 35(11), 1026–1028. https://doi.org/10.1038/nbt.3988

[3] Sievers, F., Wilm, A., Dineen, D., Gibson, T. J., Karplus, K., Li, W., Lopez, R., McWilliam, H., Remmert, M., Söding, J., Thompson, J. D., & Higgins, D. G. (2011). Fast, scalable generation of high-quality protein multiple sequence alignments using Clustal Omega. Molecular Systems Biology, 7(1), 539. https://doi.org/10.1038/MSB.2011.75

[4] Sievers, F., & Higgins, D. G. (2018). Clustal Omega for making accurate alignments of many protein sequences. Protein Science, 27(1), 135–145. https://doi.org/10.1002/PRO.3290

[5] Steinegger, M., Meier, M., Mirdita, M., Vöhringer, H., Haunsberger, S. J., & Söding, J. (2019). HH-suite3 for fast remote homology detection and deep protein annotation. BMC Bioinformatics, 20(1), 1–15. https://doi.org/10.1186/S12859-019-3019-7/FIGURES/7

2.8.8 Session Info

[12]:
```
# session info
import session_info
session_info.show(html = False, dependencies = True)
```

```
-----
homelette           1.4
pandas              1.5.3
session_info        1.0.0
-----
PIL                 7.0.0
altmod              NA
anyio               NA
asttokens           NA
attr                19.3.0
babel               2.12.1
backcall            0.2.0
certifi             2022.12.07
chardet             3.0.4
charset_normalizer  3.1.0
comm                0.1.2
cycler              0.10.0
cython_runtime      NA
dateutil            2.8.2
debugpy             1.6.6
decorator           4.4.2
executing           1.2.0
fastjsonschema      NA
idna                3.4
importlib_metadata  NA
importlib_resources NA
ipykernel           6.21.3
ipython_genutils    0.2.0
jedi                0.18.2
jinja2              3.1.2
json5               NA
jsonschema          4.17.3
jupyter_events      0.6.3
jupyter_server      2.4.0
jupyterlab_server   2.20.0
kiwisolver          1.0.1
markupsafe          2.1.2
matplotlib          3.1.2
modeller            10.4
more_itertools      NA
mpl_toolkits        NA
nbformat            5.7.3
numexpr             2.8.4
numpy               1.24.2
ost                 2.3.1
packaging           20.3
parso               0.8.3
```

(continues on next page)

(continued from previous page)

```
pexpect             4.8.0
pickleshare         0.7.5
pkg_resources       NA
platformdirs        3.1.1
prometheus_client   NA
promod3             3.2.1
prompt_toolkit      3.0.38
psutil              5.5.1
ptyprocess          0.7.0
pure_eval           0.2.2
pydev_ipython       NA
pydevconsole        NA
pydevd              2.9.5
pydevd_file_utils   NA
pydevd_plugins      NA
pydevd_tracing      NA
pygments            2.14.0
pyparsing           2.4.6
pyrsistent          NA
pythonjsonlogger    NA
pytz                2022.7.1
qmean               NA
requests            2.28.2
rfc3339_validator   0.1.4
rfc3986_validator   0.1.1
send2trash          NA
sitecustomize       NA
six                 1.12.0
sniffio             1.3.0
stack_data          0.6.2
swig_runtime_data4  NA
tornado             6.2
traitlets           5.9.0
urllib3             1.26.15
wcwidth             NA
websocket           1.5.1
yaml                6.0
zipp                NA
zmq                 25.0.1
-----
IPython             8.11.0
jupyter_client      8.0.3
jupyter_core        5.2.0
jupyterlab          3.6.1
notebook            6.5.3
-----
Python 3.8.10 (default, Nov 14 2022, 12:59:47) [GCC 9.4.0]
Linux-4.15.0-206-generic-x86_64-with-glibc2.29
-----
Session information updated at 2023-03-15 23:40
```

CHAPTER

THREE

API DOCUMENTATION

This is the documentation for all classes, methods and functions in homelette.

3.1 homelette.organization

The *homelette.organization* submodule contains classes for organizing workflows.

Task is an object orchestrating model generation and evaluation.

Model is an object used for storing information about generated models.

3.1.1 Tutorials

For an introduction to *homelette*'s workflow, *Tutorial 1* is useful. Assembling custom pipelines is discussed in *Tutorial 7*.

3.1.2 Classes

The following classes are part of this submodule:

Task *Model*

---

class homelette.organization.**Task**(*task_name: str*, *target: str*, *alignment: Type[*Alignment*]*, *task_directory: str = None*, *overwrite: bool = False*)

Class for directing modelling and evaluation.

It is designed for the modelling of one target sequence from one or multiple templates.

If an already existing folder with models is specified, the Task object will load those models in automatically. In this case, it can also be used exclusively for evaluation purposes.

Parameters

- **task_name** (*str*) – The name of the task
- **target** (*str*) – The identifier of the protein to model
- **alignment** (Alignment) – The alignment object that will be used for modelling
- **task_directory** (*str, optional*) – The directory that will be used for this modelling task (default is creating a new one based on the task_name)

- **overwrite** (*bool, optional*) – Boolean value determining if an already existing task_directory should be overwriten. If a directory already exists for a given task_name or task_directory, this will determine whether the directory and all its contents will be overwritten (True), or whether the contained models will be imported (False) (default is False)

Variables

- **task_name** (*str*) – The name of the task
- **task_directory** (*str*) – The directory that will be used for this modelling task (default is to use the task_name)
- **target** (*str*) – The identifier of the protein to model
- **alignment** (*Alignment*) – The alignment object that will be used for modelling
- **models** (*list*) – List of models generated or imported by this task
- **routines** (*list*) – List of modelling routines executed by this task

Return type
None

execute_routine(*tag: str*, *routine: Type[routines.Routine]*, *templates: Iterable*, *template_location: str = '.'*, ***kwargs*) → None

Generates homology models using a specified modelling routine

Parameters

- **tag** (*str*) – The identifier associated with this combination of routine and template(s). Has to be unique between all routines executed by the same task object
- **routine** (*Routine*) – The routine object used to generate the models
- **templates** (*list*) – The iterable containing the identifier(s) of the template(s) used for model generation
- **template_location** (*str, optional*) – The location of the template PDB files. They should be named according to their identifiers in the alignment (i.e. for a sequence named "1WXN" to be used as a template, it is expected that there will be a PDB file named "1WXN.pdb" in the specified template location (default is current working directory)
- ****kwargs** – Named parameters passed directly on to the Routine object when the modelling is performed. Please check the documentation in order to make sure that the parameters passed on are available with the Routine object you intend to use

Return type
None

evaluate_models(**args: Type[evaluation.Evaluation]*, *n_threads: int = 1*) → None

Evaluates models using one or multiple evaluation metrics

Parameters

- ***args** (*Evaluation*) – Evaluation objects that will be applied to the models
- **n_threads** (*int, optional*) – Number of threads used for model evaluation (default is 1, which deactivates parallelization)

Return type
None

get_evaluation() → pandas.DataFrame

Return evaluation for all models as pandas dataframe.

Returns
Dataframe containing all model evaluation

Return type
pd.DataFrame

class homelette.organization.**Model**(*model_file: str*, *tag: str*, *routine: str*)

Interface used to interact with created protein structure models.

Parameters

- **model_file** (*str*) – The file location of the PDB file for this model
- **tag** (*str*) – The tag that was used when generating this model (see Task.execute_routine for more details)
- **routine** (*str*) – The name of the routine that was used to generate this model

Variables

- **model_file** (*str*) – The file location of the PDB file for this model
- **tag** (*str*) – The tag that was used when generating this model (see Task.execute_routine for more details)
- **routine** (*str*) – The name of the routine that was used to generate this model
- **info** (*dict*) – Dictionary that can be used to store metadata about the model (i.e. for some evaluation metrics)

Return type
None

parse_pdb() → pandas.DataFrame

Parses ATOM and HETATM records in PDB file to pandas dataframe Useful for giving some evaluations methods access to data from the PDB file.

Return type
pd.DataFrame

Notes

Information is extracted according to the PDB file specification (version 3.30) and columns are named accordingly. See https://www.wwpdb.org/documentation/file-format for more information.

get_sequence() → str

Retrieve the 1-letter amino acid sequence of the PDB file associated with the Model object.

Returns
Amino acid sequence

Return type
str

rename(*new_name: str*) → None

Rename the PDB file associated with the Model object.

Parameters
new_name (*str*) – New name of PDB file

Return type
None

3.2 homelette.alignment

The *homelette.alignment* submodule contains a selection of tools for handling sequences and alignments, as well as for the automatic generation of sequences from a target sequence.

3.2.1 Tutorials

Basic handing of alignments with *homelette* is demonstrated in *Tutorial 1*. The assembling of alignments for complex modelling is discussed in *Tutorial 6*. The automatic generation of alignments is shown in *Tutorial 8*.

3.2.2 Functions and classes

Functions and classes present in *homelette.alignment* are listed below:

Alignment *Sequence* *AlignmentGenerator* *AlignmentGenerator_pdb* *AlignmentGenerator_hhblits* *AlignmentGenerator_from_aln* *assemble_complex_aln()*

---

class homelette.alignment.**Alignment**(*file_name: Optional[str] = None*, *file_format: str = 'fasta'*)

Bases: object

Class for managing sequence alignments.

Parameters

- **file_name** (*str, optional*) – The file to read the alignment from. If no file name is given, an empty alignment object will be created (default None)
- **file_format** (*str, optional*) – The format of the alignment file. Can be 'fasta' or 'pir' (default 'fasta')

Variables
sequences (*dict*) – Collection of sequences. Sequences names are the dictionary keys, Sequence objects the values

Raises
ValueError – File_format specified is not 'fasta' or 'pir'

get_sequence(*sequence_name: str*) → Type[*Sequence*]

Retrieve sequence object by sequence name.

Parameters
sequence_name (*str*) – Name of sequence to retrieve

Return type
Sequence

select_sequences(*sequence_names: Iterable*) → None

Select sequences to remain in the alignment by sequence name

Parameters
sequence_names (*iterable*) – Iterable of sequence names

Return type
None

Raises
KeyError – Sequence name not found in alignment

remove_sequence(*sequence_name: str*) → None

Remove a sequence from the alignment by sequence name.

Parameters
sequence_name (*str*) – Sequence name to remove from alignment

Return type
None

rename_sequence(*old_name: str*, *new_name: str*) → None

Rename sequence in the alignment

Parameters

- **old_name** (*str*) – Old name of sequence
- **new_name** (*str*) – New name of sequence

Return type
None

write_pir(*file_name: str*, *line_wrap: int = 50*) → None

Write alignment to file in the PIR file format.

Parameters

- **file_name** (*str*) – File name to write to
- **line_wrap** (*int*) – Characters per line (default 50)

Return type
None

write_fasta(*file_name: str*, *line_wrap: int = 80*) → None

Write alignment to file in the FASTA alignment file format.

Parameters

- **file_name** (*str*) – File name to write to
- **line_wrap** (*int*) – Characters per line (default 80)

Return type
None

print_clustal(*line_wrap: int = 80*) → None

Print alignment to console in the clustal file format.

Parameters
line_wrap (*int*) – Characters per line (default 80)

Return type
None

write_clustal(*file_name: str*, *line_wrap: int = 50*) → None

Write alignment to file in the clustal file format.

