PyInKnife2 user guide

Reference publication

PyInteraph2 and PyInKnife2 to analyze networks in protein structural ensembles

Valentina Sora [1,2], Matteo Tiberti [1], Deniz Dogan [1], Shahriyar Mahdi Robbani [1], Joshua Rubin [1], Elena Papaleo [1,2]

[1] Computational Biology Laboratory, Danish Cancer Society Research Center, Copenhagen, Strandboulevarden 49, 2100, Copenhagen, Denmark

[2] Cancer Systems Biology, Section of Bioinformatics, Department of Health and Technology, Technical University of Denmark, 2800, Lyngby, Denmark

An early version of the pre-print is available on biorXiv: **https://doi.org/10.1101/2020.11.22.381616**

Introduction

PyInKnife2 is an accompanying tool for PyInteraph2. Therefore, we encourage the user to read the introduction part of the PyInteraph2 user guide before diving into this guide.

PyInKnife2 implements the approach initially proposed by Salamanca Viloria and coworkers (Salamanca Viloria et al., 2017).

The goal of PyInKnife2 is to provide a tool to assess the robustness of the PSNs generated by PyInteraph2 from protein conformational ensembles generated by molecular dynamics (MD) simulations.

Please refer to the original publication for a detailed description of the theoretical framework and the methodological choices underlying PyInKnife2 (Salamanca Viloria et al., 2017).

Furthermore, for a more in-depth explanation of the options used to generate the networks with PyInteraph2, please refer to the original PyInteraph publication (Tiberti et al., 2014) and to the PyInteraph2 pre-print (Sora and Tiberti et al., 2020) and the associated GitHub repository, which can be found at **https://github.com/ELELAB/pyinteraph2** .

In this guide, you will find:

- An in-depth description of the PyInKnife executables and their command-line options.
- Five step-by-step tutorials detailing the usage of PyInKnife2 with different types of PyInteraph2 networks: center-of-mass PSNs (cmPSNs), atomic contacts PSNs (acPSNs), networks of intra-/inter-molecular hydrophobic contacts (hbIINs), networks of intra-/inter-molecular salt bridges (sbIINs), and networks of intra-/inter-molecular hydrogen bonds (hbIINs).

The PyInKnife2 software

PyInKnife2 consists of three executables:

- `pyinknife_run` , which is responsible for running the PyInKnife2 pipeline.
- `pyinknife_aggregate` , which takes care of aggregating the raw data generated by the pipeline.
- `pyinknife_plot` , which provides utilities to visualize the aggregated data.

For information on installing PyInKnife2, please consult the dedicated installation guide at: **https://github.com/ELELAB/PyInKnife2/blob/master/INSTALL.md** .

PyInKnife2 relies on an ecosystem of YAML configuration files defining the parameters of the pipeline (for example, which networks should be generated) and options for aggregating the raw data and plotting.

Examples of such configuration files can be found inside the `config` directory within the package.

pyinknife_run

Command line

```
pyinknife_run [-h] -f TRJ -s TOP [-r REF] [-c CONFIGFILE] -d RUNDIR [-n NPROC] [-ncaa [NONCANONICAL_RESIDUES
[NONCANONICAL_RESIDUES ...]]]
```

Options

Option	Description
`-h` , `--help`	Show the help message and exit.
`-f` , `--trj`	The input trajectory.
`-s` , `--top`	The input topology.
`-r` , `--ref`	The reference structure.
`-c` , `--configfile`	The configuration file that will be used to run the pipeline. The default is `$INSTALLDIR/PyInKnife2/PyInKnife/config/run.yaml` .
`-d` , `--rundir`	The directory where the pipeline will be run. Use `'.'` to run in the current working directory.
`-n` , `--nproc`	The number of processes to start in parallel. The default is 1 process(es).
`-ncaa` , `--noncanonical-residues`	A list of the names of noncanonical residues present in the system, if any.

Description

The PyInKnife2 pipeline starts by performing N resamplings on the trajectory using a jackknife strategy, meaning that $100/N$ percent of structures in the trajectory will be excluded in each resampling. For example, if 10 resamplings are performed, 10% of the structures will be excluded from each of them. Since the structures are ordered inside the trajectory, the block of structures removed in each resampling will be contiguous.

Then, PyInKnife2 can compute different types of networks for the full-length trajectory and each resampled trajectory. In detail, PyInKnife2 supports PyInteraph2's center-of-mass PSNs (cmPSNs), atomic contacts PSNs (acPSNs), hydrophobic contacts' networks (hcINs), salt bridges' networks (sbINs), and hydrogen bonds' networks (hbINs).

Different networks can be built for different distance and occurrence cut-offs for all network types and for different modalities for salt bridges (i.e., interactions between charged groups with the same charge, between charged groups with opposite charge, or both), hydrogen bonds (i.e., main chain - main chain, main chain - side chain, and side chain - side chain hydrogen bonds), acPSNs (different I_{min} cut-offs - see the acPSN tutorial for more details), and cmPSNs (different edge correction methods - see the cmPSN tutorial for more details).

PyInKnife2 can then calculate connected components and hubs for each network produced. The comparison between the values of these two metrics in the networks built from the resamplings of the trajectory is used to assess the robustness of the network built on the full-length trajectory.

Using PyInteraph2 options in the configuration file

PyInKnife2 uses PyInteraph2 commands under the hood (`pyinteraph` to build the networks, `filter_graph` to filter them, and `graph_analysis` to analyze them), and most command-line options used in PyInteraph2 can be set in the configuration file used to run the PyInKnife2 pipeline.

However, a few exceptions apply. For instance, some options used to provide input files cannot be specified since the pipeline takes care of providing the correct inputs to the commands. The same applies to the options used to define distance cut-offs in all types of supported networks since PyInKnife2 allows the user to set ranges of cut-offs to be probed and takes care of building the corresponding networks automatically.

Here is a complete list of the options that will be ignored by PyInKnife2 if specified in the configuration file and the reasons why they are ignored.

Options for building networks

These options pertain to the `pyinteraph` command.

Option	Option description	Reason why the option is ignored
`-h` , `--help`	Show the help message and exit.	The program exits after having displayed the help message.
`-s` , `--top`	Input topology file.	`pyinknife_run` automatically passes the topology file to `pyinteraph` every time it builds a network.
`-t` , `--trj`	Input trajectory.	`pyinknife_run` automatically passes the trajectory to `pyinteraph` every time it builds a network.
`-r` , `--ref`	Input reference structure.	`pyinknife_run` automatically passes the reference structure to `pyinteraph` every time it builds a network.
`-m` , `--cmpsn`	Build a cmPSN.	`pyinknife_run` automatically determines whether the user requested the construction of cmPSNs from the configuration file.
`--cmpsn-correction`	Correction to apply to the cmPSN, if any.	The different corrections to be probed when building the cmPSNs are defined in the `pyinteraph:cmpsn:corrections` sub-section of the configuration file.
`--cmpsn-co` , `--cmpsn-cutoff`	Distance cut-off used to build the cmPSN.	The different distance cut-offs to be probed when building the cmPSNs are defined in the `pyinteraph:cmpsn:dcuts` sub-section of the configuration file.
`-a` , `--acpsn`	Build an acPSN.	`pyinknife_run` automatically determines whether the user requested the construction of acPSNs from the configuration file.
`--acpsn-imin`	Interaction strength cut-off used to build the acPSN.	The different interaction strength cut-offs to be probed when building the acPSNs are defined in the `pyinteraph:acpsn:imins` sub-section of the configuration file.
`--acpsn-co` , `--acpsn-cutoff`	Distance cut-off used to build the acPSN.	The different distance cut-offs to be probed when building the acPSNs are defined in the `pyinteraph:acpsn:dcuts` sub-section of the configuration file.
`-f` , `--hydrophobic`	Build a hcIIN.	`pyinknife_run` automatically determines whether the user requested the construction of hcIINs from the configuration file.
`--hc-co` , `--hc-cutoff`	Distance cut-off used to build the hcIIN.	The different distance cut-offs to be probed when building the hcIINs are defined in the `pyinteraph:hc:dcuts` sub-section of the configuration file.
`-b` , `--salt-bridges`	Build a sbIIN.	`pyinknife_run` automatically determines whether the user requested the construction of sbIINs from the configuration file.
`--sb-mode`	Mode used to build the sbIIN.	The different modalities to be probed when building the sbIINs are defined in the `pyinteraph:sb:modes` sub-section of the configuration file.
`--sb-co` , `--sb-cutoff`	Distance cut-off used to build the sbIIN.	The different distance cut-offs to be probed when building the sbIINs are defined in the `pyinteraph:sb:dcuts` sub-section of the configuration file.
`-y` , `--hydrogen-bonds`	Build a hbIIN.	`pyinknife_run` automatically determines whether the user requested the construction of hbIINs.
`--hb-class`	Class of hydrogen bonds considered when building the hbIIN.	The different classes of hydrogen bonds probed when building the hbIINs are defined in the `pyinteraph:hb:modes` sub-section of the configuration file.
`--hb-co` , `--hb-cutoff`	Distance cut-off used to build the hbIIN.	The different distance cut-offs to be probed when building the hbIINs are defined in the `pyinteraph:hb:dcuts` sub-section of the configuration file.

Options for filtering networks

These options pertain to the `filter_graph` command.

