provis

Release 0.0.2

Kristof Czirjak

Jan 31, 2022

GETTING STARTED:

Provis is a protein visualization library based on python. As the name suggests it is used to visualize proteins from their .pdb data formats (from the 3D coordinates). One can visualize the protein as a stick and point model or as a surface, and many different options can be specified to show the desired properties.

The library and this documentation were created by Kristof Czirjak as his Bachelor's Thesis at ETH Zurich.

You can find installation instructions under *Setting up Provis*, a tutorial on *How to use Provis*, as well as the documentation of the source code.

What **provis** is capable of:

The following video shows the hydrophobic trajectory of a given protein.

CHAPTER

ONE

REQUIREMENTS FOR PROVIS

Provis is based on the idea of not to reinvent the wheel, so quite a few third party packages and binaries are required to run it.

1.1 Binaries

Binaries are 3rd party, ready to use programs, that are provided to the user as is. Here is a list of the required external programs and installation instructions.

1.1.1 OpenBabel

Easiest to install on Linux is by calling:

```
sudo apt install openbabel
```

Alterantively: http://openbabel.org/wiki/Main_Page

OpenBabel is needed to create the *mol.2* files. These files store the bond information.

1.1.2 PDB2PQR

Download from: https://www.poissonboltzmann.org/

Pdb2pqr is required for the surface feature plotting. If creates the *.pqr* file needed for the feature information calculation.

1.1.3 MSMS

MSMS is optional but highly suggested. It is used to compute the surface, but a native method for the surface computation also exists in **provis** (and while it is fast, it is chemically less accurate).

Download MSMS form: https://ccsb.scripps.edu/mgltools/downloads/

This tutorial might help: http://biskit.pasteur.fr/install/applications/deprecated/msms

1.2 Pip

If provis was downloaded via pip (and not from the github) then all of the following packages should be installed.

If not, then run the following command in the root directory of provis:

```
python3 setup.py develop
```

This should pip install everything, including the provis package.

Here is a list of provis' pip dependencies:

- BioPython
- Trimesh
- PyVista
- Biopandas
- Torch
- Pyvtk
- Open3d
- rTree
- Panel
- Imageio-ffmpeg

CHAPTER

TWO

DOWNLOAD PROVIS

Provis can be downloaded by pip or from github.

Pip:

```
pip install provis
```

Github: https://github.com/czirjakkethz/provis

If provis was downoaded from the git repository then the following command has to be run: (run it from the root directory)

```
python3 setup.py develop
```

This will set up provis as a python library on your machine, but also download all the python (pip) dependencies.

CHAPTER

THREE

SETTING UP PROVIS

Some file paths are hard coded in provis in order to maintain an uncluttered and organized directory structure.

However, this also means, that for provis to work this specific directory structure has to exist.

3.1 Directory structure

```
provis
    ├── data
    │   ├── data/
    │   ├── pdb/
    │   ├── img/
    │   ├── meshes/
    │   └── tmp/
    └── binaries
        ├── apbs
        ├── msms
        └── pdb2pqr
            └── pdb2pqr
```

3.1.1 Easy option

Clone provis from github and simply use this git directory (provis) as the base directory.

3.1.2 More versitile option

A data/ and (potentially) a binaries/ directory within the root directory will have to be created.

> – If the environment variables for the binaries are set then the binaries/ directory is not needed (as provis will then use these variables to find the binaries). Otherwise the binaries from the Requirements *Requirements for Provis* section will all have to be copied into the binaries/ directory. –

Subdirectiories of the data/ directory:

```
data
├── data
├── pdb
├── img
├── meshes
└── tmp
```

The pdb directory is the location to store the pdb files. If a *.pdb* file is stored here then it is enough to pass the pdb id (filename without extension) to provis. Otherwise the full path to the *.pdb* file needs to be passed.

The img directory stores all the screenshots of the outputted plots.

The tmp directory stores all temporary files created by provis, such as the *.face* and *.vert* files of MSMS or the *.mol2* files needed for the bonds.