Parameters

- **file_name** (*str*) – File name to write to
- **line_wrap** (*int*) – Characters per line (default 50)

Return type
None

remove_redundant_gaps() → None

Remove gaps in the alignment that are present in every column.

Return type
None

replace_sequence(*seq_name: str*, *new_sequence: str*) → None

Targeted replacement of sequence in alignment.

Parameters

- **seq_name** (*str*) – The identifier of the sequence that will be replaced.
- **new_sequence** (*str*) – The new sequence.

Notes

This replacement is designed to introduce missing residues from template structures into the alignment and therefore has very strict requirements. The new and old sequence have to be identical, except that the new sequence might contain unmodelled residues. These are indicated by the letter 'X' in the new sequence, and will result in a gap '-' in the alignment after replacement. It is important that all unmodelled residues, even at the start or beginning of the template sequence are correctly labeled as 'X'.

Examples

```
>>> aln = hm.Alignment(None)
>>> aln.sequences = {
...     'seq1': hm.alignment.Sequence('seq1', 'AAAACCCCDDDD'),
...     'seq2': hm.alignment.Sequence('seq2', 'AAAAEEEEDDDD'),
...     'seq3': hm.alignment.Sequence('seq3', 'AAAA----DDDD')
...     }
>>> replacement_seq1 = 'AAAAXXXXXDDD'
>>> replacement_seq3 = 'AAXXXXDD'
>>> aln.replace_sequence('seq1', replacement_seq1)
>>> aln.print_clustal()
seq1        AAAA-----DDD
seq2        AAAAEEEEDDDD
seq3        AAAA----DDDD
>>> aln.replace_sequence('seq3', replacement_seq3)
>>> aln.print_clustal()
seq1        AAAA-----DDD
seq2        AAAAEEEEDDDD
seq3        AA--------DD
```

calc_identity(*sequence_name_1: str*, *sequence_name_2: str*) → float

Calculate sequence identity between two sequences in the alignment.

Parameters

- **sequence_name_1** (*str*) – Sequence pair to calculate identity for

- **sequence_name_2** (*str*) – Sequence pair to calculate identity for

Returns
identity – Sequence identity between the two sequences

Return type
float

See also:

calc_identity_target, *calc_pairwise_identity_all*

Notes

There are mutiple ways of calculating sequence identity, which can be useful in different situations. Here implemented is one way which makes a lot of sence for evaluating templates for homology modelling. The sequence identity is calculated by dividing the number of matches by the length of sequence 1 (mismatches and gaps are handled identically, no gap compression).

$$\text{seqid} = \frac{\text{matches}}{\text{length}(\text{sequence1})}$$

Examples

Gaps and mismatches are treated equally.

```
>>> aln = hm.Alignment(None)
>>> aln.sequences = {
...     'seq1': hm.alignment.Sequence('seq1', 'AAAACCCCDDDD'),
...     'seq2': hm.alignment.Sequence('seq2', 'AAAAEEEEDDDD'),
...     'seq3': hm.alignment.Sequence('seq3', 'AAAA----DDDD')
...     }
>>> aln.calc_identity('seq1', 'seq2')
66.67
>>> aln.calc_identity('seq1', 'seq3')
66.67
```

Normalization happens for the length of sequence 1, so the order of sequences matters.

```
>>> aln = hm.Alignment(None)
>>> aln.sequences = {
...     'seq1': hm.alignment.Sequence('seq1', 'AAAACCCCDDDD'),
...     'seq2': hm.alignment.Sequence('seq3', 'AAAA----DDDD')
...     }
>>> aln.calc_identity('seq1', 'seq2')
66.67
>>> aln.calc_identity('seq2', 'seq1')
100.0
```

calc_pairwise_identity_all() → Type[pandas.DataFrame]

Calculate identity between all sequences in the alignment.

Returns
identities – Dataframe with pairwise sequence identites

Return type
pd.DataFrame

See also:

calc_identity, *calc_identity_target*

Notes

Calculates sequence identity as descripted for calc_identity:

$$\text{seqid} = \frac{\text{matches}}{\text{length}(\text{sequence1})}$$

calc_identity_target(*sequence_name: str*) → Type[pandas.DataFrame]

Calculate identity of all sequences in the alignment to specified target sequence.

Parameters
sequence_name (*str*) – Target sequence

Returns
identities – Dataframe with pairwise sequence identities

Return type
pd.DataFrame

See also:

calc_identity, *calc_pairwise_identity_all*

Notes

Calculates sequence identity as descripted for calc_identity:

$$\text{seqid} = \frac{\text{matches}}{\text{length}(\text{sequence1})}$$

calc_coverage(*sequence_name_1: str*, *sequence_name_2: str*) → float

Calculation of coverage of sequence 2 to sequence 1 in the alignment.

Parameters

- **sequence_name_1** (*str*) – Sequence pair to calculate coverage for
- **sequence_name_2** (*str*) – Sequence pair to calculate coverage for

Returns
coverage – Coverage of sequence 2 to sequence 1

Return type
float

See also:

calc_coverage_target, *calc_pairwise_coverage_all*

Notes

Coverage in this context means how many of the residues in sequences 1 are assigned a residue in sequence 2. This is useful for evaluating potential templates, because a low sequence identity (as implemented in homelette) could be caused either by a lot of residues not being aligned at all, or a lot of residues being aligned but not with identical residues.

$$\text{coverage} = \frac{\text{aligned residues}}{\text{length(sequence1)}}$$

Examples

Gaps and mismatches are not treated equally.

```
>>> aln = hm.Alignment(None)
>>> aln.sequences = {
...     'seq1': hm.alignment.Sequence('seq1', 'AAAACCCCDDDD'),
...     'seq2': hm.alignment.Sequence('seq2', 'AAAAEEEEDDDD'),
...     'seq3': hm.alignment.Sequence('seq3', 'AAAA----DDDD')
...     }
>>> aln.calc_coverage('seq1', 'seq2')
100.0
>>> aln.calc_coverage('seq1', 'seq3')
66.67
```

Normalization happens for the length of sequence 1, so the order of sequences matters.

```
>>> aln = hm.Alignment(None)
>>> aln.sequences = {
...     'seq1': hm.alignment.Sequence('seq1', 'AAAACCCCDDDD'),
...     'seq2': hm.alignment.Sequence('seq3', 'AAAA----DDDD')
...     }
>>> aln.calc_coverage('seq1', 'seq2')
66.67
>>> aln.calc_coverage('seq2', 'seq1')
100.0
```

calc_coverage_target(*sequence_name: str*) → Type[pandas.DataFrame]

Calculate coverage of all sequences in the alignment to specified target sequence.

Parameters
sequence_name (*str*) – Target sequence

Returns
coverages – Dataframe with pairwise coverage

Return type
pd.DataFrame

See also:

calc_coverage, *calc_pairwise_coverage_all*

Notes

Calculates coverage as described for calc_coverage:

$$\text{coverage} = \frac{\text{aligned residues}}{\text{length(sequence1)}}$$

calc_pairwise_coverage_all() → Type[pandas.DataFrame]

Calculate coverage between all sequences in the alignment.

Returns
coverages – Dataframe with pairwise coverage

Return type
pd.DataFrame

See also:

calc_coverage, *calc_coverage_target*

Notes

Calculates coverage as described for calc_coverage:

$$\text{coverage} = \frac{\text{aligned residues}}{\text{length(sequence1)}}$$

class homelette.alignment.**Sequence**(*name: str*, *sequence: str*, ***kwargs*)

Bases: object

Class that contains individual sequences and miscellaneous information about them.

Parameters

- **name** (*str*) – Identifier of the sequence
- **sequence** (*str*) – Sequence in 1 letter amino acid code
- ****kwargs** – Annotations, for acceptable keys see *Sequence.annotate()*

Variables

- **name** (*str*) – Identifier of the sequence
- **sequence** (*str*) – Sequence in 1 letter amino acid code
- **annotation** (*dict*) – Collection of annotation for this sequence

Notes

See *Sequence.annotate()* for more information on the annotation of sequences.

annotate(***kwargs: str*)

Change annotation for sequence object.

Keywords not specified in the Notes section will be ignored.

Parameters
kwargs (*str*) – Annotations. For acceptible values, see Notes.

Return type
None

Notes

Annotations are important for MODELLER in order to properly process alignment in PIR format. The following annotations are supported and can be modified.

annotation	explanation
seq_type	Specification whether sequence should be treated as a template (set to 'structure') or as a target (set to 'sequence')
pdb_code	PDB code corresponding to sequence (if available)
begin_res	Residue number for the first residue of the sequence in the corresponing PDB file
begin_chain	Chain identifier for the first residue of the sequence in the corresponding PDB file
end_res	Residue number for the last residue of the sequence in the corresponding PDB file
end_chain	Chain identifier for the last residue of the sequence in the corresponding PDB file
prot_name	Protein name, optional
prot_source	Protein source, optional
resolution	Resolution of PDB structure, optional
R_factor	R-factor of PDB structure, optional

Different types of annotations are required, depending whether a target or a template is annotated. For *targets*, it is sufficient to seq the seq_type to 'sequence'. For *templates*, it is required by MODELLER that seq_type and pdb_code are annotated. begin_res, begin_chain, end_res and end_chain are recommended. The rest can be left unannoted.

Examples

Annotation for a target sequence.

```
>>> target = hm.alignment.Sequence(name = 'target', sequence =
...     'TARGET')
>>> target.annotation
{'seq_type': '', 'pdb_code': '', 'begin_res': '', 'begin_chain': '',
'end_res': '', 'end_chain': '', 'prot_name': '', 'prot_source': '',
'resolution': '', 'r_factor': ''}
>>> target.annotate(seq_type = 'sequence')
>>> target.annotation
{'seq_type': 'sequence', 'pdb_code': '', 'begin_res': '',
'begin_chain': '', 'end_res': '', 'end_chain': '', 'prot_name': '',
'prot_source': '', 'resolution': '', 'r_factor': ''}
```

Annotation for a template structure.

```
>>> template = hm.alignment.Sequence(name = 'template', sequence =
...     'TEMPLATE')
>>> template.annotation
{'seq_type': '', 'pdb_code': '', 'begin_res': '', 'begin_chain': '',
'end_res': '', 'end_chain': '', 'prot_name': '', 'prot_source': '',
'resolution': '', 'r_factor': ''}
>>> template.annotate(seq_type = 'structure', pdb_code = 'TMPL',
```

(continues on next page)

(continued from previous page)

```
...     begin_res = '1', begin_chain = 'A', end_res = '8', end_chain =
...     'A')
>>> template.annotation
{'seq_type': 'structure', 'pdb_code': 'TMPL', 'begin_res': '1',
'begin_chain': 'A', 'end_res': '8', 'end_chain': 'A', 'prot_name': '',
'prot_source': '', 'resolution': '', 'r_factor': ''}
```

get_annotation_pir() → str

Return annotation in the colon-separated format expected from the PIR alignment format used by MODELLER.