Option	Option description	Reason why the option is ignored
`-h` , `--help`	Show the help message and exit.	The program exits after having displayed the help message.
`-d` , `--input-dat`	Input network to be filtered.	Each network to be filtered is automatically passed to `filter_graph` by `pyinknife_run` .
`-t` , `--filter-threshold`	Threshold used to filter the network.	The different thresholds to be probed when filtering the networks are defined in the `filter_graph:pcuts` sub-section of the configuration file.

Options for analyzing networks

These options pertain to the `graph_analysis` command.

Option	Option description	Reason why the option is ignored
`-h` , `--help`	Show the help message and exit.	The program exits after having displayed the help message.
`-r` , `--reference`	Reference structure used to write the PDB file with either the node degree or the numeric ID of the connected components in the b-factor column.	`pyinknife_run` automatically passes the reference structure to `graph_analysis` when running the analysis of hubs and connected components.
`-a` , `--adj-matrix`	Input network to be analyzed.	Each network to be analyzed is automatically passed to `graph_analysis` by `pyiknife_run` .
`-u` , `--hubs`	Calculate hubs.	`pyinknife_run` automatically determines whether the user requested the calculation of hubs from the configuration file.
`-c` , `--components`	Calculate connected components.	`pyinknife_run` automatically determines whether the user requested the calculation of connected components from the configuration file.

Expected running time

Running `pyinknife_run` could be time-consuming depending on the trajectory length or the number of networks and options (distance cut-offs, filtering cut-offs, modalities, etc.) the user wants to test using the same configuration file.

In general, the calculation of acPSNs and hbIINs is the most time-consuming.

To give some guidance for new users, we have estimated the time required with different combinations of input parameters.

The tests have been carried out using the same trajectory as in the tutorials (164 residues, 2471 atoms).

Network type	Number of frames	Number of cores	Number of cut-offs tested	Number of resamplings used	Running time (minutes)
cmPSN	10000	4	4	10	8.3
cmPSN	50000	4	4	10	37
cmPSN	250000	4	4	10	184.15
sbIIN	10000	4	4	10	1.94
sbIIN	250000	4	4	10	29
hbIIN (sc-sc)	1000	4	4	10	19

pyinknife_aggregate

Command line

```
pyinknife_aggregate [-h] [-c CONFIGFILE] [-ca CONFIGFILE_AGGREGATE] -d RUNDIR -od OUTDIR [--firstccs FIRSTCCS]
```

Options

Option	Description
`-h` , `--help`	Show the help message and exit.
`-c` , `--configfile`	The configuration file that was used to run the pipeline. The default is `$INSTALLDIR/PyInKnife2/PyInKnife/config/run.yaml` .
`-ca` , `--configfile-aggregate`	The configuration file that will be used for data aggregation. The default is `$INSTALLDIR/PyInKnife2/PyInKnife/config/aggregate.yaml` .
`-d` , `--rundir`	The directory where the pipeline was run. Use `'.'` to indicate that the pipeline was run in the current working directory.
`-od` , `--outdir`	The directory where the output files will be saved.
`--firstccs`	The first # most populated connected components to be considered when aggregating the data. The default is 5.

Description

`pyinknife_aggregate` parses the output files generated by `pyinknife_run` to extract information regarding the number of hubs and the size of the most populated connected components found in the networks.

For each network type (and each combination of cut-offs, modalities, etc. used), such information for the networks built from all resampled trajectories is aggregated in a CSV file.

The aggregated data contained in these CSV files can then be plotted using `pyinknife_plot` .

pyinknife_plot

Command line

```
pyinknife_plot [-h] [-c CONFIGFILE] [-ca CONFIGFILE_AGGREGATE] [-cp CONFIGFILE_PLOT] [-p {hubs,ccs}] -d RUNDIR -od
OUTDIR
```

Options

Option	Description
`-h` , `--help`	Show the help message and exit.
`-ca` , `--configfile-aggregate`	The configuration file that was used for data aggregation. The default is `$INSTALLDIR/PyInKnife2/PyInKnife/config/aggregate.yaml` .
`-cp` , `--configfile-plot`	The configuration file that will be used for plotting. The default depends on what is defined by the `-p` , `--plot` option. All default files for plotting can be found in `$INSTALLDIR/PyInKnife2/PyInKnife/config` .
`-p` , `--plot`	What to plot. The available choices are `"hubs"` , `"ccs"` . The default is `"hubs"` .
`-d` , `--rundir`	The directory where the aggregate files are. Use `'.'` to indicate that the aggregate files are in the current working directory.
`-od` , `--outdir`	The directory where the output plots will be saved.

Description

`pyinknife_plot` takes in input the output files generated by `pyinknife_aggregate` and produces bar plots summarizing the variation observed in either the size of the most populated connected component in each resampled network (if `"ccs"` was specified in the `-p` , `--plot` option) or in the number of hubs found for each hub degree (if `"hubs"` was specified).

A case study of CypA

This guide provides a step-by-step guide to the tutorials in the `PyInKnife2/PyInKnife/tutorial` directory.

We use as a case study a 1-microsecond MD trajectory of the Cyclophilin A wild-type enzyme, which was previously published (Salamanca Viloria et al., 2017). The trajectory was resolved for periodic boundary conditions (PBC), and we generated two skipped trajectories from the full-length trajectory to run the tutorials. A step-by-step description of how we generated the input files used in the tutorials is provided in the directories numbered 1 to 4 inside the `PyInKnife2/PyInKnife/tutorial` directory.

In each tutorial's sub-directory (inside `PyInKnife2/PyInKnife/tutorial/5-pyinknife`), we provide both the inputs and the outputs of the PyInKnife2 executables, apart from the trajectory (`.xtc`) files of the resampled sub-trajectories, since they would make the tutorials too big to be uploaded on GitHub.

Finally, a `readme.txt` file is available inside each sub-directory to assist you when running the tutorials.

Center-of-mass PSN (cmPSN)

Center-of-mass PSNs (cmPSNs) are networks of residue-residue interactions where an edge is identified between two residues if the centers of mass of the residues lie within a certain distance cut-off. We first presented cmPSNs in the original PyInteraph publication (Tiberti et al., 2014) as a special case of hcIINs (hydrophobic contacts' networks) where all types of residues having a side chain are included. However, in PyInteraph2 (and, subsequently, PyInKnife2), hcIINs and cmPSNs are treated separately and have their own sets of command-line (for PyInteraph2) and configuration file (for PyInKnife2) options.

Step 1- Run the PyInKnife2 pipeline

In order to use PyInKnife2 to build only cmPSNs, `pyinknife_run` is run with a configuration file where the `run` option is set to `True` in the `pyinteraph:cmpsn` sub-section of the configuration.

We use 10 resamplings so that 10% of the structures are excluded in each of them, as suggested in the original approach (Salamanca Viloria et al. 2017). We turn on the jackknife resampling by setting the `run` option of the `resampling` sub-section of the configuration to `True`, and we set the `nsamplings` option to `10` so that 10 resamplings are performed. If we set `resampling:run` equal to `False`, `pyinknife_run` will only run on the full-length trajectory and not generate any resampled trajectory. We can choose the names of the directories that will contain the outputs for the full-length trajectory and each resampled trajectory by modifying the default names set with the `resampling:dirnames:trj` and `resampling:dirnames:subtrj` options.

We test four different distance cut-offs in the range between 4.5-6.0 Å and separated by 0.5 Å each. To set these cut-offs, we modify the `pyinteraph:cmpsn:dcuts` option.

Furthermore, we decide to build cmPSNs without applying any corrections. Therefore, we set the `pyinteraph:cmpsn:corrections` option to `["null"]`. If we want, we can add the `"rg"` correction to the list to also build cmPSNs where the identification of an interaction is corrected by the radius of gyration of the residues involved in the interaction. You can read more about the theoretical framework and the effect of this correction in the publication presenting PyInteraph2 and PyInknife2 (Sora, Tiberti al., 2020).

cmPSNs are usually built by including all residues with a side chain. However, you can customize the list of residue types included in the analysis by passing a list of residue names to the `--cmpsn-residues` option (in the `pyinteraph:cmpsn:options` sub-section of the configuration file), The names should be specified as they appear in your reference structure (a PDB file). For instance, the default list, including all standard residue types (apart from glycine since it does not have a side chain), is `["ALA", "CYS", "ASP", "GLU", "PHE", "HIS", "ILE", "LYS", "LEU", "MET", "ASN", "PRO", "GLN", "ARG", "SER", "THR", "VAL", "TRP", "TYR"]`. If you want to build a network using only hydrophobic residues, please consider building a hcIIN instead. **This section** presents a tutorial on how to do it using PyInKnife2.

We can also define the name of the output file containing the cmPSN as a matrix using the `--cmpsn-graph` option in the `pyinteraph:cmpsn:options` sub-section of the configuration file. In the same sub-section, modifying the `--cmpsn-csv` option allows us to change the name of the output file storing the interactions found in the cmPSN as a table.

Furthermore, we can customize the names of the output files produced by the `filter_graph` and `graph_analysis` commands (used to filter and analyze the networks) by changing the arguments provided to the `--output-dat` option (in the `filter_graph:options` sub-section of the configuration), to the `--hubs-pdb` option (in the `graph_analysis:hubs:options` sub-section), and to the `--components-pdb` option (in the `graph_analysis:ccs:options` sub-section). The names of the log files capturing the standard output of the `pyinteraph`, `filter_graph`, and `graph_analysis` commands can be changed with the `out` option in the `pyinteraph`, `filter_graph`, `graph_analysis:hubs`, and `graph_analysis:ccs` sub-sections of the configuration, respectively.