3.2 Binaries

After installing the binaries either set the environment variables to specify the path of their location or manually move the binaries to where provis can find them.

3.2.1 Binaries directory

If you are running Ubuntu (20.04.3 LTS) and installed provis by cloning the github repository then you are all set.

Otherwise move the binary files to the 'binaries' directory as explained below.

Subdirectiories and executables of the binaries/ directory:

```
binaries
├── apbs
├── msms
└── pdb2pqr
    └── pdb2pqr
```

apbs and **msms** are executables and *pdb2pqr* is the directory downloaded from the official website, containing the **pdb2pqr** binary.

3.2.2 Setting environment variables

Alternatively the environment variables can be set to point to the binary files.

Set the MSMS_BIN, PDB2PQR_BIN and APBS_BIN variables to the full path to their appropriate binary files.

Example:

```
export MSMS_BIN='/home/username/Downloads/msms'
```

CHAPTER

FOUR

PROVIS

This section contains the documentation for the actual source code of **provis**.

You can find a description for every class and every method for all of the *.py* files in **provis**.

4.1 Source Code

All the important classes of **provis** are defined here.

4.1.1 provis.src.plotting package

The plotting package includes the four classes that handle plotting. (While the **Structure** and **Surface** classes still work, all of their functions also exist in the **Plotter**.)

To plot a protein, one can simply call the plotting member functions of these classes (after initialization).

provis.src.plotting.dynamic_plotter module

class provis.src.plotting.dynamic_plotter.**DynamicPlotter**(*prot:* provis.src.processing.protein.Protein, *msms=True*, *notebook=False*, *plot_solvent=False*)

Bases: `object`

The DynamicPlotter class, similarly to the Plotter class, encapsulates every other class and creates a user friendly way to plot your desired dynamic structure of a protein molecules.

While the class is built similarly to the Plotter class it does not use the Plotter class itself. This is due to the fact that the Plotter class is a class made to plot a static molecules.

plot_atoms(*box=0*, *res=0*, *outname=None*, *camera=None*)

Plot the atoms as spheres. Each atom has a radius proportianal to its calculated atomic radius.

Consult https://en.wikipedia.org/wiki/CPK_coloring for the coloring.

Parameters:

box: bool, optional ptional - If True bounding box also visualized, default: 0.

res: Residue, optional Residues passed in 'res' will be plotted with a bounding box around them. Defaults to None.

outname: string, optional Save image of plot to specified filename. Will appear in data/img directory. Defaults to data/img/{pdb_id}_atoms.png.

camera: pyvista.Camera, optional Pass a Pyvista Camera https://docs.pyvista.org/api/core/camera.html to manually set the camera position. If nothing/None is passed then the camera position will be set to [0, 4 * "max distance from the center", 0] (see: https://pro-vis.readthedocs.io/en/latest/tutorial.html for more detail). Default: None.

Returns:

Pyvista.Plotter window Window with interactive plot.

`plot_backbone`(*box=0*, *res=0*, *outname=0*, *camera=None*)
Plots the backbone (roughly the amide bonds) of the protein.

Parameters:

box: bool, optional If True bounding box also visualized, default: 0.

res: Residue, optional Residues passed in 'res' will be plotted with a bounding box around them. Defaults to None.

outname: string, optional Save image of plot to specified filename. Will appear in data/img directory. Defaults to data/img/{pdb_id}_backbone.png.

camera: pyvista.Camera, optional Pass a Pyvista Camera https://docs.pyvista.org/api/core/camera.html to manually set the camera position. If nothing/None is passed then the camera position will be set to [0, 4 * "max distance from the center", 0] (see: https://pro-vis.readthedocs.io/en/latest/tutorial.html for more detail). Default: None.

Returns:

Pyvista.Plotter window Window with interactive plot.

`plot_bonds`(*box=0*, *res=0*, *outname=0*, *colorful=False*, *camera=None*)
Plot only the bonds. By default all bonds will be plotted uniformly.