Returns
Annotation in PIR format

Return type
str

Examples

```
>>> template = hm.alignment.Sequence(name = 'template', sequence =
...     'TEMPLATE', seq_type = 'structure', pdb_code = 'TMPL',
...     begin_res = '1', begin_chain = 'A', end_res = '8', end_chain =
...     'A')
>>> template.get_annotation_pir()
'structure:TMPL:1:A:8:A::::'
```

get_annotation_print() → None

Print annotation to console

Return type
None

Examples

```
>>> template = hm.alignment.Sequence(name = 'template', sequence =
...     'TEMPLATE', seq_type = 'structure', pdb_code = 'TMPL',
...     begin_res = '1', begin_chain = 'A', end_res = '8', end_chain =
...     'A')
>>> template.get_annotation_print()
Sequence Type   structure
PDB ID          TMPL
Start Residue   1
Start Chain     A
End Residue     8
End Chain       A
Protein Name
Protein Source
Resolution
R-Factor
```

get_gaps() → tuple

Find gap positions in sequence

Returns
Positions of gaps in sequence

Return type
tuple

Examples

```
>>> seq = hm.alignment.Sequence(name = 'seq', sequence = 'SEQ-UEN--CE')
>>> seq.get_gaps()
(3, 7, 8)
```

remove_gaps(*remove_all: bool = False*, *positions: Optional[Iterable[int]] = None*) → None

Remove gaps from the sequence.

Gaps in the alignment are symbolized by '-'. Removal can either happen at specific or all positions. Indexing for specific positions is zero-based and checked before removal (raises Warning if the attempted removal of a non-gap position is detected)

Parameters

- **remove_all** (*bool*) – Remove all gaps (default False)
- **positions** (*iterable*) – Positions to remove (zero-based indexing)

Return type
None

Warns
UserWarning – Specified position is not a gap

Examples

Example 1: remove all

```
>>> seq = hm.alignment.Sequence(name = 'seq', sequence = 'SEQ-UEN--CE')
>>> seq.remove_gaps(remove_all = True)
>>> seq.sequence
'SEQUENCE'
```

Example 2: selective removal

```
>>> seq = hm.alignment.Sequence(name = 'seq', sequence = 'SEQ-UEN--CE')
>>> seq.remove_gaps(positions = (7, 8))
>>> seq.sequence
'SEQ-UENCE'
```

class homelette.alignment.**AlignmentGenerator**(*sequence: str*, *target: str = 'target'*, *template_location: str = './templates/'*)

Bases: ABC

Parent class for the auto-generation of alignments and template selection based on sequence input.

Parameters

- **sequence** (*str*) – Target sequence in 1 letter amino acid code.

- **target** (*str*) – The name of the target sequence (default "target"). If longer than 14 characters, will be truncated.
- **template_location** (*str*) – Directory where processed templates will be stored (default "./templates/").

Variables

- **alignment** (Alignment) – The alignment.
- **target_seq** (*str*) – The target sequence.
- **target** (*str*) – The name of the target sequence.
- **template_location** (*str*) – Directory where processed templates will be stored.
- **state** – Dictionary describing the state of the AlignmentGenerator object

Return type
None

abstract get_suggestion()

Generate suggestion for templates and alignment

classmethod from_fasta(*fasta_file: str*, *template_location: str = './templates/'*) → *AlignmentGenerator*

Generates an instance of the AlignemntGenerator with the first sequence in the fasta file.

Parameters

- **fasta_file** (*str*) – Fasta file from which the first sequence will be read.
- **template_location** (*str*) – Directory where processed templates will be stored (default "./templates/").

Return type
AlignmentGenerator

Raises
ValueError – Fasta file not properly formatted

show_suggestion(*get_metadata: bool = False*) → Type[pandas.DataFrame]

Shows which templates have been suggested by the AlignmentGenerator, as well as some useful statistics (sequence identity, coverage).

Parameters
get_metadata (*bool*) – Retrieve additional metadata (experimental method, resolution, structure title) from the RCSB.

Returns
suggestion – DataFrame with calculated sequence identity and sequence coverage for target

Return type
pd.DataFrame

Raises
RuntimeError – Alignment has not been generated yet

See also:

Alignment.calc_identity, *Alignment.calc_coverage*

Notes

The standard output lists the templates in the alignment and shows both coverage and sequence identity to the target sequence. The templates are ordered by sequence identity.

In addition, the experimental method (Xray, NMR or Electron Microscopy), the resolution (if applicable) and the title of the template structure can be retrieved from the RCSB. Retrieving metadata from the PDB requires a working internet connecction.

select_templates(*templates: Iterable*) → None

Select templates from suggested templates by identifier.

Parameters
templates (*iterable*) – The selected templates as an interable.

Return type
None

Raises
RuntimeError – Alignment has not been generated yet

get_pdbs(*pdb_format: str = 'auto'*, *verbose: bool = True*) → None

Downloads and processes templates present in alignment.

Parameters

- **pdb_format** (*str*) – Format of PDB identifiers in alignment (default auto)
- **verbose** (*bool*) – Explain what operations are performed

Raises

- **RuntimeError** – Alignment has not been generated yet
- **ValueError** – PDB format could not be guessed

Notes

pdb_format tells the function how to parse the template identifiers in the alignment:

- auto: Automatic guess for pdb_format
- entry: Sequences are named only be their PDB identifier (i.e. 4G0N)
- entity: Sequences are named in the format PDBID_ENTITY (i.e. 4G0N_1)
- instance: Sequences are named in the format PDBID_CHAIN (i.e. 4G0N_A)

Please make sure that all templates follow one naming convention, and that there are no sequences in the alignment that violate the naming convention (except the target sequence).

During the template processing, all hetatms will be remove from the template, as well as all other chains. All chains will be renamed to "A" and the residue number will be set to 1 on the first residue. The corresponding annotations are automatically made in the alignment object.

initialize_task(*task_name: ~typing.Optional[str] = None*, *overwrite: bool = False*, *task_class: ~homelette.organization.Task = <class 'homelette.organization.Task'>*) → *Task*

Initialize a homelette Task object for model generation and evaluation.

Parameters

- **task_name** (*str*) – The name of the task to initialize. If None, initialize as models_{target}.

- **overwrite** (*bool*) – Whether to overwrite the task directory if a directory of the same name already exists (default False).
- **task_class** (Task) – The class to initialize the Task with. This makes it possible to define custom child classes of Task and construct them from this function (default Task)

Return type
Task

Raises
RuntimeError – Alignment has not been generated or templates have not been downloaded and processed.

class homelette.alignment.**AlignmentGenerator_pdb**(*sequence: str*, *target: str = 'target'*, *template_location: str = './templates/'*)

Bases: *AlignmentGenerator*

Identification of templates using the RCSB search API, generation of alignment using Clustal Omega and download and processing of template structures.

Parameters

- **sequence** (*str*) – Target sequence in 1 letter amino acid code.
- **target** (*str*) – The name of the target sequence (default "target"). If longer than 14 characters, will be truncated.
- **template_location** (*str*) – Directory where processed templates will be stored (default "./templates/").

Variables

- **alignment** (Alignment) – The alignment.
- **target_seq** (*str*) – The target sequence.
- **target** (*str*) – The name of the target sequence.
- **template_location** (*str*) – Directory where processed templates will be stored.
- **state** – Dictionary describing the state of the AlignmentGenerator object

Return type
None

Notes

The AlignmentGenerator uses the RCSB Search API[1] to identify potential template structures given the target sequence using MMseq2[2]. The sequences of the potentially downloaded and locally aligned using Clustal Omega[3][4].

[1] Rose, Y., Duarte, J. M., Lowe, R., Segura, J., Bi, C., Bhikadiya, C., Chen, L., Rose, A. S., Bittrich, S., Burley, S. K., & Westbrook, J. D. (2021). RCSB Protein Data Bank: Architectural Advances Towards Integrated Searching and Efficient Access to Macromolecular Structure Data from the PDB Archive. Journal of Molecular Biology, 433(11), 166704. https://doi.org/10.1016/J.JMB.2020.11.003

[2] Steinegger, M., & Söding, J. (2017). MMseqs2 enables sensitive protein sequence searching for the analysis of massive data sets. Nature Biotechnology 2017 35:11, 35(11), 1026–1028. https://doi.org/10.1038/nbt.3988

[3] Sievers, F., Wilm, A., Dineen, D., Gibson, T. J., Karplus, K., Li, W., Lopez, R., McWilliam, H., Remmert, M., Söding, J., Thompson, J. D., & Higgins, D. G. (2011). Fast, scalable generation of high-quality protein multiple sequence alignments using Clustal Omega. Molecular Systems Biology, 7(1), 539. https://doi.org/10.1038/MSB.2011.75

[4] Sievers, F., & Higgins, D. G. (2018). Clustal Omega for making accurate alignments of many protein sequences. Protein Science, 27(1), 135–145. https://doi.org/10.1002/PRO.3290

References

get_suggestion(*seq_id_cutoff: float = 0.5*, *min_length: int = 30*, *max_results: int = 50*, *xray_only: bool = True*, *verbose: bool = True*) → None

Identifies potential templates, retrieves their sequences and aligns them locally using Clustal Omega.

Parameters

- **seq_id_cutoff** (*float*) – The sequence identity cutoff for the identification of template structures. Templates below this threshold will be ignored (default 0.5).
- **min_length** (*int*) – The minimum length of template sequence to be included in the results (default 30 amino acids).
- **max_results** (*int*) – The number of results returned (default 50).
- **xray_only** (*bool*) – Only consider templates structures generated with X-ray crystallography (default True).
- **verbose** (*bool*) – Explain what is done (default True).

Return type
None

Raises
RuntimeError – Alignment already generated.

classmethod from_fasta(*fasta_file: str*, *template_location: str = './templates/'*) → *AlignmentGenerator*

Generates an instance of the AlignemntGenerator with the first sequence in the fasta file.

Parameters

- **fasta_file** (*str*) – Fasta file from which the first sequence will be read.
- **template_location** (*str*) – Directory where processed templates will be stored (default "./templates/").

Return type
AlignmentGenerator

Raises
ValueError – Fasta file not properly formatted

get_pdbs(*pdb_format: str = 'auto'*, *verbose: bool = True*) → None

Downloads and processes templates present in alignment.

Parameters

- **pdb_format** (*str*) – Format of PDB identifiers in alignment (default auto)
- **verbose** (*bool*) – Explain what operations are performed

Raises

- **RuntimeError** – Alignment has not been generated yet
- **ValueError** – PDB format could not be guessed

Notes

pdb_format tells the function how to parse the template identifiers in the alignment:

- auto: Automatic guess for pdb_format
- entry: Sequences are named only be their PDB identifier (i.e. 4G0N)
- entity: Sequences are named in the format PDBID_ENTITY (i.e. 4G0N_1)
- instance: Sequences are named in the format PDBID_CHAIN (i.e. 4G0N_A)

Please make sure that all templates follow one naming convention, and that there are no sequences in the alignment that violate the naming convention (except the target sequence).

During the template processing, all hetatms will be remove from the template, as well as all other chains. All chains will be renamed to "A" and the residue number will be set to 1 on the first residue. The corresponding annotations are automatically made in the alignment object.

initialize_task(*task_name: ~typing.Optional[str] = None*, *overwrite: bool = False*, *task_class: ~homelette.organization.Task = <class 'homelette.organization.Task'>*) → *Task*

Initialize a homelette Task object for model generation and evaluation.

Parameters

- **task_name** (*str*) – The name of the task to initialize. If None, initialize as models_{target}.
- **overwrite** (*bool*) – Whether to overwrite the task directory if a directory of the same name already exists (default False).
- **task_class** (Task) – The class to initialize the Task with. This makes it possible to define custom child classes of Task and construct them from this function (default Task)

Return type
Task

Raises
RuntimeError – Alignment has not been generated or templates have not been downloaded and processed.

select_templates(*templates: Iterable*) → None

Select templates from suggested templates by identifier.