The minimum number of edges a node must have to be considered a hub can be changed by modifying the value provided to the `--hubs-cutoff` option in the `graph_analysis:hubs:options` sub-section.

We recommend using at least 4 cores to run the process (one for each distance cut-off scrutinized). If you cannot do this, a suggestion is to change the configuration file so that you test the tutorial with only one or two distance cut-off values.

Suppose we start from the tutorial directory `PyInKnife2/PyInKnife/tutorial/5-pyinknife`. To start the analysis, we first enter the corresponding directory, `cmPSN`:

```
cd cmPSN
```

To be able to run this part of the tutorial, you need the `run.yaml` configuration file, the input trajectory, the input topology file, and the reference structure. In our case, we take the `traj_prot_dt100.xtc` trajectory in the `../../3-filter_traj` directory as the input trajectory and the `first_structure.pdb` structure in the `../../4-extract_first_structure` directory as the input topology file and reference structure.

We can launch `pyinknife_run` using the following command:

```
pyinknife_run -f ../../3-filter_traj/traj_prot_dt100.xtc -s ../../4-extract_first_structure/first_structure.pdb -r
../../4-extract_first_structure/first_structure.pdb -c run.yaml -d . -n 4
```

We use the `-d .` option to tell PyInKnife2 that all the outputs should be generated in the current working directory, and the `-n 4` option to specify the number of cores that the run should use.

At the end of the run, you should have ten `resampling*` directories (numbered 0 to 9, i.e., `resampling0`, `resampling1`, etc.) and one `trjfull` directory in your working directory. The `resampling*` directories contain the results for each resampled sub-trajectory, while the `trjfull` directory contains the results for the full-length trajectory.

Both the `trjfull` and each of the `resampling*` directories have the following internal structure:

```
# We use the 'trjfull' directory as an example
trjfull/

  # Here, we only have the 'cmpsn' sub-directory because we only ran
  # the cmPSN analysis
  cmpsn/

    # Here, we only have the 'null' sub-directory because we only ran
    # the analysis for cmPSN without any correction. If, in the
    # configuration file, we added the 'rg' mode to the cmPSN
    # 'correction' option, we would have another 'rg' sub-directory
    # here containing the results for the cmPSN with 'rg'
    # correction
    null/

      # The sub-directory for the cmPSN built using a 4.5 Å
      # distance cut-off
      4.5/

        # The output CSV file containing all the edges found
        # in the cmPSN as a table. If the system is a protein
        # complex, additional files may be created.
        # For instance, with a system containing two protein
        # chains, the 'cmpsn_intra_A.csv', 'cmpsn_intra_B.csv',
        # and 'cmpsn_inter_A-B.csv' files would also be created,
        # containing the intra-chain edges found in chain A and
        # chain B and the inter-chain edges found between
        # chain A and chain B, respectively
        cmpsn_all.csv
        # The output DAT file containing all the edges found in
        # the cmPSN as a matrix. If the system is a protein
        # complex, additional files may be created.
        # For instance, with a system containing two protein
        # chains, the 'cmpsn-graph_intra_A.dat',
        # 'cmpsn-graph_intra_B.dat', and 'cmpsn-graph_inter_A-B.csv'
        # files would also be created, containing the
        # intra-chain edges found in chain A and chain B and the
        # inter-chain edges found between chain A and chain B,
        # respectively
        cmpsn-graph_all.dat
        # The log file with the output of the 'pyinteraph'
        # command that generated the cmPSN
        cmpsn.log

        # The sub-directory for the cmPSN filtered at 20% (only
        # edges present in 20% of the frames are kept
        # in the filtered network). Only the full cmPSN is
        # filtered, and, in case of a multi-chain protein
        # system, the sub-cmPSNs containing only the
        # intra-chain or inter-chain edges are ignored,
        # since PyInKnife2 is devised to assess the properties
        # of full protein structure networks
        20.0/

          # The output DAT file containing the edges found
          # in the filtered cmPSN.
          filtered_graph.dat
          # The log file with the output of the 'filter_graph'
          # command that generated the filtered cmPSN
          filter_graph.log

          # The sub-directory containing the results for the
          # analysis of the connected components (on the
          # filtered cmPSN)
          ccs/
```

```
              # The output of the 'graph_analysis' command
              # containing the connected components found
              # in the filtered cmPSN
              ccs.out
              # The PDB file containing the reference structure
              # with the b-factor column filled with the numeric
              # ID of the connected component each residue belongs
              # to
              ccs.pdb

            # The sub-directory containing the results for the
            # analysis of hubs (on the filtered cmPSN)
            hubs/

              # The output of the 'graph_analysis' command
              # containing the hubs found in the filtered
              # cmPSN
              hubs.out
              # The PDB file containing the reference structure
              # with the b-factor column filled with the node
              # degree of each residue
              hubs.pdb

        # The sub-directory for the cmPSN built using a 5 Å distance cut-off
        5/
        # ... the internal structure is identical to that of the sub-directory
        # for the cmPSN built using a 4.5 Å distance cut-off

        # The sub-directory for the cmPSN built using a 5.5 Å distance cut-off
        5.5/
        # ... the internal structure is identical to that of the sub-directory
        # for the cmPSN built using a 4.5 Å distance cut-off

        # The sub-directory for the cmPSN built using a 6.0 Å distance cut-off
        6.0/
        # ... the internal structure is identical to that of the sub-directory
        # for the cmPSN built using a 4.5 Å distance cut-off
```

Step 2 - Aggregate the data

Before running `pyinknife_aggregate` to aggregate the raw data for all resamplings, we create an `aggregate` directory in the current working directory where the aggregated data files will be saved:

```
mkdir aggregate
```

To run the aggregation, we need the configuration file used to run the PyInKnife2 pipeline (`run.yaml`). This file is necessary because `pyinknife_aggregate` infers the location of the output files from the options used to run the pipeline. Then, we need the configuration file containing the options to aggregate the data (`../aggregate.yaml`).

We specify the path to the directory where the aggregated data files should be saved with the `-od` option and the number of most populated connected components to consider when aggregating the data with the `--firstccs` option.

We can then launch `pyinknife_aggregate` with the following line:

```
pyinknife_aggregate -c run.yaml -ca ../aggregate.yaml -d . -od aggregate --firstccs 5
```

If the run is completed successfully, we will have a series of CSV files in our `aggregate` directory, containing the aggregated results for the analyses of hubs and connected components across the different resamplings.

```
aggregate/

  # The output CSV file containing the aggregated results
  # for the analysis of connected components for the cmPSN
  # generated with no correction, a distance cut-off of
  # 4.5 Å, and filtered with a cut-off of 20.0
```

```
    cmpsn_null_4.5_20.0_ccs.csv
    # The output CSV file containing the aggregated results
    # for the analysis of hubs for the cmSPN generated with
    # no correction, a distance cut-off of 4.5 Å, and
    # filtered with a cut-off of 20.0
    cmpsn_null_4.5_20.0_hubs.csv

    # ... the output files for the cmPSNs generated with
    # other corrections, distance cut-offs, and filtering
    # cut-offs follow the same naming conventions
```

If no hubs are found in a given network, PyInKnife2 will inform you that the corresponding aggregated data file is empty, and such files will be ignored when plotting the aggregated data.

Step 3 - Plot the aggregated data

Since we want to generate our plots in a separate directory, we create a `plots` directory inside the current working directory before running:

```
mkdir plots
```

To plot the aggregated results for the analysis of hubs across the different resamplings, we need the configuration file used for running the PyInKnife2 pipeline (`run.yaml`) and the one used to aggregate the data (`../aggregate.yaml`).

Both configuration files are needed since `pyinknife_plot` uses the options defined in them to read the content of the aggregated data files correctly.

Then, we need the configuration file defining the aesthetic of our plots (`../plot_hubs_barplot.yaml`).

We set the `-p` option to `hubs` to instruct the program that we want to plot the results of the analysis of hubs.

We can then run `pyinknife_plot` using the following command:

```
pyinknife_plot -c run.yaml -ca ../aggregate.yaml -cp ../plot_hubs_barplot.yaml -p hubs -d aggregate -od plots
```

To plot the aggregated results for the analysis of connected components and save the results to the `plots` directory, we can use a command similar to the one above, setting the `-p` option to `ccs` and using the `../plot_ccs_barplot.yaml` as the configuration file for plotting:

```
pyinknife_plot -c run.yaml -ca ../aggregate.yaml -cp ../plot_ccs_barplot.yaml -p ccs -d aggregate -od plots
```

The plots can be used to understand several network behaviors. First, we can assess the stability of the hubs and connected components values during the simulation from the height of the error bars (the higher the bar, the less stable the network).

Then, we can observe which distance cut-off is a good compromise between a network that is too connected or too sparse by evaluating the number of nodes in the connected components and the distribution of hub degrees. Indeed, having the vast majority of the nodes concentrated in one big component and tens of hubs with more than 5-6 edges usually indicates an overly connected network, while having several small components (containing very few nodes each) and almost no hubs may suggest a network that is too sparse.

Atomic contacts PSN (acPSN)

Atomic contacts PSNs (acPSNs) are networks of pairwise interactions between residues based on a normalized count of the number of pairwise contacts found between the heavy atoms of the residues' side chains. These networks were originally proposed by Kannan and Vishveshwara (Kannan and Vishveshwara, 1999), and, in PyInteraph2, we provide a Python implementation of acPSNs.

In acPSNs, an interaction is identified between two residues if their normalized count of atomic contacts (pairs of atoms within a certain distance cut-off) exceeds a certain threshold (Kannan and Vishveshwara, 1999). We refer to this threshold as I_{min} in PyInKnife2.

The count of atomic contacts is normalized based on the square root of the product of two "normalization factors" depending on the type of residues whose interaction we are evaluating. The normalization factors for the twenty standard residues are provided in the original acPSN publication (Kannan and Vishveshwara, 1999).

Henceforth, we will refer to the normalized count as the "interaction strength" between two residues.

Step 1- Run the PyInKnife2 pipeline

To run the PyInKnife2 pipeline to build and analyze acPSNs, we first go to the dedicated tutorial directory `acpSN` from the `PyInKnife2/PyInKnife/tutorial/5-pyinknife` directory:

```
cd acPSN
```

We should have a `run.yaml` file available in this directory, whose options are already set to run the analysis. Specifically, the `run` option of the `acpsn` section of the configuration is set to `True`.

When working with acPSNs, we can decide on a range of I_{min} values we want to probe when constructing the network. In this tutorial, we probe the following I_{min} cut-offs: 2.9, 3.0, 3.1, and 3.2.

We use the default normalization factors provided by PyInteraph2, which correspond to those defined in the original publication (Kannan and Vishveshwara, 1999). A file with user-defined normalization factors can be passed to PyInKnife2 by setting the `--acpsn-nf-file` option in the `pyinteraph:acpsn:options` sub-section of the configuration file.

If your protein or protein complex contains non-standard residues, the default behavior of PyInKnife2 is to assign them a normalization factor of 999.9. You can change this number to any other value by using the `--acpsn-nf-default` option in the `pyinteraph:acpsn:options` sub-section of the configuration. On the other hand, if you want to operate strictly on systems whose residues have a normalization factor defined in the provided normalization factors file, you can set the `pyinteraph:acpsn:options:--acpsn-nf-permissive` option to `False`. In this case, PyInKnife2 will throw an error upon encountering residues with no associated normalization factor.

Other customizable acPSN-specific options include, for instance, the minimum sequence distance between a pair of residues (namely, how many residues are between them in the protein sequence) for the pair to be considered when calculating their interaction strength, which is defined by the `--acpsn-proxco` option (in the `pyinteraph:acpsn:options` sub-section of the configuration). This option defaults to 1, meaning that any two residues that are separated by at least one residue in the protein sequence will be considered as possibly interacting.

Differently from cmPSNs, hcIINs, sbIINs, and hbIINs, acPSNs also offer the possibility of choosing between two types of weighting for the network edges. The weighting scheme can be set using the `--acpsn-ew` option in the `pyinteraph:acpsn:options` sub-section of the configuration. The default value is `"strength"`, meaning that the edges will be weighted according to their average interaction strength (calculated per structure in the trajectory and then averaged over the total number of structures). Still, edges can also be weighted by their `"persistence"`, meaning their occurrence in the trajectory (for each edge, the percentage of structures the edge appears in). It is worth highlighting that the `filter_graph` step of the PyInKnife2 pipeline, despite having been originally devised to filter networks on the edges' occurrence, can be used to filter acPSNs weighted on the edges' average interaction strengths. in that case, the filtering threshold should be intended as a cut-off on the interaction strength. In this tutorial, we weigh acPSNs on the edges' average interaction strength and do not perform any filtering at the `filter_graph` step (only one cut-off, `0.0`, is passed to the `pcuts` option in the `pyinteraph:acpsn` sub-section of the configuration).

PyInKnife2 also allows setting a range of distance cut-offs to test with the `dcuts` option in the `pyinteraph:acpsn` subsection of the configuration. However, we recommend using a distance cut-off of 4.5 Å when building acPSNs using the default normalization factors file, since it is the distance cut-off that was used to calculate the normalization factors (Kannan and Vishveshwara, 1999).

Furthermore, we can customize the names of the output files produced by the `filter_graph` and `graph_analysis` commands (used to filter and analyze the networks) by changing the arguments provided to the `--output-dat` option (in the `filter_graph:options` sub-section of the configuration), to the `--hubs-pdb` option (in the `graph_analysis:hubs:options` sub-section), and to the `--components-pdb` option (in the `graph_analysis:ccs:options` sub-section). The names of the log files capturing the standard output of the `pyinteraph`, `filter_graph`, and `graph_analysis` commands can be changed with the `out` option in the `pyinteraph`, `filter_graph`, `graph_analysis:hubs`, and `graph_analysis:ccs` sub-sections of the configuration, respectively.

The minimum number of edges a node must have to be considered a hub can be changed by modifying the value provided to the `--hubs-cutoff` option in the `graph_analysis:hubs:options` sub-section.

We use 10 resamplings so that 10% of the structures are excluded in each of them. We turn on the jackknife resampling by setting the `run` option of the `resampling` sub-section of the configuration to `True`, and we set the `nsamplings` option to `10` so that 10 resamplings are performed. If we set `resampling:run` equal to `False`, `pyinknife_run` will only run on the full-length trajectory and not generate any resampled trajectory. We can choose the names of the directories that will contain the outputs for the full-length trajectory and each resampled trajectory by modifying the default names set with the `resampling:dirnames:trj` and `resampling:dirnames:subtrj` options.

Since acPSN calculations can be rather slow for long trajectories, we always recommend running on as many cores as the I_{min} values tested.

We use `../../3-filter_traj/traj_prot_dt1000.xtc` as the input trajectory (shorter than the one used for cmPSN to speed up the analysis) and `../../4-extract_first_structure/first_structure.pdb` as the input topology and reference structure. We instruct `pyinknife_run` to run the pipeline and write the output files in the current working directory (`-d .`) and to use four cores (`-n 4`).

To run the PyInKnife2 pipeline to build the acPSNs, we can use the following command:

```
pyinknife_run -f ../../3-filter_traj/traj_prot_dt1000.xtc -s ../../4-extract_first_structure/first_structure.pdb -r
../../4-extract_first_structure/first_structure.pdb -c run.yaml -d . -n 4
```

If the run completes successfully, we should have as many `resampling*` as the number of resamplings specified in the configuration file (numbered 0 to 9 in our case) and a `trjfull` directory. The `resampling*` directories contain the results for each of the resampled trajectories, while the `trjfull` directory contains the results for the full-length trajectory.

The internal structure of each of these directories should look like the following:

```
# We use the 'trjfull' directory as an example
trjfull/

  # Here, we only have the 'acpsn' sub-directory because we only ran
  # the acPSN analysis
  acpsn/

    # The sub-directory for the acPSN built using an i_min
    # cut-off of 2.9
    2.9/

      # The sub-directory for the acPSN built using a 4.5 Å
      # distance cut-off
      4.5/

        # The output CSV file containing the edges found
        # in the in the acPSN as a table. If the system
        # is a protein complex, additional files may be created.
        # For instance, with a system containing two protein
        # chains, the 'acpsn_intra_A.csv', 'acpsn_intra_B.csv',
        # and 'acpsn_inter_A-B.csv' files would also be created,
        # containing the intra-chain edges found in chain A and
        # chain B and the inter-chain edges found between
        # chain A and chain B, respectively
        acpsn_all.csv

        # The output DAT file containing the edges found
        # in the acPSN as a matrix. If the system is a
        # protein complex, additional files may be created.
        # For instance, with a system containing two protein
        # chains, the 'acpsn-graph_intra_A.dat',
        # 'acpsn-graph_intra_B.dat', and 'acpsn-graph_inter_A-B.dat'
        # files would also be created, containing the
        # intra-chain edges found in chain A and chain B and
        # the inter-chain edges found between chain A and
        # chain B, respectively
        acpsn-graph_all.dat

        # The log file with the output of the 'pyinteraph'
        # command that generated the acPSN
        acpsn.log

        # The sub-directory for the acPSN filtered at 0.0
        # (unfiltered). Only the full acPSN is filtered, and,
        # in case of a multi-chain protein system, the
        # sub-acPSNs containing only the intra-chain or
        # inter-chain edges are ignored, since PyInKnife2
        # is devised to assess the properties of full
        # protein structure networks
        0.0/
```

```
            # The output DAT file containing the edges found
            # in the filtered acPSN
            filtered_graph.dat
            # The log file with the output of the 'filter_graph'
            # command that generated the filtered acPSN
            filter_graph.log

            # The sub-directory containing the results for the
            # analysis of the connected components (on the
            # filtered acPSN)
            ccs/

              # The output of the 'graph_analysis' command
              # containing the connected components found
              # in the filtered acPSN
              ccs.out
              # The PDB file containing the reference structure
              # with the b-factor column filled with the numeric
              # ID of the connected component each residue belongs
              # to
              ccs.pdb

            # The sub-directory containing the results for the
            # analysis of hubs (on the filtered acPSN)
            hubs/

              # The output of the 'graph_analysis' command
              # containing the hubs found in the filtered
              # acPSN
              hubs.out
              # The PDB file containing the reference structure
              # with the b-factor column filled with the node
              # degree of each residue
              hubs.pdb

      # The sub-directory for the acPSN built using an i_min
      # cut-off of 2.9
      2.9/
```

Step 2 - Aggregate the data

Before running the aggregation, we create an `aggregate` folder inside the current working directory to store the aggregated data files:

```
mkdir aggregate
```

To run the aggregation, we use the configuration file used to run the PyInKnife2 pipeline (`run.yaml`), from whose options the location of the raw data files and the names of the aggregated output files will be determined. Furthermore, we provide the configuration file with the options needed to aggregate the data (`../aggregate.yaml`), and we specify the directory where to write the aggregated data files with the `-od` option. Finally, we define the number of most populated connected components to consider when aggregating the data using the `--firstccs` option.