Parameters
templates (*iterable*) – The selected templates as an interable.

Return type
None

Raises
RuntimeError – Alignment has not been generated yet

show_suggestion(*get_metadata: bool = False*) → Type[pandas.DataFrame]

Shows which templates have been suggested by the AlignmentGenerator, as well as some useful statistics (sequence identity, coverage).

Parameters
get_metadata (*bool*) – Retrieve additional metadata (experimental method, resolution, structure title) from the RCSB.

Returns
suggestion – DataFrame with calculated sequence identity and sequence coverage for target

Return type
pd.DataFrame

Raises
RuntimeError – Alignment has not been generated yet

See also:

Alignment.calc_identity, *Alignment.calc_coverage*

Notes

The standard output lists the templates in the alignment and shows both coverage and sequence identity to the target sequence. The templates are ordered by sequence identity.

In addition, the experimental method (Xray, NMR or Electron Microscopy), the resolution (if applicable) and the title of the template structure can be retrieved from the RCSB. Retrieving metadata from the PDB requires a working internet connecction.

class homelette.alignment.**AlignmentGenerator_hhblits**(*sequence: str*, *target: str = 'target'*, *template_location: str = './templates/'*)

Bases: *AlignmentGenerator*

Identification of templates using hhblits to search a local PDB database, generation of alignment by combining pairwise alignments of target and template together.

Parameters

- **sequence** (*str*) – Target sequence in 1 letter amino acid code.
- **target** (*str*) – The name of the target sequence (default "target"). If longer than 14 characters, will be truncated.
- **template_location** (*str*) – Directory where processed templates will be stored (default "./templates/").

Variables

- **alignment** (Alignment) – The alignment.
- **target_seq** (*str*) – The target sequence.
- **target** (*str*) – The name of the target sequence.
- **template_location** (*str*) – Directory where processed templates will be stored.
- **state** – Dictionary describing the state of the AlignmentGenerator object.

Return type
None

Notes

HHblits from the HHsuite[5] is used to query the databases. The resulting pairwise sequence alignments of template to target are combined using the target sequence as the master sequence. The resulting alignment is therefore, strictly speaking, not a proper multiple sequence alignment. However, all information from the pairwise alignments is preserved, and for homology modelling, the alignments of templates among each others do not have any influence.

[5] Sievers, F., & Higgins, D. G. (2018). Clustal Omega for making accurate alignments of many protein sequences. Protein Science, 27(1), 135–145. https://doi.org/10.1002/PRO.3290

References

get_suggestion(*database_dir: str = './databases/'*, *use_uniref: bool = False*, *evalue_cutoff: float = 0.001*, *iterations: int = 2*, *n_threads: int = 2*, *neffmax: float = 10.0*, *verbose: bool = True*) → None

Use HHblits to identify template structures and create a multiple sequence alignment by combination of pairwise alignments on target sequence.

Parameters

- **database_dir** (*str*) – The directory where the pdb70 (and the UniRef30) database are stored (default ./databases/).
- **use_uniref** (*bool*) – Use UniRef30 to create a MSA before querying the pdb70 database (default False). This leads to better results, however it takes longer and requires the UniRef30 database on your system.
- **evalue_cutoff** (*float*) – E-value cutoff for inclusion in result alignment (default 0.001)
- **iterations** (*int*) – Number of iterations when querying the pdb70 database.
- **n_threads** (*int*) – Number of threads when querying the pdb70 (or UniRef30) database (default 2).
- **neffmax** (*float*) – The neffmax value used when querying the pdb70 database (default 10.0).
- **verbose** (*bool*) – Explain which operations are performed (default True).

Return type
None

Raises
RuntimeError – Alignment has already been generated.

Notes

This function expects "hhblits" to be installed and in the path. In addition, the pdb70 database needs to be downloaded and extracted in the database_dir. The files need to be called "pdb70_*" for hhblits to correctly find the database. If UniRef30 is used to create a pre-alignment for better results, the UniRef30 database needs to be downloaded and extracted in the database_dir. The files need to be called "UniRef30_*".

For more information on neffmax, please check the hhblits documentation.

If UniRef30 is used to generate a prealignment, then hhblits will be called for one iteration with standard parameters.

classmethod from_fasta(*fasta_file: str*, *template_location: str = './templates/'*) → *AlignmentGenerator*

Generates an instance of the AlignemntGenerator with the first sequence in the fasta file.

Parameters

- **fasta_file** (*str*) – Fasta file from which the first sequence will be read.
- **template_location** (*str*) – Directory where processed templates will be stored (default "./templates/").

Return type
AlignmentGenerator

Raises
ValueError – Fasta file not properly formatted

get_pdbs(*pdb_format: str = 'auto'*, *verbose: bool = True*) → None

Downloads and processes templates present in alignment.

Parameters

- **pdb_format** (*str*) – Format of PDB identifiers in alignment (default auto)
- **verbose** (*bool*) – Explain what operations are performed

Raises

- **RuntimeError** – Alignment has not been generated yet
- **ValueError** – PDB format could not be guessed

Notes

pdb_format tells the function how to parse the template identifiers in the alignment:

- auto: Automatic guess for pdb_format
- entry: Sequences are named only be their PDB identifier (i.e. 4G0N)
- entity: Sequences are named in the format PDBID_ENTITY (i.e. 4G0N_1)
- instance: Sequences are named in the format PDBID_CHAIN (i.e. 4G0N_A)

Please make sure that all templates follow one naming convention, and that there are no sequences in the alignment that violate the naming convention (except the target sequence).

During the template processing, all hetatms will be remove from the template, as well as all other chains. All chains will be renamed to "A" and the residue number will be set to 1 on the first residue. The corresponding annotations are automatically made in the alignment object.

initialize_task(*task_name: ~typing.Optional[str] = None*, *overwrite: bool = False*, *task_class: ~homelette.organization.Task = <class 'homelette.organization.Task'>*) → *Task*

Initialize a homelette Task object for model generation and evaluation.

Parameters

- **task_name** (*str*) – The name of the task to initialize. If None, initialize as models_{target}.
- **overwrite** (*bool*) – Whether to overwrite the task directory if a directory of the same name already exists (default False).
- **task_class** (Task) – The class to initialize the Task with. This makes it possible to define custom child classes of Task and construct them from this function (default Task)

Return type
Task

Raises
RuntimeError – Alignment has not been generated or templates have not been downloaded and processed.

select_templates(*templates: Iterable*) → None

Select templates from suggested templates by identifier.

Parameters
templates (*iterable*) – The selected templates as an interable.

Return type
None

Raises
RuntimeError – Alignment has not been generated yet

show_suggestion(*get_metadata: bool = False*) → Type[pandas.DataFrame]

Shows which templates have been suggested by the AlignmentGenerator, as well as some useful statistics (sequence identity, coverage).

Parameters
get_metadata (*bool*) – Retrieve additional metadata (experimental method, resolution, structure title) from the RCSB.

Returns
suggestion – DataFrame with calculated sequence identity and sequence coverage for target

Return type
pd.DataFrame

Raises
RuntimeError – Alignment has not been generated yet

See also:

Alignment.calc_identity, *Alignment.calc_coverage*

Notes

The standard output lists the templates in the alignment and shows both coverage and sequence identity to the target sequence. The templates are ordered by sequence identity.

In addition, the experimental method (Xray, NMR or Electron Microscopy), the resolution (if applicable) and the title of the template structure can be retrieved from the RCSB. Retrieving metadata from the PDB requires a working internet connecction.

class homelette.alignment.**AlignmentGenerator_from_aln**(*alignment_file: str*, *target: str*, *template_location: str = './templates/'*, *file_format: str = 'fasta'*)

Bases: *AlignmentGenerator*

Reads an alignment from file into the AlignmentGenerator workflow.

Parameters

- **alignment_file** (*str*) – The file to read the alignment from.
- **target** (*str*) – The name of the target sequence in the alignment.
- **template_location** (*str*) – Directory where processed templates will be stored (default './templates/').
- **file_format** (*str, optional*) – The format of the alignment file. Can be 'fasta' or 'pir' (default 'fasta').

Variables

- **alignment** (Alignment) – The alignment.
- **target_seq** (*str*) – The target sequence.
- **target** (*str*) – The name of the target sequence.
- **template_location** (*str*) – Directory where processed templates will be stored.
- **state** (*dict*) – Dictionary describing the state of the AlignmentGenerator object.

Return type
None

Notes

Useful for making use of the PDB download and processing functions that come with the AlignmentGenerator classes.

`get_suggestion`()

Not implemented, since alignment is read from file on initialization.

Raises
`NotImplementedError` –

`from_fasta`(**args*, ***kwargs*)

Not implemented, since alignment is read from file on initialization.

Raises
`NotImplementedError` –

`get_pdbs`(*pdb_format: str = 'auto'*, *verbose: bool = True*) → None

Downloads and processes templates present in alignment.

Parameters

- `pdb_format` (*str*) – Format of PDB identifiers in alignment (default auto)
- `verbose` (*bool*) – Explain what operations are performed

Raises

- `RuntimeError` – Alignment has not been generated yet
- `ValueError` – PDB format could not be guessed

Notes

pdb_format tells the function how to parse the template identifiers in the alignment:

- auto: Automatic guess for pdb_format
- entry: Sequences are named only be their PDB identifier (i.e. 4G0N)
- entity: Sequences are named in the format PDBID_ENTITY (i.e. 4G0N_1)
- instance: Sequences are named in the format PDBID_CHAIN (i.e. 4G0N_A)

Please make sure that all templates follow one naming convention, and that there are no sequences in the alignment that violate the naming convention (except the target sequence).

During the template processing, all hetatms will be remove from the template, as well as all other chains. All chains will be renamed to "A" and the residue number will be set to 1 on the first residue. The corresponding annotations are automatically made in the alignment object.

`initialize_task`(*task_name: ~typing.Optional[str] = None*, *overwrite: bool = False*, *task_class: ~homelette.organization.Task = <class 'homelette.organization.Task'>*) → *Task*

Initialize a homelette Task object for model generation and evaluation.

Parameters

- `task_name` (*str*) – The name of the task to initialize. If None, initialize as models_{target}.

- **overwrite** (*bool*) – Whether to overwrite the task directory if a directory of the same name already exists (default False).
- **task_class** (Task) – The class to initialize the Task with. This makes it possible to define custom child classes of Task and construct them from this function (default Task)

Return type
Task

Raises
RuntimeError – Alignment has not been generated or templates have not been downloaded and processed.

select_templates(*templates: Iterable*) → None

Select templates from suggested templates by identifier.

Parameters
templates (*iterable*) – The selected templates as an interable.

Return type
None

Raises
RuntimeError – Alignment has not been generated yet

show_suggestion(*get_metadata: bool = False*) → Type[pandas.DataFrame]

Shows which templates have been suggested by the AlignmentGenerator, as well as some useful statistics (sequence identity, coverage).

Parameters
get_metadata (*bool*) – Retrieve additional metadata (experimental method, resolution, structure title) from the RCSB.

Returns
suggestion – DataFrame with calculated sequence identity and sequence coverage for target

Return type
pd.DataFrame

Raises
RuntimeError – Alignment has not been generated yet

See also:

Alignment.calc_identity, *Alignment.calc_coverage*

Notes

The standard output lists the templates in the alignment and shows both coverage and sequence identity to the target sequence. The templates are ordered by sequence identity.