Therefore, we launch `pyinknife_aggregate` with the following command:

```
pyinknife_aggregate -c run.yaml -ca ../aggregate.yaml -d . -od aggregate --firstccs 5
```

At the end of the aggregation, we can inspect the files generated in the `aggregate` directory:

```
aggregate/

  # The output CSV file containing the aggregated results
  # for the analysis of connected components for the acPSN
  # generated with an i_min cut-off of 2.9, a distance
  # cut-off of 4.5 Å, and unfiltered (filtering cut-off
  # set to 0.0)
  acpsn_2.9_4.5_0.0_ccs.csv
  # The output CSV file containing the aggregated results
  # for the analysis of hubs for the acSPN generated with
  # an i_min cut-off of 2.9, distance cut-off of 4.5 Å,
  # and unfiltered (filtering cut-off set to 0.0)
  acpsn_2.9_4.5_0.0_hubs.csv

  # ... the output files for the acPSNs generated with
  # other i_min cut-offs, distance cut-offs, and filtering
  # cut-offs follow the same naming conventions
```

If no hubs are found in a given network, PyInKnife2 will inform you that the corresponding aggregated data file is empty, and such files will be ignored when plotting the aggregated data.

Step 3 - Plot the aggregated data

Before running the plotting step, we create a `plots` directory inside the current working directory to store the plots:

```
mkdir plots
```

First, we plot the aggregated results for the analysis of hubs across the acPSNs generated for the different resampled trajectories (for each combination of I_{min} cut-offs, distance cut-offs, and filtering cut-offs).

To do this, we need three configuration files: (i) the one used to run the PyInKnife2 pipeline (`run.yaml`), which is used to retrieve the options used to run the pipeline, (ii) the one used to aggregate the data (`../aggregate.yaml`), which contains the format specifications of the aggregate data files, and (iii) the one defining the options pertaining to the plots' aesthetics, which varies according to the type of plot generated.

We set `pyinknife_plot` 's `-p` option to `hubs` to instruct the program to plot the aggregated results of the hubs analysis.

We can then run `pyinknife_plot` using the following command:

```
pyinknife_plot -c run.yaml -ca ../aggregate.yaml -cp ../plot_hubs_barplot.yaml -p hubs -d aggregate -od plots
```

We use a similar command to plot the results of the analysis of connected components, saving, once again, the plots to the `plots` directory. However, we set the `-p` option to `ccs,` and we provide a different configuration file for the plots' aesthetics, `plot_ccs_barplot.yaml` :

```
pyinknife_plot -c run.yaml -ca ../aggregate.yaml -cp ../plot_ccs_barplot.yaml -p ccs -d aggregate -od plots
```

These plots can be used to understand: i) the stability of the hubs and connected components values during the simulation by observing the error bars; ii) which I_{min} cut-off is a good compromise between avoiding a network that is too connected or too sparse. The network density can be assessed in terms of the number of nodes in the connected components along a distribution of hub degrees spanning different degrees not too skewed towards very high degrees (i.e., nodes making more than 5-6 edges).

Network of intra-/inter-molecular hydrophobic contacts (hcIIN)

Sometimes, we may be interested in analyzing only specific types of interactions in a protein or a protein complex. In this case, we can use PyInKnife2 to investigate intra-protein (or inter-proteins) networks of hydrophobic contacts and whether these networks are stable.

Step 1 - Run the PyInKnife2 pipeline

First, we enter the `hc` directory (from inside the `PyInKnife2/PyInKnife/tutorial/5-pyinknife` directory) where the analysis will be performed:

```
cd hc
```

hcIINs are usually built by including all hydrophobic residues (alanine, valine, leucine, isoleucine, phenylalanine, proline, tryptophan, and methionine). However, you can customize the list of residue types included in the analysis by passing a list of residue names to the `--hc-residues` option (in the `pyinteraph:hc:options` sub-section of the configuration file), The names should be specified as they appear in your reference structure (a PDB file). For instance, the default list, including all standard hydrophobic residue types (apart from glycine since it does not have a side chain) is `["ALA", "VAL", "LEU", "ILE", "ILE", "PHE", "PRO", "TRP", "MET"]` .

We can also define the name of the output files containing the hcIINs as matrices using the `--hc-graph` option in the `pyinteraph:hc:options` sub-section of the configuration file. In the same sub-section, modifying the `--hc-csv` option allows us to change the name of the output files storing the interactions found in the hcIINs as tables.

Furthermore, we can customize the names of the output files produced by the `filter_graph` and `graph_analysis` commands (used to filter and analyze the networks) by changing the arguments provided to the `--output-dat` option (in the `filter_graph:options` sub-section of the configuration), to the `--hubs-pdb` option (in the `graph_analysis:hubs:options` sub-section), and to the `--components-pdb` option (in the `graph_analysis:ccs:options` sub-section). The names of the log files capturing the standard output of the `pyinteraph`, `filter_graph`, and `graph_analysis` commands can be changed with the `out` option in the `pyinteraph`, `filter_graph`, `graph_analysis:hubs`, and `graph_analysis:ccs` sub-sections of the configuration, respectively.

The minimum number of edges a node must have to be considered a hub can be changed by modifying the value provided to the `--hubs-cutoff` option in the `graph_analysis:hubs:options` sub-section.

We test four different distance cut-offs in the range between 4.5-6.0 Å and separated by 0.5 Å each. To set these cut-offs, we modify the `pyinteraph:hc:dcuts` option.

We use 10 resamplings so that 10% of the structures are excluded in each of them. We turn on the jackknife resampling by setting the `run` option of the `resampling` sub-section of the configuration to `True`, and we set the `nsamplings` option to `10` so that 10 resamplings are performed. If we set `resampling:run` equal to `False`, `pyinknife_run` will only run on the full-length trajectory and not generate any resampled trajectory. We can choose the names of the directories that will contain the outputs for the full-length trajectory and each resampled trajectory by modifying the default names set with the `resampling:dirnames:trj` and `resampling:dirnames:subtrj` options.

We recommend using at least 4 cores to run the process (one for each distance cut-off scrutinized). If you cannot do this, a suggestion is to change the configuration file so that you test the tutorial with only one or two distance cut-off values.

To be able to run this part of the tutorial, you need the `run.yaml` configuration file, the input trajectory, the input topology file, and the reference structure. In our case, we take the `traj_prot_dt100.xtc` trajectory in the `../../3-filter_traj` directory as the input trajectory and the `first_structure.pdb` structure in the `../../4-extract_first_structure` directory as the input topology file and reference structure.

We can launch `pyinknife_run` using the following command:

```
pyinknife_run -f ../../3-filter_traj/traj_prot_dt100.xtc -s ../../4-extract_first_structure/first_structure.pdb -r
../../4-extract_first_structure/first_structure.pdb -c run.yaml -d . -n 4
```

We use the `-d .` option to tell PyInKnife2 that all the outputs should be generated in the current working directory, and the `-n 4` option to specify the number of cores that the run should use.

At the end of the run, you should have ten `resampling*` directories (numbered 0 to 9, i.e., `resampling0`, `resampling1`, etc.) and one `trjfull` directory in your working directory. The `resampling*` directories contain the results for each resampled sub-trajectory, while the `trjfull` directory contains the results for the full-length trajectory.

Both the `trjfull` and each of the `resampling*` directories have the following internal structure:

```
# We use the 'trjfull' directory as an example
trjfull/

  # Here, we only have the 'cmpsn' sub-directory because we only ran
  # the hcIIN analysis
  hc/

    # The sub-directory for the hcIIN built using a 4.5 Å
    # distance cut-off
    4.5/

      # The output CSV file containing all the edges found
      # in the hcIIN as a table. If the system is a protein
      # complex, additional files may be created.
      # For instance, for a system containing two protein
      # chains, the 'hc_intra_A.csv', 'hc_intra_B.csv',
      # and 'hc_inter_A-B.csv' files would also be created,
      # containing the intra-chain edges found in chain A and
      # chain B and the inter-chain edges found between
      # chain A and chain B, respectively
      hc_all.csv
      # The output DAT file containing all the edges found in
      # the hcIIN as a matrix. If the system is a protein
      # complex, additional files may be created.
      # For instance, for a system containing two protein
```

```
        # chains, the 'hc-graph_intra_A.dat',
        # 'hc-graph_intra_B.dat', and 'hc-graph_inter_A-B.csv'
        # files would also be created, containing the
        # intra-chain edges found in chain A and chain B and the
        # inter-chain edges found between chain A and chain B,
        # respectively
        hc-graph_all.dat
        # The log file with the output of the 'pyinteraph'
        # command that generated the hcIIN
        hc.log

        # The sub-directory for the hcIIN filtered at 20% (only
        # edges present in 20% of the frames are kept
        # in the filtered network). Only the full hcIIN is
        # filtered, and, in case of a multi-chain protein
        # system, the sub-hcIINs containing only the
        # intra-chain or inter-chain edges are ignored,
        # since PyInKnife2 is devised to assess the properties
        # of full protein structure networks
        20.0/

          # The output DAT file containing the edges found
          # in the filtered hcIIN.
          filtered_graph.dat
          # The log file with the output of the 'filter_graph'
          # command that generated the filtered hcIIN
          filter_graph.log

          # The sub-directory containing the results for the
          # analysis of the connected components (on the
          # filtered hcIIN)
          ccs/

            # The output of the 'graph_analysis' command
            # containing the connected components found
            # in the filtered hcIIN
            ccs.out
            # The PDB file containing the reference structure
            # with the b-factor column filled with the numeric
            # ID of the connected component each residue belongs
            # to
            ccs.pdb

          # The sub-directory containing the results for the
          # analysis of hubs (on the filtered hcIIN)
          hubs/

            # The output of the 'graph_analysis' command
            # containing the hubs found in the filtered
            # hcIIN
            hubs.out
            # The PDB file containing the reference structure
            # with the b-factor column filled with the node
            # degree of each residue
            hubs.pdb

      # The sub-directory for the hcIIN built using a 5 Å distance cut-off
      5/
      # ... the internal structure is identical to that of the sub-directory
      # for the hcIIN built using a 4.5 Å distance cut-off

      # The sub-directory for the hcIIN built using a 5.5 Å distance cut-off
      5.5/
      # ... the internal structure is identical to that of the sub-directory
      # for the hcIIN built using a 4.5 Å distance cut-off

      # The sub-directory for the hcIIN built using a 6.0 Å distance cut-off
      6.0/
      # ... the internal structure is identical to that of the sub-directory
      # for the hcIIN built using a 4.5 Å distance cut-off
```

Step 2 - Aggregate the data

Before running `pyinknife_aggregate` to aggregate the raw data for all resamplings, we create an `aggregate` directory in the current working directory where the aggregated data files will be saved:

```
mkdir aggregate
```

To run the aggregation, we need the configuration file used to run the PyInKnife2 pipeline (`run.yaml`). This file is necessary because `pyinknife_aggregate` infers the location of the output files from the options used to run the pipeline. Then, we need the configuration file containing the options to aggregate the data (`../aggregate.yaml`).

We specify the path to the directory where the aggregated data files should be saved with the `-od` option and the number of most populated connected components to consider when aggregating the data with the `--firstccs` option.

We can then launch `pyinknife_aggregate` with the following line:

```
pyinknife_aggregate -c run.yaml -ca ../aggregate.yaml -d . -od aggregate --firstccs 5
```

If the run is completed successfully, we will have a series of CSV files in our `aggregate` directory, containing the aggregated results for the analyses of hubs and connected components across the different resamplings.

```
aggregate/

  # The output CSV file containing the aggregated results
  # for the analysis of connected components for the hcIIN
  # generated a distance cut-off of 4.5 Å and filtered with
  # a cut-off of 20.0
  hc_4.5_20.0_ccs.csv
  # The output CSV file containing the aggregated results
  # for the analysis of hubs for the hcIIN generated with
  # a distance cut-off of 4.5 Å and filtered with a cut-off
  # of 20.0
  hc_4.5_20.0_hubs.csv

  # ... the output files for the hcIINs generated with
  # other distance cut-offs and filtering cut-offs
  # follow the same naming conventions
```

If no hubs are found in a given network, PyInKnife2 will inform you that the corresponding aggregated data file is empty, and such files will be ignored when plotting the aggregated data.

Step 3 - Plot the aggregated data

Since we want to generate our plots in a separate directory, we create a `plots` directory inside the current working directory before running:

```
mkdir plots
```

To plot the aggregated results for the analysis of hubs across the different resamplings, we need the configuration file used for running the PyInKnife2 pipeline (`run.yaml`) and the one used to aggregate the data (`../aggregate.yaml`).

Both configuration files are needed since `pyinknife_plot` uses the options defined in them to read the content of the aggregated data files correctly.

Then, we need the configuration file defining the aesthetic of our plots (`../plot_hubs_barplot.yaml`).

We set the `-p` option to `hubs` to instruct the program that we want to plot the results of the analysis of hubs.

We can then run `pyinknife_plot` using the following command:

```
pyinknife_plot -c run.yaml -ca ../aggregate.yaml -cp ../plot_hubs_barplot.yaml -p hubs -d aggregate -od plots
```

To plot the aggregated results for the analysis of connected components and save the results to the `plots` directory, we can use a command similar to the one above, setting the `-p` option to `ccs` and using the `../plot_ccs_barplot.yaml` as the configuration file for plotting:

```
pyinknife_plot -c run.yaml -ca ../aggregate.yaml -cp ../plot_ccs_barplot.yaml -p ccs -d aggregate -od plots
```

The plots can be used to understand several network behaviors. First, we can assess the stability of the hubs and connected components values during the simulation from the height of the error bars (the higher the bar, the less stable the network).

Then, we can observe which distance cut-off is a good compromise between a network that is too connected or too sparse by evaluating the number of nodes in the connected components and the distribution of hub degrees. Indeed, having the vast majority of the nodes concentrated in one big component and tens of hubs with more than 5-6 edges usually indicates an overly connected network, while having several small components (containing very few nodes each) and almost no hubs may suggest a network that is too sparse.

Network of intra-/inter-molecular salt bridges (sbIIN)

We can use PyInKnife2 to investigate the intra-protein (or inter-proteins) networks of salt bridges (sbIINs) and whether those networks are stable.

Step 1- Run the PyInKnife2 pipeline

If we want to run PyInKnife2 to generate sbIINs, we need to provide a `run.yaml` file where the `run` option in the `pyinteraph:sb` sub-section is set to `True`.

We can probe different calculation modalities by overwriting the `pyinteraph:sb:modes` option and a range of distance cut-offs with the `pyinteraph:sb:dcuts` option. In our case, we use the `"different_charge"` modality (meaning that only interactions between oppositely charged side chains will be considered), and we probe four different cut-offs: 4.0 A, 4.5 A, 5.0 Å, and 5.5 Å. Other available modalities are `"same_charge"`, which identifies interactions only between groups with the same charge, and `"all"`, which allows determining interactions between both oppositely charged and similarly charged groups.

We can also can modify the chemical groups considered charged when calculating salt bridges by providing a custom file to the `--sb-cg-file` option in the `pyinteraph:sb:options` section. If the option is not defined, the charged groups file set as default in your installation of PyInteraph2 will be used. You can see which file is used as default by PyInteraph2 by running the `pyinteraph -h` command to show `pyinteraph`'s help message and look for the description of the `--sb-cg-file` option.

Moreover, you can modify the name of the output file that will contain the salt bridge network as a matrix (using the `--sb-graph` option in the `pyinteraph:sb:options` sub-section) and the one that will list the salt bridges in a table (using the `--sb-csv` option in the `pyinteraph:sb:option` sub-section).

Furthermore, we can customize the names of the output files produced by the `filter_graph` and `graph_analysis` commands (used to filter and analyze the networks) by changing the arguments provided to the `--output-dat` option (in the `filter_graph:options` sub-section of the configuration), to the `--hubs-pdb` option (in the `graph_analysis:hubs:options` sub-section), and to the `--components-pdb` option (in the `graph_analysis:ccs:options` sub-section). The names of the log files capturing the standard output of the `pyinteraph`, `filter_graph`, and `graph_analysis` commands can be changed with the `out` option in the `pyinteraph`, `filter_graph`, `graph_analysis:hubs`, and `graph_analysis:ccs` sub-sections of the configuration, respectively.

The minimum number of edges a node must have to be considered a hub can be changed by modifying the value provided to the `--hubs-cutoff` option in the `graph_analysis:hubs:options` sub-section.

We use 10 resamplings so that 10% of the structures are excluded in each of them. We turn on the jackknife resampling by setting the `run` option of the `resampling` sub-section of the configuration to `True`, and we set the `nsamplings` option to `10` so that 10 resamplings are performed. If we set `resampling:run` equal to `False`, `pyinknife_run` will only run on the full-length trajectory and not generate any resampled trajectory. We can choose the names of the directories that will contain the outputs for the full-length trajectory and each resampled trajectory by modifying the default names set with the `resampling:dirnames:trj` and `resampling:dirnames:subtrj` options.

To run the process, we recommend using at least 4 cores, if possible, namely one for each distance cut-off probed.

Suppose we start from the tutorial directory `PyInKnife2/PyInKnife/tutorial/5-pyinknife`. We enter the `sb` directory:

```
cd sb
```

Before launching `pyinknife_run`, we make sure to have the `run.yaml` configuration file, the input trajectory, the input topology file, and the reference structure. In our case, we use the `traj_prot_dt100.xtc` trajectory in the `../../3-filter_traj` directory as the input trajectory and the `first_structure.pdb` structure in the `../../4-extract_first_structure` directory as the input topology file and reference structure.