In addition, the experimental method (Xray, NMR or Electron Microscopy), the resolution (if applicable) and the title of the template structure can be retrieved from the RCSB. Retrieving metadata from the PDB requires a working internet connecction.

homelette.alignment.**assemble_complex_aln**(**args: Type[*Alignment*]*, *names: dict*) → Type[*Alignment*]

Assemble complex alignments compatible with MODELLER from individual alignments.

Parameters

- ***args** (Alignment) – The input alignments

- **names** (*dict*) – Dictionary instructing how sequences in the different alignment objects are supposed to be arranged in the complex alignment. The keys are the names of the sequences in the output alignments. The values are iterables of the sequence names from the input alignments in the order they are supposed to appaer in the output alignment. Any value that can not be found in the alignment signals that this position in the complex alignment should be filled with gaps.

Returns
Assembled complex alignment

Return type
Alignment

Examples

```
>>> aln1 = hm.Alignment(None)
>>> aln1.sequences = {
...     'seq1_1': hm.alignment.Sequence('seq1_1', 'HELLO'),
...     'seq2_1': hm.alignment.Sequence('seq2_1', 'H---I'),
...     'seq3_1': hm.alignment.Sequence('seq3_1', '-HI--')
...     }
>>> aln2 = hm.Alignment(None)
>>> aln2.sequences = {
...     'seq2_2': hm.alignment.Sequence('seq2_2', 'KITTY'),
...     'seq1_2': hm.alignment.Sequence('seq1_2', 'WORLD')
...     }
>>> names = {'seq1': ('seq1_1', 'seq1_2'),
...          'seq2': ('seq2_1', 'seq2_2'),
...          'seq3': ('seq3_1', 'gaps')
...     }
>>> aln_assembled = hm.alignment.assemble_complex_aln(
...     aln1, aln2, names=names)
>>> aln_assembled.print_clustal()
seq1        HELLO/WORLD
seq2        H---I/KITTY
seq3        -HI--/-----
```

3.3 homelette.routines

The *homelette.routines* submodule contains classes for model generation. Routines are the building blocks that are used to generate homology models.

Currently, a number of pre-implemented routines based on *MODELLER*, *altMOD* and *ProMod3* are available. It is possible to implement custom routines for model generation and use them in the *homelette* framework.

3.3.1 Tutorials

The basics of generating homology models with pre-implemented modelling routines are presented in *Tutorial 2*. Complex modelling with *homelette* is introduced in *Tutorial 6*. Implementing custom modelling routines is discussed in *Tutorial 4*. Assembling custom pipelines is discussed in *Tutorial 7*.

3.3.2 Classes

The following standard modelling routines are implemented:

Routine_automodel_default *Routine_automodel_slow* *Routine_altmod_default* *Routine_altmod_slow* *Routine_promod3*

Modelling routines for loop modelliing:

Routine_loopmodel_default *Routine_loopmodel_slow*

Specifically for the modelling of complex structures, the following routines are implemented:

Routine_complex_automodel_default *Routine_complex_automodel_slow* *Routine_complex_altmod_default* *Routine_complex_altmod_slow*

---

class homelette.routines.**Routine_automodel_default**(*alignment: Type[*Alignment*]*, *target: str*, *templates: Iterable*, *tag: str*, *n_threads: int = 1*, *n_models: int = 1*)

Class for performing homology modelling using the automodel class from modeller with a default parameter set.

Parameters

- **alignment** (Alignment) – The alignment object that will be used for modelling
- **target** (*str*) – The identifier of the protein to model
- **templates** (*Iterable*) – The iterable containing the identifier(s) of the template(s) used for the modelling
- **tag** (*str*) – The identifier associated with a specific execution of the routine
- **n_threads** (*int*) – Number of threads used in model generation (default 1)
- **n_models** (*int*) – Number of models generated (default 1)

Variables

- **alignment** (Alignment) – The alignment object that will be used for modelling
- **target** (*str*) – The identifier of the protein to model
- **templates** (*Iterable*) – The iterable containing the identifier(s) of the template(s) used for the modelling

- **tag** (*str*) – The identifier associated with a specific execution of the routine
- **n_threads** (*int*) – Number of threads used for model generation
- **n_models** (*int*) – Number of models generated
- **routine** (*str*) – The identifier associated with a specific routine
- **models** (*list*) – List of models generated by the execution of this routine

Raises

ImportError – Unable to import dependencies

Notes

The following modelling parameters can be set when initializing this Routine object:

- n_models
- n_threads

The following modelling parameters are set for this class:

modelling parameter	value
model_class	modeller.automodel.automodel
library_schedule	modeller.automodel.autosched.normal
md_level	modeller.automodel.refine.very_fast
max_var_iterations	200
repeat_optmization	1

generate_models() → None

Generate models with the parameter set automodel_default.

Return type

None

class homelette.routines.**Routine_automodel_slow**(*alignment: Type[*Alignment*]*, *target: str*, *templates: Iterable*, *tag: str*, *n_threads: int = 1*, *n_models: int = 1*)

Class for performing homology modelling using the automodel class from modeller with a slow parameter set.

Parameters

- **alignment** (Alignment) – The alignment object that will be used for modelling
- **target** (*str*) – The identifier of the protein to model
- **templates** (*Iterable*) – The iterable containing the identifier(s) of the template(s) used for the modelling
- **tag** (*str*) – The identifier associated with a specific execution of the routine
- **n_threads** (*int*) – Number of threads used in model generation
- **n_models** (*int*) – Number of models generated

Variables

- **alignment** (Alignment) – The alignment object that will be used for modelling
- **target** (*str*) – The identifier of the protein to model

- **templates** (*Iterable*) – The iterable containing the identifier(s) of the template(s) used for the modelling
- **tag** (*str*) – The identifier associated with a specific execution of the routine
- **n_threads** (*int*) – Number of threads used for model generation
- **n_models** (*int*) – Number of models generated
- **routine** (*str*) – The identifier associated with a specific routine
- **models** (*list*) – List of models generated by the execution of this routine

Raises

ImportError – Unable to import dependencies

Notes

The following modelling parameters can be set when initializing this Routine object:

- n_models
- n_threads

The following modelling parameters are set for this class:

modelling parameter	value
model_class	modeller.automodel.automodel
library_schedule	modeller.automodel.autosched.slow
md_level	modeller.automodel.refine.very_slow
max_var_iterations	400
repeat_optmization	3

generate_models() → None

Generate models with the parameter set automodel_slow.

Return type

None

class homelette.routines.**Routine_altmod_default**(*alignment: Type[*Alignment*]*, *target: str*, *templates: Iterable*, *tag: str*, *n_threads: int = 1*, *n_models: int = 1*)

Class for performing homology modelling using the Automodel_statistical_potential class from altmod with a default parameter set.

Parameters

- **alignment** (Alignment) – The alignment object that will be used for modelling
- **target** (*str*) – The identifier of the protein to model
- **templates** (*iterable*) – The iterable containing the identifier(s) of the template(s) used for the modelling
- **tag** (*str*) – The identifier associated with a specific execution of the routine
- **n_threads** (*int*) – Number of threads used in model generation
- **n_models** (*int*) – Number of models generated

Variables

- **alignment** (*Alignment*) – The alignment object that will be used for modelling
- **target** (*str*) – The identifier of the protein to model
- **templates** (*list*) – The iterable containing the identifier(s) of the template(s) used for the modelling
- **tag** (*str*) – The identifier associated with a specific execution of the routine
- **n_threads** (*int*) – Number of threads used for model generation
- **n_models** (*int*) – Number of models generated
- **routine** (*str*) – The identifier associated with a specific routine
- **models** (*list*) – List of models generated by the execution of this routine

Raises

ImportError – Unable to import dependencies

Notes

The following modelling parameters can be set when initializing this Routine object:

- n_models
- n_threads

The following modelling parameters are set for this class:

modelling parameter	value
model_class	altmod.Automodel_statistical_potential
library_schedule	modeller.automodel.autosched.normal
md_level	modeller.automodel.refine.very_fast
max_var_iterations	200
repeat_optmization	1

Autmodel_statistical_potential uses the DOPE potential for model refinement.

generate_models() → None

Generate models with the parameter set altmod_default.

Return type

None

class homelette.routines.**Routine_altmod_slow**(*alignment: Type[*Alignment*]*, *target: str*, *templates: Iterable*, *tag: str*, *n_threads: int = 1*, *n_models: int = 1*)

Class for performing homology modelling using the Automodel_statistical_potential class from altmod with a slow parameter set.

Parameters

- **alignment** (*Alignment*) – The alignment object that will be used for modelling
- **target** (*str*) – The identifier of the protein to model
- **templates** (*iterable*) – The iterable containing the identifier(s) of the template(s) used for the modelling
- **tag** (*str*) – The identifier associated with a specific execution of the routine
- **n_threads** (*int*) – Number of threads used in model generation

- **n_models** (*int*) – Number of models generated

Variables

- **alignment** (Alignment) – The alignment object that will be used for modelling
- **target** (*str*) – The identifier of the protein to model
- **templates** (*list*) – The iterable containing the identifier(s) of the template(s) used for the modelling
- **tag** (*str*) – The identifier associated with a specific execution of the routine
- **n_threads** (*int*) – Number of threads used for model generation
- **n_models** (*int*) – Number of models generated
- **routine** (*str*) – The identifier associated with a specific routine
- **models** (*list*) – List of models generated by the execution of this routine

Raises

ImportError – Unable to import dependencies

Notes

The following modelling parameters can be set when initializing this Routine object:

- n_models
- n_threads

The following modelling parameters are set for this class:

modelling parameter	value
model_class	altmod.Automodel_statistical_potential
library_schedule	modeller.automodel.autosched.slow
md_level	modeller.automodel.refine.very_slow
max_var_iterations	400
repeat_optmization	3

Autmodel_statistical_potential uses the DOPE potential for model refinement.

generate_models() → None

Generate models with the parameter set altmod_slow.

Return type

None

class homelette.routines.**Routine_promod3**(*alignment: Type[*Alignment*]*, *target: str*, *templates: Iterable*, *tag: str*)

Class for performing homology modelling using the ProMod3 engine with default parameters.

Parameters

- **alignment** (Alignment) – The alignment object that will be used for modelling
- **target** (*str*) – The identifier of the protein to model
- **templates** (*iterable*) – The iterable containing the identifier of the template used for the modelling

Variables

- **alignment** (Alignment) – The alignment object that will be used for modelling
- **target** (*str*) – The identifier of the protein to model
- **templates** (*iterable*) – The iterable containing the identifier of the template used for the modelling
- **tag** (*str*) – The identifier associated with a specific execution of the routine
- **routine** (*str*) – The identifier associated with this specific routine: promod3
- **models** (*list*) – List of models generated by the execution of this routine

Raises

- **ImportError** – Unable to import dependencies
- **ValueError** – Number of given templates is not 1

generate_models() → None

Generate models with the ProMod3 engine with default parameters.

Return type
None

class homelette.routines.**Routine_loopmodel_default**(*alignment: Type[*Alignment*]*, *target: str*, *templates: Iterable*, *tag: str*, *loop_selections: Iterable*, *n_models: int = 1*, *n_loop_models: int = 1*)

Class for performing homology loop modelling using the loopmodel class from modeller with a default parameter set.

Parameters

- **alignment** (Alignment) – The alignment object that will be used for modelling
- **target** (*str*) – The identifier of the protein to model
- **templates** (*Iterable*) – The iterable containing the identifier(s) of the template(s) used for the modelling
- **tag** (*str*) – The identifier associated with a specific execution of the routine
- **loop_selections** (*Iterable*) – Selection(s) with should be refined with loop modelling, in modeller format (example: [['18:A', '22:A'], ['29:A', '33:A']])
- **n_models** (*int*) – Number of models generated (default 1)
- **n_loop_models** (*int*) – Number of loop models generated for each model (default 1)

Variables

- **alignment** (Alignment) – The alignment object that will be used for modelling
- **target** (*str*) – The identifier of the protein to model
- **templates** (*Iterable*) – The iterable containing the identifier(s) of the template(s) used for the modelling
- **tag** (*str*) – The identifier associated with a specific execution of the routine
- **loop_selections** (*Iterable*) – Selection(s) with should be refined with loop modelling
- **n_models** (*int*) – Number of models generated
- **n_loop_models** (*int*) – Number of loop models generated for each model

- **routine** (*str*) – The identifier associated with a specific routine
- **models** (*list*) – List of models generated by the execution of this routine

Raises
ImportError – Unable to import dependencies

Notes

The following modelling parameters can be set when initializing this Routine object:

- loop_selections
- n_models
- n_loop_models

The following modelling parameters are set for this class:

modelling parameter	value
model_class	modeller.automodel.LoopModel
library_schedule	modeller.automodel.autosched.normal
md_level	modeller.automodel.refine.very_fast
max_var_iterations	200
repeat_optmization	1
loop_library_schedule	modeller.automodel.autosched.loop
loop_md_level	modeller.automodel.refine.slow
loop_max_var_iterations	200
n_threads	1

generate_models() → None

Generate models with the parameter set loopmodel_default.

Return type
None

class homelette.routines.**Routine_loopmodel_slow**(*alignment: Type[*Alignment*]*, *target: str*, *templates: Iterable*, *tag: str*, *loop_selections: Iterable*, *n_models: int = 1*, *n_loop_models: int = 1*)

Class for performing homology loop modelling using the loopmodel class from modeller with a slow parameter set.

Parameters

- **alignment** (Alignment) – The alignment object that will be used for modelling
- **target** (*str*) – The identifier of the protein to model
- **templates** (*Iterable*) – The iterable containing the identifier(s) of the template(s) used for the modelling
- **tag** (*str*) – The identifier associated with a specific execution of the routine
- **loop_selections** (*Iterable*) – Selection(s) with should be refined with loop modelling, in modeller format (example: [[‘18:A’, ‘22:A’], [‘29:A’, ‘33:A’]])
- **n_models** (*int*) – Number of models generated (default 1)
- **n_loop_models** (*int*) – Number of loop models generated for each model (default 1)

Variables

- **alignment** (Alignment) – The alignment object that will be used for modelling
- **target** (*str*) – The identifier of the protein to model
- **templates** (*Iterable*) – The iterable containing the identifier(s) of the template(s) used for the modelling
- **tag** (*str*) – The identifier associated with a specific execution of the routine
- **loop_selections** (*Iterable*) – Selection(s) with should be refined with loop modelling
- **n_models** (*int*) – Number of models generated
- **n_loop_models** (*int*) – Number of loop models generated for each model
- **routine** (*str*) – The identifier associated with a specific routine
- **models** (*list*) – List of models generated by the execution of this routine

Raises

ImportError – Unable to import dependencies

Notes

The following modelling parameters can be set when initializing this Routine object:

- loop_selections
- n_models
- n_loop_models

The following modelling parameters are set for this class:

modelling parameter	value
model_class	modeller.automodel.LoopModel
library_schedule	modeller.automodel.autosched.slow
md_level	modeller.automodel.refine.very_slow
max_var_iterations	400
repeat_optmization	3
loop_library_schedule	modeller.automodel.autosched.slow
loop_md_level	modeller.automodel.refine.very_slow
loop_max_var_iterations	400
n_threads	1

generate_models() → None

Generate models with the parameter set loopmodel_slow.

Return type

None

class homelette.routines.**Routine_complex_automodel_default**(*alignment: Type[*Alignment*]*, *target: str*, *templates: Iterable*, *tag: str*, *n_threads: int = 1*, *n_models: int = 1*)

Class for performing homology modelling of complexes using the automodel class from modeller with a default parameter set.

Parameters

- **alignment** (Alignment) – The alignment object that will be used for modelling
- **target** (*str*) – The identifier of the protein to model
- **templates** (*Iterable*) – The iterable containing the identifier(s) of the template(s) used for the modelling
- **tag** (*str*) – The identifier associated with a specific execution of the routine
- **n_threads** (*int*) – Number of threads used in model generation (default 1)
- **n_models** (*int*) – Number of models generated (default 1)

Variables

- **alignment** (Alignment) – The alignment object that will be used for modelling
- **target** (*str*) – The identifier of the protein to model
- **templates** (*Iterable*) – The iterable containing the identifier(s) of the template(s) used for the modelling
- **tag** (*str*) – The identifier associated with a specific execution of the routine
- **n_threads** (*int*) – Number of threads used for model generation
- **n_models** (*int*) – Number of models generated
- **routine** (*str*) – The identifier associated with a specific routine
- **models** (*list*) – List of models generated by the execution of this routine

Raises

ImportError – Unable to import dependencies

Notes

The following modelling parameters can be set when initializing this Routine object:

- n_models
- n_threads

The following modelling parameters are set for this class:

modelling parameter	value
model_class	modeller.automodel.automodel
library_schedule	modeller.automodel.autosched.normal
md_level	modeller.automodel.refine.very_fast
max_var_iterations	200
repeat_optmization	1

generate_models() → None

Generate complex models with the parameter set automodel_default.

Return type

None

class homelette.routines.**Routine_complex_automodel_slow**(*alignment: Type[*Alignment*]*, *target: str*, *templates: Iterable*, *tag: str*, *n_threads: int = 1*, *n_models: int = 1*)

Class for performing homology modelling of complexes using the automodel class from modeller with a slow parameter set.

Parameters

- **alignment** (*Alignment*) – The alignment object that will be used for modelling
- **target** (*str*) – The identifier of the protein to model
- **templates** (*Iterable*) – The iterable containing the identifier(s) of the template(s) used for the modelling
- **tag** (*str*) – The identifier associated with a specific execution of the routine
- **n_threads** (*int*) – Number of threads used in model generation (default 1)
- **n_models** (*int*) – Number of models generated (default 1)

Variables

- **alignment** (*Alignment*) – The alignment object that will be used for modelling
- **target** (*str*) – The identifier of the protein to model
- **templates** (*Iterable*) – The iterable containing the identifier(s) of the template(s) used for the modelling
- **tag** (*str*) – The identifier associated with a specific execution of the routine
- **n_threads** (*int*) – Number of threads used for model generation
- **n_models** (*int*) – Number of models generated
- **routine** (*str*) – The identifier associated with a specific routine
- **models** (*list*) – List of models generated by the execution of this routine

Raises

ImportError – Unable to import dependencies

Notes

The following modelling parameters can be set when initializing this Routine object:

- n_models
- n_threads

The following modelling parameters are set for this class:

modelling parameter	value
model_class	modeller.automodel.automodel
library_schedule	modeller.automodel.autosched.slow
md_level	modeller.automodel.refine.very_slow
max_var_iterations	400
repeat_optmization	3

generate_models() → None

Generate complex models with the parameters set automodel_slow.

Return type

None

class homelette.routines.**Routine_complex_altmod_default**(*alignment: Type[*Alignment*]*, *target: str*, *templates: Iterable*, *tag: str*, *n_threads: int = 1*, *n_models: int = 1*)

Class for performing homology modelling of complexes using the Automodel_statistical_potential class from altmod with a default parameter set.

Parameters

- **alignment** (*Alignment*) – The alignment object that will be used for modelling
- **target** (*str*) – The identifier of the protein to model
- **templates** (*iterable*) – The iterable containing the identifier(s) of the template(s) used for the modelling
- **tag** (*str*) – The identifier associated with a specific execution of the routine
- **n_threads** (*int*) – Number of threads used in model generation
- **n_models** (*int*) – Number of models generated

Variables

- **alignment** (*Alignment*) – The alignment object that will be used for modelling
- **target** (*str*) – The identifier of the protein to model
- **templates** (*list*) – The iterable containing the identifier(s) of the template(s) used for the modelling
- **tag** (*str*) – The identifier associated with a specific execution of the routine
- **n_threads** (*int*) – Number of threads used for model generation
- **n_models** (*int*) – Number of models generated
- **routine** (*str*) – The identifier associated with a specific routine
- **models** (*list*) – List of models generated by the execution of this routine

Raises

ImportError – Unable to import dependencies

Notes

The following modelling parameters can be set when initializing this Routine object:

- n_models
- n_threads

The following modelling parameters are set for this class:

modelling parameter	value
model_class	altmod.Automodel_statistical_potential
library_schedule	modeller.automodel.autosched.normal
md_level	modeller.automodel.refine.very_fast
max_var_iterations	200
repeat_optmization	1

Autmodel_statistical_potential uses the DOPE potential for model refinement.

generate_models() → None

Generate complex models with the parameter set altmod_default.

Return type
None

class homelette.routines.**Routine_complex_altmod_slow**(*alignment: Type[*Alignment*]*, *target: str*, *templates: Iterable*, *tag: str*, *n_threads: int = 1*, *n_models: int = 1*)

Class for performing homology modelling of complexes using the Automodel_statistical_potential class from altmod with a slow parameter set.

Parameters

- **alignment** (Alignment) – The alignment object that will be used for modelling
- **target** (*str*) – The identifier of the protein to model
- **templates** (*iterable*) – The iterable containing the identifier(s) of the template(s) used for the modelling
- **tag** (*str*) – The identifier associated with a specific execution of the routine
- **n_threads** (*int*) – Number of threads used in model generation
- **n_models** (*int*) – Number of models generated

Variables

- **alignment** (Alignment) – The alignment object that will be used for modelling
- **target** (*str*) – The identifier of the protein to model
- **templates** (*list*) – The iterable containing the identifier(s) of the template(s) used for the modelling
- **tag** (*str*) – The identifier associated with a specific execution of the routine
- **n_threads** (*int*) – Number of threads used for model generation
- **n_models** (*int*) – Number of models generated
- **routine** (*str*) – The identifier associated with a specific routine
- **models** (*list*) – List of models generated by the execution of this routine

Raises
ImportError – Unable to import dependencies

Notes

The following modelling parameters can be set when initializing this Routine object:

- n_models
- n_threads

The following modelling parameters are set for this class:

modelling parameter	value
model_class	altmod.Automodel_statistical_potential
library_schedule	modeller.automodel.autosched.slow
md_level	modeller.automodel.refine.very_slow
max_var_iterations	400
repeat_optmization	3

Autmodel_statistical_potential uses the DOPE potential for model refinement.

generate_models() → None

Generate complex models with the parameter set altmod_slow.

Return type
None

3.4 homelette.evaluation

The *homelette.evaluation* submodule contains different classes for evaluating homology models.

It is possible to implement custom Evaluation building blocks and use them in the *homelette* framework.

3.4.1 Tutorials

Working with model evaluations in *homelette* is discussed in detail in *Tutorial 3*. Implementing custom evaluation metrics is discussed in *Tutorial 4*. Assembling custom pipelines is discussed in *Tutorial 7*.