Then, we can run the pipeline using the following line:

```
pyinknife_run -f ../../3-filter_traj/traj_prot_dt100.xtc -s ../../4-extract_first_structure/first_structure.pdb -r
../../4-extract_first_structure/first_structure.pdb -c run.yaml -d . -n 4
```

The `-d .` option is used to ensure that the results of the pipeline are stored inside the current working directory, while the `-n 4` option specifies that four cores should be used.

At the end of the run, we will have ten directories containing the results for the resampled trajectories (named `resampling0`, `resampling1`, and so forth up to `resampling9`) and one directory containing the results for the full-length trajectory (named `trjfull`). The names of these directories can be customized by modifying the configuration file before running `pyinknife_run`.

All these output directories have a similar internal structure, which can be represented as follows:

```
# We use the 'trjfull' directory as an example
trjfull/

  # Here, we only have the 'sb' sub-directory because we only ran
  # the analysis of salt bridges
  sb/

    # The sub-directory for the sbIIN built using the
    # 'different_charge' modality
    different_charge/

      # The sub-directory for the sbIIN built using a
      # 4.0 Å distance cut-off
      4.0/

        # The output CSV file containing the edges found in the
        # sbIIN as a table. If the system is a protein complex,
        # additional files may be created.
        # For instance, for a system containing two protein
        # chains, the 'sb_intra_A.csv', 'sb_intra_B.csv',
        # and 'sb_inter_A-B.csv' files would also be created,
        # containing the intra-chain edges found in chain A and
        # chain B and the inter-chain edges found between
        # chain A and chain B, respectively
        sb_all.csv
        # The output DAT file containing the edges found in the
        # sbIIN as a matrix. If the system is a protein complex,
        # additional files may be created.
        # For instance, for a system containing two protein
        # chains, the 'sb-graph_intra_A.dat', 'sb-graph_intra_B.dat',
        # and 'sb-graph_inter_A-B.dat' files would also be created,
        # containing the intra-chain edges found in chain A and
        # chain B and the inter-chain edges found between
        # chain A and chain B, respectively
        sb-graph_all.dat
        # The log file with the output of the 'pyinteraph'
        # command that generated the sbIIN
        sb.log

        # The sub-directory for the sbIIN filtered at 0.0
        # (unfiltered). Only the full sbIIN is filtered and,
        # in case of a multi-chain protein system, the
        # sub-networks containing only the intra-chain or
        # inter-chain edges are ignored, since PyInKnife
        # is devised to assess the properties
        # of full protein structure networks
        0.0/

          # The output DAT file containing the interactions
          # found in the filtered sbIIN
          filtered_graph.dat
          # The log file with the output of the 'filter_graph'
          # command that generated the filtered sbIIN
          filter_graph.log
```

```
            # The sub-directory containing the results for the
            # analysis of the connected components (on the
            # filtered sbIIN)
            ccs/

              # The output of the 'graph_analysis' command
              # containing the connected components found
              # in the filtered sbIIN
              ccs.out
              # The PDB file containing the reference structure
              # with the b-factor column filled with the numeric
              # ID of the connected component each residue belongs
              # to
              ccs.pdb

            # The sub-directory containing the results for the
            # analysis of hubs (on the filtered sbIIN)
            hubs/

              # The output of the 'graph_analysis' command
              # containing the hubs found in the filtered
              # sbIIN
              hubs.out
              # The PDB file containing the reference structure
              # with the b-factor column filled with the node
              # degree of each residue
              hubs.pdb

        # The sub-directory for the sbIIN built using a
        # 4.5 Å distance cut-off
        4.5/
        # ... the internal structure is identical to that
        # of the sub-directory for the sbIIN built using
        # a 4.0 Å distance cut-off

        # The sub-directory for the sbIIN built using a
        # 5.0 Å distance cut-off
        5.0/
        # ... the internal structure is identical to that
        # of the sub-directory for the sbIIN built using
        # a 4.0 Å distance cut-off

        # The sub-directory for the sbIIN built using a
        # 5.5 Å distance cut-off
        5.5/
        # ... the internal structure is identical to that
        # of the sub-directory for the sbIIN built using
        # a 4.0 Å distance cut-off
```

Step 2 - Aggregate the data

If we want to store the aggregated data files in a separate directory to keep the analysis more structured, we must create the directory before running `pyinknife_aggregate`. In our case, we create an `aggregate` directory inside the current working directory:

```
mkdir aggregate
```

Then, we need the configuration file used for running the PyInKnife2 pipeline (`run.yaml`) and the one containing the options for aggregating the data (`../aggregate.yaml`).

We can run `pyinknife_aggregate` using the following command:

```
pyinknife_aggregate -c run.yaml -ca ../aggregate.yaml -d . -od aggregate --firstccs 5
```

We set the `aggregate` directory as the output directory using the `-od` option, and we instruct the program to look for the results of the PyInKnife2 pipeline in the current working directory (`-d .`). Furthermore, we consider only the first five connected components of each network when aggregating the data (`--firstccs 5`).

If the aggregation was successful, we would have the `aggregate` directory populated as follows:

```
aggregate/

  # The output CSV file containing the aggregated results
  # for the analysis of connected components for the sbIIN
  # generated with the 'different_charge' modality, a
  # distance cut-off of 4.0 Å, and unfiltered (filtering
  # cut-off set to 0.0)
  sb_different_charge_4.0_0.0_ccs.csv
  # The output CSV file containing the aggregated results
  # for the analysis of hubs for the sbIIN generated with
  # the 'different_charge' modality, a distance cut-off of
  # 4.0 Å, and unfiltered (filtering cut-off set to 0.0)
  sb_different_charge_4.0_0.0_hubs.csv

  # ... the output files for the sbIINs generated using
  # other modalities, distance cut-offs, and filtering
  # cut-offs follow the same naming conventions
```

If no hubs are found in a given network, PyInKnife2 will inform you that the corresponding aggregated data file is empty, and such files will be ignored when plotting the aggregated data.

Step 3 - Plot the aggregated data

Since we want to generate our plots in a separate directory, we create a `plots` directory inside the current working directory before running:

```
mkdir plots
```

To plot the aggregated results for the analysis of hubs across the different resamplings, we need the configuration file used for running the PyInKnife2 pipeline (`run.yaml`) and the one used to aggregate the data (`../aggregate.yaml`).

Both configuration files are needed since `pyinknife_plot` uses the options defined in them to read the content of the aggregated data files correctly.

Then, we need the configuration file defining the aesthetic of our plots (`../plot_hubs_barplot.yaml`).

We set the `-p` option to `hubs` to instruct the program that we want to plot the results of the analysis of hubs.

We can then run `pyinknife_plot` using the following command:

```
pyinknife_plot -c run.yaml -ca ../aggregate.yaml -cp ../plot_hubs_barplot.yaml -p hubs -d aggregate -od plots
```

To plot the aggregated results for the analysis of connected components and save the results to the `plots` directory, we can use a command similar to the one above, setting the `-p` option to `ccs` and using the `../plot_ccs_barplot.yaml` as the configuration file for plotting:

```
pyinknife_plot -c run.yaml -ca ../aggregate.yaml -cp ../plot_ccs_barplot.yaml -p ccs -d aggregate -od plots
```

The plots can be used to understand several network behaviors. First, we can assess the stability of the hubs and connected components values during the simulation from the height of the error bars (the higher the bar, the less stable the network).

Network of intra-/inter-molecular hydrogen bonds (hbIIN)

Together with salt bridges, PyInKnife2 is able to build interaction networks based on the hydrogen bonds found in a protein system (hbIINs), allowing the user to inspect the two types of electrostatic interactions separately.

Step 1- Run the PyInKnife2 pipeline

To run the PyInKnife2 pipeline to identify and analyze hbIINs, we need to set the `pyinteraph:hb:run` option in the `run.yaml` configuration file to `True`.

Then, we can choose which classes of hydrogen bonds we want to probe by enumerating them in the `pyinteraph:hb:modes` option. We can choose to build different networks including only main chain - main chain hydrogen bonds (`"mc-mc"`), main chain - side chain hydrogen bonds (`"mc-sc"`), side chain - side chain hydrogen bonds (`"sc-sc"`), or all of them (`"all"`). In this tutorial, we only consider side chain - side chain hydrogen bonds, therefore setting the option to `["sc-sc"]`. It is also possible to use a custom definition of hydrogen bonds to identify them. In this case, the option must be set to `"custom"`, and two custom chemical groups to calculate the hydrogen bonds must be defined using the `--hb-custom-group-1` and the `--hb-custom-group-2` options in the `pyinteraph:hb:options` sub-section of the configuration file.

Moreover, users can choose which atoms are considered donors, acceptors, and hydrogens mediating hydrogen bonds by providing a customized file to the `--hb-ad-file` option in the `pyinteraph:hb:options` sub-section of the configuration file. The format of this file must comply with the format of the default file used by PyInteraph2 (and, subsequently, by PyInKnife2) for the definition of donors, acceptors, and hydrogens. You can see which file is used as default (and where it is) by running the `pyinteraph -h` command from your command line, which prints out `pyinteraph`'s help message. The name and location of the file can be found in the description of the `--hb-ad-file` option in the help message.

We can also customize the minimum value for the donor-hydrogen-acceptor angle to identify a hydrogen bond by setting the `--hb-ang` (or `--hb-angle`) option in the `pyinteraph:hb:options` sub-section.

We can also change the name of the output files that will contain the hbIINs as matrices (with the `--hb-graph` option in the `pyinteraph:hb:options` sub-section) and the name of the files that will list the edges of the hbIINs in tables (with the `--hb-csv` option, also in the `pyinteraph:hb:options` sub-section).

Furthermore, we can customize the names of the output files produced by the `filter_graph` and `graph_analysis` commands (used to filter and analyze the networks) by changing the arguments provided to the `--output-dat` option (in the `filter_graph:options` sub-section of the configuration), to the `--hubs-pdb` option (in the `graph_analysis:hubs:options` sub-section), and to the `--components-pdb` option (in the `graph_analysis:ccs:options` sub-section). The names of the log files capturing the standard output of the `pyinteraph`, `filter_graph`, and `graph_analysis` commands can be changed with the `out` option in the `pyinteraph`, `filter_graph`, `graph_analysis:hubs`, and `graph_analysis:ccs` sub-sections of the configuration, respectively.

The minimum number of edges a node must have to be considered a hub can be changed by modifying the value provided to the `--hubs-cutoff` option in the `graph_analysis:hubs:options` sub-section.

We use 10 resamplings so that 10% of the structures are excluded in each of them. We turn on the jackknife resampling by setting the `run` option of the `resampling` sub-section of the configuration to `True`, and we set the `nsamplings` option to `10` so that 10 resamplings are performed. If we set `resampling:run` equal to `False`, `pyinknife_run` will only run on the full-length trajectory and not generate any resampled trajectory. We can choose the names of the directories that will contain the outputs for the full-length trajectory and each resampled trajectory by modifying the default names set with the `resampling:dirnames:trj` and `resampling:dirnames:subtrj` options.

Then, we can run the pipeline using the following line:

```
pyinknife_run -f ../../3-filter_traj/traj_prot_dt1000.xtc -s ../../4-extract_first_structure/first_structure.pdb -r
../../4-extract_first_structure/first_structure.pdb -c run.yaml -d . -n 4
```

At the end of the run, `pyinknife_run` should have generated as many directories as the number of resamplings performed containing the results for the resampled trajectories (named `resampling0`, `resampling1`, etc.), plus one directory containing the results for the full-length trajectory (named `fulltrj`).

These directories should have the following internal structure:

```
# We use the 'trjfull' directory as an example
trjfull/

  # Here, we only have the 'hb' sub-directory because we only
  # ran the analysis of hydrogen bonds
  sb/

    # The sub-directory for the hbIIN built considering only
    # side chain - side chain hydrogen bonds
    sc-sc/
```

```
# The sub-directory for the hbIIN built using a
# 3.0 Å distance cut-off
3.0/

  # The output CSV file containing the edges found in the
  # hbIIN as a table. If the system is a protein complex,
  # additional files may be created.
  # For instance, for a system containing two protein
  # chains, the 'hb_intra_A.csv', 'hb_intra_B.csv',
  # and 'hb_inter_A-B.csv' files would also be created,
  # containing the intra-chain edges found in chain A and
  # chain B and the inter-chain edges found between
  # chain A and chain B, respectively
  hb_all.csv
  # The output DAT file containing the edges found in the
  # hbIIN as a matrix. If the system is a protein complex,
  # additional files may be created.
  # For instance, for a system containing two protein
  # chains, the 'hb-graph_intra_A.dat', 'hb-graph_intra_B.dat',
  # and 'hb-graph_inter_A-B.dat' files would also be created,
  # containing the intra-chain edges found in chain A and
  # chain B and the inter-chain edges found between
  # chain A and chain B, respectively
  hb-graph_all.dat
  # The log file with the output of the 'pyinteraph'
  # command that generated the hbIIN
  hb.log

  # The sub-directory for the hbIIN filtered at 0.0
  # (unfiltered). Only the full hbIIN is filtered and,
  # in case of a multi-chain protein system, the
  # sub-networks containing only the intra-chain or
  # inter-chain edges are ignored, since PyInKnife
  # is devised to assess the properties
  # of full protein structure networks
  0.0/

    # The output DAT file containing the interactions
    # found in the filtered hbIIN
    filtered_graph.dat
    # The log file with the output of the 'filter_graph'
    # command that generated the filtered hbIIN
    filter_graph.log

    # The sub-directory containing the results for the
    # analysis of the connected components (on the
    # filtered hbIIN)
    ccs/

      # The output of the 'graph_analysis' command
      # containing the connected components found
      # in the filtered hbIIN
      ccs.out
      # The PDB file containing the reference structure
      # with the b-factor column filled with the numeric
      # ID of the connected component each residue belongs
      # to
      ccs.pdb

    # The sub-directory containing the results for the
    # analysis of hubs (on the filtered hbIIN)
    hubs/

      # The output of the 'graph_analysis' command
      # containing the hubs found in the filtered
      # hbIIN
```

```
              hubs.out
              # The PDB file containing the reference structure
              # with the b-factor column filled with the node
              # degree of each residue
              hubs.pdb

        # The sub-directory for the hbIIN built using a
        # 3.5 Å distance cut-off
        4.5/
        # ... the internal structure is identical to that
        # of the sub-directory for the hbIIN built using
        # a 3.0 Å distance cut-off
```

Step 2 - Aggregate the data

If we want to store the aggregated data files in a separate directory to keep the analysis more structured, we must create the directory before running `pyinknife_aggregate`. In our case, we create an `aggregate` directory inside the current working directory:

```
mkdir aggregate
```

To run the aggregation, we need the configuration file used for running the PyInKnife2 pipeline (`run.yaml`) and the one containing the options for aggregating the data (`../aggregate.yaml`).

Then, we can run `pyinknife_aggregate` using the following command:

```
pyinknife_aggregate -c run.yaml -ca ../aggregate.yaml -d . -od aggregate --firstccs 5
```

We set the `aggregate` directory as the output directory using the `-od` option, and we instruct the program to look for the results of the PyInKnife2 pipeline in the current working directory (`-d .`). Furthermore, we consider only the first five connected components of each network when aggregating the data (`--firstccs 5`).

If the aggregation was successful, we would have the `aggregate` directory populated as follows:

```
aggregate/

  # The output CSV file containing the aggregated results
  # for the analysis of connected components for the hbIIN
  # generated considering only side chain - side chain
  # hydrogen bonds, using a distance cut-off of 3.0 Å,
  # and unfiltered (filtering cut-off set to 0.0)
  hb_sc-sc_3.0_0.0_ccs.csv
  # The output CSV file containing the aggregated results
  # considering only side chain - side chain hydrogen bonds
  # using a distance cut-off of 3.0 Å, and unfiltered
  # (filtering cut-off set to 0.0)
  hc_sc-sc_3.0_0.0_hubs.csv

  # ... the output files for the hbIINs generated considering
  # other classes of hydrogen bonds and using other distance
  # cut-offs and filtering cut-offs follow the same naming
  # conventions
```

If no hubs are found in a given network, PyInKnife2 will inform you that the corresponding aggregated data file is empty, and such files will be ignored when plotting the aggregated data.

Step 3 - Plot the aggregated data

First, we create a `plots` directory inside the current working directory to store the plots:

```
mkdir plots
```

To plot the aggregated results for the analysis of hubs across the different resamplings, we need the configuration file used for running the PyInKnife2 pipeline (`run.yaml`) and the one used to aggregate the data (`../aggregate.yaml`). Both configuration files are needed since `pyinknife_plot` uses the options defined in them to read the content of the aggregated data files correctly.

Then, we need the configuration file defining the aesthetic of our plots (`../plot_hubs_barplot.yaml`).

We set the `-p` option to `hubs` to instruct the program that we want to plot the results of the analysis of hubs.

We can then run `pyinknife_plot` using the following command:

```
pyinknife_plot -c run.yaml -ca ../aggregate.yaml -cp ../plot_hubs_barplot.yaml -p hubs -d aggregate -od plots
```

To plot the aggregated results for the analysis of connected components and save the results to the `plots` directory, we can use a command similar to the one above, setting the `-p` option to `ccs` and using the `../plot_ccs_barplot.yaml` as the configuration file for plotting:

```
pyinknife_plot -c run.yaml -ca ../aggregate.yaml -cp ../plot_ccs_barplot.yaml -p ccs -d aggregate -od plots
```

The plots can be used to understand several network behaviors. First, we can assess the stability of the hubs and connected components values during the simulation from the height of the error bars (the higher the bar, the less stable the network).

References

(Kannan and Vishveshwara, 1999) Kannan, N., and S. Vishveshwara. "Identification of side-chain clusters in protein structures by a graph spectral method." *Journal of molecular biology* 292.2 (1999): 441-464.

(Salamanca Viloria et al., 2017) Salamanca Viloria, Juan, et al. "An optimal distance cut-off for contact-based Protein Structure Networks using side-chain centers of mass." Scientific reports 7.1 (2017): 1-11.

(Sora, Tiberti al., 2020) Sora, Valentina, Tiberti, Matteo, et al. "PyInteraph2 and PyInKnife2 to analyze networks in protein structural ensembles." bioRxiv (2020).

(Tiberti et al., 2014) Tiberti, Matteo, et al. "PyInteraph: a framework for the analysis of interaction networks in structural ensembles of proteins." Journal of chemical information and modeling 54.5 (2014): 1537-1551.