3.4.2 Classes

The following evaluation metrics are implemented:

Evaluation_dope *Evaluation_soap_protein* *Evaluation_soap_pp* *Evaluation_qmean4* *Evaluation_qmean6* *Evaluation_qmeandisco* *Evaluation_mol_probity*

---

class homelette.evaluation.**Evaluation_dope**(*model: Type[*Model*]*, *quiet: bool = False*)

Class for evaluating a model with DOPE score.

Will dump the following entries to the model.evaluation dictionary:

- dope
- dope_z_score

Parameters

- **model** (Model) – The model object to evaluate
- **quiet** (*bool*) – If True, will perform evaluation with suppressing stdout (default False). Needs to be False for running it asynchronously, as done when running Task.evaluate_models with multple cores

Variables

- **model** (Model) – The model object to evaluate

- **output** (*dict*) – Dictionary that all outputs will be dumped into

Raises

ImportError – Unable to import dependencies

Notes

DOPE is a staticial potential for the evaluation of homology models[1]. For further information, please check the modeller documentation or the associated publication.

References

evaluate() → None

Run DOPE evaluation. Automatically called on object initialization

Return type

None

class homelette.evaluation.**Evaluation_soap_protein**(*model: Type[*Model*]*, *quiet: bool = False*)

Class for evaluating a model with the SOAP protein potential.

Will dump the following entries to the model.evaluation dictionary:

- soap_protein

Parameters

- **model** (Model) – The model object to evaluate
- **quiet** (*bool*) – If True, will perform evaluation with suppressing stdout (default False). Needs to be False for running it asynchronously, as done when running Task.evaluate_models with multple cores

Variables

- **model** (Model) – The model object to evaluate
- **output** (*dict*) – Dictionary that all outputs will be dumped into

Raises

ImportError – Unable to import dependencies

Notes

SOAP is a statistical potential for evaluating homology models[2]. For more information, please check the modeller and SOAP documentations or the associated publication.

[1] Shen, M., & Sali, A. (2006). Statistical potential for assessment and prediction of protein structures. Protein Science, 15(11), 2507–2524. https://doi.org/10.1110/ps.062416606

[2] Dong, G. Q., Fan, H., Schneidman-Duhovny, D., Webb, B., Sali, A., & Tramontano, A. (2013). Optimized atomic statistical potentials: Assessment of protein interfaces and loops. Bioinformatics, 29(24), 3158–3166. https://doi.org/10.1093/bioinformatics/btt560

References

evaluate() → None

Run SOAP protein evaluation. Automatically called on object initialization

Return type
None

class homelette.evaluation.**Evaluation_soap_pp**(*model: Type[*Model*]*, *quiet: bool = False*)

Class for evaluating a model with SOAP interaction potentials. This is used for the evaluation of models of protein complexes.

Will dump the following entries to the model.evaluation dictionary:

- soap_pp_all
- soap_pp_atom
- soap_pp_pair

Parameters

- **model** (Model) – The model object to evaluate
- **quiet** (*bool*) – If True, will perform evaluation with suppressing stdout (default False). Needs to be False for running it asynchronously, as done when running Task.evaluate_models with multple cores

Variables

- **model** (Model) – The model object to evaluate
- **output** (*dict*) – Dictionary that all outputs will be dumped into

Raises
ImportError – Unable to import dependencies

Notes

SOAP is a statistical potential for evaluating homology models[3]. For more information, please check the modeller and SOAP documentations or the associated publication.

References

evaluate() → None

Run SOAP interaction evaluation. Automatically called on object initialization

Return type
None

class homelette.evaluation.**Evaluation_qmean4**(*model: Type[*Model*]*, *quiet: bool = False*)

Class for evaluating a model with the QMEAN4 potential.

Will dump the following entries to the model.evaluation dictionary:

- qmean4
- qmean4_z_score

[3] Dong, G. Q., Fan, H., Schneidman-Duhovny, D., Webb, B., Sali, A., & Tramontano, A. (2013). Optimized atomic statistical potentials: Assessment of protein interfaces and loops. Bioinformatics, 29(24), 3158–3166. https://doi.org/10.1093/bioinformatics/btt560

Parameters

- **model** (Model) – The model object to evaluate.
- **quiet** (*bool*) – If True, will perform evaluation with suppressing stdout (default False). Needs to be False for running it asynchronously, as done when running Task.evaluate_models with multple cores

Variables

- **model** (Model) – The model object to evaluate
- **output** (*dict*) – Dictionary that all outputs will be dumped into

Raises

ImportError – Unable to import dependencies

See also:

Evaluation_qmean6, *Evaluation_qmeandisco*

Notes

QMEAN is a statistical potential for evaluating homology models[4][5].

Briefly, QMEAN is a combination of different components. Four compoenents (interaction, cbeta, packing and torsion) form the *qmean4* score.

For more information, please check the QMEAN documentation or the associated publications.

References

evaluate() → None

Run QMEAN4 protein evaluation. Automatically called on object initialization :rtype: None

class homelette.evaluation.**Evaluation_qmean6**(*model: Type[*Model*]*, *quiet: bool = False*)

Class for evaluating a model with the QMEAN6 potential.

Will dump the following entries to the model.evaluation dictionary:

- qmean6
- qmean6_disco

Requires the following valid entries in the model.info dictionary:

- accpro_file (.acc file)
- psipred_file (.horiz file)

Parameters

- **model** (Model) – The model object to evaluate.
- **quiet** (*bool*) – If True, will perform evaluation with suppressing stdout (default False). Needs to be False for running it asynchronously, as done when running Task.evaluate_models with multple cores

[4] Benkert, P., Tosatto, S. C. E., & Schomburg, D. (2008). QMEAN: A comprehensive scoring function for model quality assessment. Proteins: Structure, Function and Genetics, 71(1), 261–277. https://doi.org/10.1002/prot.21715

[5] Benkert, P., Biasini, M., & Schwede, T. (2011). Toward the estimation of the absolute quality of individual protein structure models. Bioinformatics, 27(3), 343–350. https://doi.org/10.1093/bioinformatics/btq662

Variables

- **model** (Model) – The model object to evaluate
- **output** (*dict*) – Dictionary that all outputs will be dumped into

Raises

ImportError – Unable to import dependencies

See also:

Evaluation_qmean4, *Evaluation_qmeandisco*

Notes

QMEAN is a statistical potential for evaluating homology models[6][7].

QMEAN6 is a combination of six different components (interaction, cbeta, packing, torsion, ss_agreement, acc_agreement). It is an extension to the QMEAN4 score, which additionally evaluates the agreement of the model to secondary structur predictions from PSIPRED[8] and solvent accessiblity predictions from ACCpro[9].

For more information, please check the QMEAN documentation or the associated publications.

References

evaluate() → None

Run QMEAN6 protein evaluation. Automatically called on object initialization

Return type

None

class homelette.evaluation.**Evaluation_qmeandisco**(*model: Type[*Model*]*, *quiet: bool = False*)

Class for evaluating a model with the QMEAN DisCo potential.

Will dump the following entries to the model.evaluation dictionary:

- qmean6
- qmean6_z_score
- qmean_local_scores_avg
- qmean_local_scores_err

Requires the following valid entries in the model.info dictionary:

- accpro_file (.acc file)
- psipred_file (.horiz file)
- disco_file (generated by qmean.DisCoContainer.Save)

Parameters

[6] Benkert, P., Tosatto, S. C. E., & Schomburg, D. (2008). QMEAN: A comprehensive scoring function for model quality assessment. Proteins: Structure, Function and Genetics, 71(1), 261–277. https://doi.org/10.1002/prot.21715

[7] Benkert, P., Biasini, M., & Schwede, T. (2011). Toward the estimation of the absolute quality of individual protein structure models. Bioinformatics, 27(3), 343–350. https://doi.org/10.1093/bioinformatics/btq662

[8] Jones, D. T. (1999). Protein secondary structure prediction based on position-specific scoring matrices. Journal of Molecular Biology, 292(2), 195–202. https://doi.org/10.1006/JMBI.1999.3091

[9] Magnan, C. N., & Baldi, P. (2014). SSpro/ACCpro 5: almost perfect prediction of protein secondary structure and relative solvent accessibility using profiles, machine learning and structural similarity. Bioinformatics, 30(18), 2592–2597. https://doi.org/10.1093/BIOINFORMATICS/BTU352

- **model** (Model) – The model object to evaluate.
- **quiet** (*bool*) – If True, will perform evaluation with suppressing stdout (default False). Needs to be False for running it asynchronously, as done when running Task.evaluate_models with multple cores

Variables

- **model** (Model) – The model object to evaluate
- **output** (*dict*) – Dictionary that all outputs will be dumped into

Raises

ImportError – Unable to import dependencies

See also:

Evaluation_qmean4, *Evaluation_qmean6*

Notes

QMEAN is a statistical potential for evaluating homology models[10][11].

QMEAN DisCo is an extension of QMEAN by the inclusion of homology derived DIStance COnstraints[12]. These distance contraints do not influence the six component of the QMEAN6 score (interaction, cbeta, packing, torsion, ss_agreement, acc_agreement), but only the local scores.

The distance contraints for the target have to be generated before and saved to a file.

For more information, please check the QMEAN documentation or the associated publications.

References

evaluate() → None

Run QMEAN DisCo protein evaluation. Automatically called on object initialization

Return type

None

class homelette.evaluation.**Evaluation_mol_probity**(*model: Type[*Model*]*, *quiet: bool = False*)

Class for evaluating a model with the MolProbity validation service.

Will dump the following entries to the model.evaluation dictionary:

- mp_score

Parameters

- **model** (Model) – The model object to evaluate
- **quiet** (*bool*) – If True, will perform evaluation with suppressing stdout (default False). Needs to be False for running it asynchronously, as done when running Task.evaluate_models with multple cores

Variables

[10] Benkert, P., Tosatto, S. C. E., & Schomburg, D. (2008). QMEAN: A comprehensive scoring function for model quality assessment. Proteins: Structure, Function and Genetics, 71(1), 261–277. https://doi.org/10.1002/prot.21715

[11] Benkert, P., Biasini, M., & Schwede, T. (2011). Toward the estimation of the absolute quality of individual protein structure models. Bioinformatics, 27(3), 343–350. https://doi.org/10.1093/bioinformatics/btq662

[12] Studer, G., Rempfer, C., Waterhouse, A. M., Gumienny, R., Haas, J., & Schwede, T. (2020). QMEANDisCo-distance constraints applied on model quality estimation. Bioinformatics, 36(6), 1765–1771. https://doi.org/10.1093/bioinformatics/btz828

- **model** (Model) – The model object to evaluate
- **output** (*dict*) – Dictionary that all outputs will be dumped into

Notes

Molprobity is a program that evaluates the quality of 3D structures of proteins based on structural features[13][14][15]. For more information, please check the MolProbity webpage or the associated publications.

References

evaluate() → None

Run MolProbity evaluation. Automatically called on object initialization

Return type
None

3.5 homelette.pdb_io

The *homelette.pdb_io* submodule contains an object for parsing and manipulating PDB files. There are several constructor function that can read PDB files or download them from the internet.

3.5.1 Functions and classes

Functions and classes present in *homelette.pdb_io* are listed below:

PdbObject *read_pdb()* *download_pdb()*

---

homelette.pdb_io.**read_pdb**(*file_name: str*) → *PdbObject*

Reads PDB from file.

Parameters
file_name (*str*) – PDB file name

Return type
PdbObject

[13] Davis, I. W., Leaver-Fay, A., Chen, V. B., Block, J. N., Kapral, G. J., Wang, X., Murray, L. W., Arendall, W. B., Snoeyink, J., Richardson, J. S., & Richardson, D. C. (2007). MolProbity: all-atom contacts and structure validation for proteins and nucleic acids. Nucleic Acids Research, 35(suppl_2), W375–W383. https://doi.org/10.1093/NAR/GKM216

[14] Chen, V. B., Arendall, W. B., Headd, J. J., Keedy, D. A., Immormino, R. M., Kapral, G. J., Murray, L. W., Richardson, J. S., & Richardson, D. C. (2010). MolProbity: All-atom structure validation for macromolecular crystallography. Acta Crystallographica Section D: Biological Crystallography, 66(1), 12–21. https://doi.org/10.1107/S0907444909042073

[15] Williams, C. J., Headd, J. J., Moriarty, N. W., Prisant, M. G., Videau, L. L., Deis, L. N., Verma, V., Keedy, D. A., Hintze, B. J., Chen, V. B., Jain, S., Lewis, S. M., Arendall, W. B., Snoeyink, J., Adams, P. D., Lovell, S. C., Richardson, J. S., & Richardson, D. C. (2018). MolProbity: More and better reference data for improved all-atom structure validation. Protein Science, 27(1), 293–315. https://doi.org/10.1002/pro.3330

Notes

If a PDB file with multiple MODELs is read, only the first model will be conserved.

homelette.pdb_io.**download_pdb**(*pdbid: str*) → *PdbObject*

Download PDB from the RCSB.

Parameters
pdbid (*str*) – PDB identifier

Return type
PdbObject

Notes

If a PDB file with multiple MODELs is read, only the first model will be conserved.

class homelette.pdb_io.**PdbObject**(*lines: Iterable*)

Object encapsulating functionality regarding the processing of PDB files

Parameters
lines (*Iterable*) – The lines of the PDB

Variables
lines – The lines of the PDB, filtered for ATOM and HETATM records

Return type
None

See also:

read_pdb, *download_pdb*

Notes

Please contruct instances of PdbObject using the constructor functions.

If a PDB file with multiple MODELs is read, only the first model will be conserved.

write_pdb(*file_name*) → None

Write PDB to file.

Parameters
file_name (*str*) – The name of the file to write the PDB to.

Return type
None

parse_to_pd() → pandas.DataFrame

Parses PDB to pandas dataframe.

Return type
pd.DataFrame

Notes

Information is extracted according to the PDB file specification (version 3.30) and columns are named accordingly. See https://www.wwpdb.org/documentation/file-format for more information.

get_sequence(*ignore_missing: bool = True*) → str

Retrieve the 1-letter amino acid sequence of the PDB, grouped by chain.

Parameters
ignore_missing (*bool*) – Changes behaviour with regards to unmodelled residues. If True, they will be ignored for generating the sequence (default). If False, they will be represented in the sequence with the character X.

Returns
Amino acid sequence

Return type
str

get_chains() → list

Extract all chains present in the PDB.

Return type
list

transform_extract_chain(*chain*) → *PdbObject*

Extract chain from PDB.

Parameters
chain (*str*) – The chain ID to be extracted.

Return type
PdbObject

transform_renumber_residues(*starting_res: int = 1*) → *PdbObject*

Renumber residues in PDB.

Parameters
starting_res (*int*) – Residue number to start renumbering at (default 1)

Return type
PdbObject

Notes

Missing residues in the PDB (i.e. unmodelled) will not be considered in the renumbering. If multiple chains are present in the PDB, numbering will be continued from one chain to the next one.

transform_change_chain_id(*new_chain_id*) → *PdbObject*

Replace chain ID for every entry in PDB.

Parameters
new_chain_id (*str*) – New chain ID.

Return type
PdbObject

transform_remove_hetatm() → *PdbObject*

Remove all HETATM entries from PDB.

Return type
PdbObject

transform_filter_res_name(*selection: Iterable*, *mode: str = 'out'*) → *PdbObject*

Filter PDB by residue name.

Parameters

- **selection** (*Iterable*) – For which residue names to filter
- **mode** (*str*) – Filtering mode. If mode = "out", the selection will be filtered out (default). If mode = "in", everything except the selection will be filtered out.

Return type
PdbObject

transform_filter_res_seq(*lower: int*, *upper: int*) → *PdbObject*

Filter PDB by residue number.

Parameters

- **lower** (*int*) – Lower bound of range to filter with.
- **upper** (*int*) – Upper bound of range to filter with, inclusive.

Return type
PdbObject

transform_concat(**others:* PdbObject) → *PdbObject*

Concat PDB with other PDBs.

Parameters
***others** (*'PdbObject*) – Any number of PDBs.

Return type
PdbObject

CHAPTER

FOUR

EXTENSIONS

homelette can be extended by new building blocks. This section introduces how extensions work, and where to find them.

4.1 homelette Extensions

Extensions are homology modelling building blocks (model generation, model evaluation) that are developed by users and expand the homelette interface. homelette can and should be extended by custom Routines and Evaluations. We strongly encourage users to share extensions they themselves found useful with the community.

4.1.1 Using Extensions

Extensions are placed in the extension folder in the homelette package. The extension folder on your device can be found in the following way:

```
import homelette.extension as ext
print(ext.__file__)
```

After an extension has been placed in the extension folder, it can be used as such:

```
import homelette.extension.your_extension as ext_1
```

4.1.2 Submitting Extensions

Please contact us with a Pull Request on GitHub or via Email (philipp.junk@ucdconnect.ie) if you want to share your extension! Please make sure your extension is sufficiently annotated for others to use, in particular mentioning dependencies or other requirements.

4.1.3 Existing Extensions

The following extensions have already been implemented. They should be already included in the latest version of homelette. If not, they are available from our GitHub page.

FoldX extension to homelette

Philipp Junk, 2021

This extension contains evaluation metrics based on FoldX, a force field for energy calculation and protein design (https://foldxsuite.crg.eu/)[1][2].

Usage

```
import homelette.extension.extension_foldx as extension_foldx
help(extension_foldx.Evaluation_foldx_stability)
```

This extension expects FoldX to be installed and in your path.

Functions and classes

Currently contains the following items:

Evaluation_foldx_repairmodels *Evaluation_foldx_interaction* *Evaluation_foldx_stability* *Evaluation_foldx_alascan_buildmodels* *Evaluation_foldx_alascan_interaction*

---

class homelette.extension.extension_foldx.**Evaluation_foldx_repairmodels**(*model*, *quiet=False*)

Creates a modified version of the PDB and runs RepairPDB on it

Will not dump an entry to the model.evaluation dictionary

Parameters

- **model** (Model) – The model object to evaluate
- **quiet** (*bool*) – If True, will perform evaluation with suppressing stdout (default False). Needs to be False for running it asynchronously, as done when running Task.evaluate_models with multple cores

Variables

- **model** (Model) – The model object to evaluate
- **output** (*dict*) – Dictionary that all outputs will be dumped into

Notes

Most PDBs work fine with FoldX. For a specific use case in which I was working with GTP heteroatoms, I had to rename a few atoms to make the PDB compliant with FoldX.

evaluate()

Repairs models with FoldX. Automatically called on object initialization

Return type
None

[1] Guerois, R., Nielsen, J. E., & Serrano, L. (2002). Predicting Changes in the Stability of Proteins and Protein Complexes: A Study of More Than 1000 Mutations. Journal of Molecular Biology, 320(2), 369–387. https://doi.org/10.1016/S0022-2836(02)00442-4

[2] Schymkowitz, J., Borg, J., Stricher, F., Nys, R., Rousseau, F., & Serrano, L. (2005). The FoldX web server: an online force field. Nucleic Acids Research, 33(Web Server), W382–W388. https://doi.org/10.1093/nar/gki387

class homelette.extension.extension_foldx.**Evaluation_foldx_interaction**(*model*, *quiet=False*)

Calculates interaction energy with FoldX

Requires a protein-protein complex. Expects Evaluation_foldx_repairmodels to have been performed beforehand.

Will dump the following entries to the model.evaluation dictionary:

- foldx_interaction

Parameters

- **model** (Model) – The model object to evaluate
- **quiet** (*bool*) – If True, will perform evaluation with suppressing stdout (default False). Needs to be False for running it asynchronously, as done when running Task.evaluate_models with multple cores

Variables

- **model** (Model) – The model object to evaluate
- **output** (*dict*) – Dictionary that all outputs will be dumped into

evaluate()

Calculates protein interaction energy with FoldX. Automatically called on object initialization.

Return type
None

class homelette.extension.extension_foldx.**Evaluation_foldx_stability**(*model*, *quiet=False*)

Calculate protein stability with FoldX

Expects Evaluation_foldx_repairmodels to have been performed beforehand.

Will dump the following entries to the model.evaluation dictionary:

- foldx_stability

Parameters

- **model** (Model) – The model object to evaluate
- **quiet** (*bool*) – If True, will perform evaluation with suppressing stdout (default False). Needs to be False for running it asynchronously, as done when running Task.evaluate_models with multple cores

Variables

- **model** (Model) – The model object to evaluate
- **output** (*dict*) – Dictionary that all outputs will be dumped into

evaluate()

Calculates protein stability with FoldX. Automatically called on object initialization.

Return type
None

class homelette.extension.extension_foldx.**Evaluation_foldx_alascan_buildmodels**(*model*, *quiet=False*)

Generates alanine point mutations for all positions in the given model using FoldX. Automatically called on object initialization.

Expects Evaluation_foldx_repairmodels to have been performed beforehand.

Will not dump an entry to the model.evaluation dictionary.

Parameters

- **model** (Model) – The model object to evaluate
- **quiet** (*bool*) – If True, will perform evaluation with suppressing stdout (default False). Needs to be False for running it asynchronously, as done when running Task.evaluate_models with multple cores

Variables

- **model** (Model) – The model object to evaluate
- **output** (*dict*) – Dictionary that all outputs will be dumped into

See also:

Evaluation_foldx_alascan_interaction

Notes

This Evaluation is very RAM intensive, so expect only to run 1 or 2 threads ni parallel.

evaluate()

Generates alanine point mutations for all positions in the given model. Automatically called on object initialization.

Return type
None

class homelette.extension.extension_foldx.**Evaluation_foldx_alascan_interaction**(*model*, *quiet=False*)

Calculates protein interaction energy with FoldX for all alanine point mutations generated by Evaluation_foldx_alascan_buildmodels.

Expects Evaluation_foldx_alascan_buildmodels to have been run before.

Will dump the following entry to the model.evaluation dictionary:

- **foldx_alascan: Dictionary of all interaction energies for all alanine** scan mutations.

Parameters

- **model** (Model) – The model object to evaluate
- **quiet** (*bool*) – If True, will perform evaluation with suppressing stdout (default False). Needs to be False for running it asynchronously, as done when running Task.evaluate_models with multple cores

Variables

- **model** (Model) – The model object to evaluate
- **output** (*dict*) – Dictionary that all outputs will be dumped into

See also:

Evaluation_foldx_alascan_buildmodels

evaluate()

Calculates protein interaction energy with FoldX for all alanine point mutations generated by Evaluation_foldx_alascan_buildmodels.

Return type
None

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex

PYTHON MODULE INDEX

h

INDEX

F

G

H

I

M

P

R

S

T

W