If the difference in bond types is of interest set the "colorful" variable to True. Coloring: - Single bonds: white - Double bonds: blue - Triple bonds: green - Amide bonds: red - Aromatic bonds: purple - Undefined/Anything else: black

Parameters:

box: bool, optional If True bounding box also visualized, default: 0.

res: Residue, optional Residues passed in 'res' will be plotted with a bounding box around them. Defaults to None.

outname: string, optional Save image of plot to specified filename. Will appear in data/img directory. Defaults to data/img/{pdb_id}_atoms.png.

colorful: bool, optional If True bonds will be plotted in a colorful manner. If False all bonds are white. Default: False

camera: pyvista.Camera, optional Pass a Pyvista Camera https://docs.pyvista.org/api/core/camera.html to manually set the camera position. If nothing/None is passed then the camera position will be set to [0, 4 * "max distance from the center", 0] (see: https://pro-vis.readthedocs.io/en/latest/tutorial.html for more detail). Default: None.

Returns:

Pyvista.Plotter window Window with interactive plot.

`plot_charge`(*box=None*, *res=None*, *outname=None*, *camera=None*)
Plot the charge features of a protein.

Parameters:

box, optional: bool, optional If True bounding box also visualized, default: 0.

res: Residue, optional Residues passed in 'res' will be plotted with a bounding box around them. Defaults to None.

outname: string, optional save image of plot to specified filename. Will appear in data/img/ directory. Default: data/img/{self._out_path}_surface.

camera: pyvista.Camera, optional Pass a Pyvista Camera https://docs.pyvista.org/api/core/camera.html to manually set the camera position. If nothing/None is passed then the camera position will be set to [0, 3 * "max distance from the center", 0] (see: https://pro-vis.readthedocs.io/en/latest/tutorial.html for more detail). Default: None.

Returns:

Pyvista.Plotter window Window with interactive plot.

`plot_hydrophob`(*box=None*, *res=None*, *outname=None*, *camera=None*)
Plot the hydrophobic features of a protein.

Parameters:

box, optional: bool, optional If True bounding box also visualized, default: 0.

res: Residue, optional Residues passed in 'res' will be plotted with a bounding box around them. Defaults to None.

outname: string, optional save image of plot to specified filename. Will appear in data/img/ directory. Default: data/img/{self._out_path}_surface.

camera: pyvista.Camera, optional Pass a Pyvista Camera https://docs.pyvista.org/api/core/camera.html to manually set the camera position. If nothing/None is passed then the camera position will be set to [0, 3 * max_distance_from_center, 0]. Default: None.

Returns:

Pyvista.Plotter window Window with interactive plot.

`plot_residues`(*box=0*, *res=0*, *outname=0*, *camera=None*)
Plot the residues as Spheres. Each sphere is the approximate size of the radius of the given residue. This plot should only be used to get a general feel for the layout of the protein.

For coloring information please visit: http://acces.ens-lyon.fr/biotic/rastop/help/colour.htm

Parameters:

box: bool, optional ptional - If True bounding box also visualized, default: 0.

res: Residue, optional Residues passed in 'res' will be plotted with a bounding box around them. Defaults to None.

outname: string, optional Save image of plot to specified filename. Will appear in data/img directory. Defaults to data/img/{pdb_id}_residues.png.

camera: pyvista.Camera, optional Pass a Pyvista Camera https://docs.pyvista.org/api/core/camera.html to manually set the camera position. If nothing/None is passed then the camera position will be set to [0, 4 * "max distance from the center", 0] (see: https://pro-vis.readthedocs.io/en/latest/tutorial.html for more detail). Default: None.

Returns:

Pyvista.Plotter window Window with interactive plot.

`plot_shape`(*box=None*, *res=None*, *outname=None*, *camera=None*)
Plot the shape features of a protein.

Parameters:

box, optional: bool, optional If True bounding box also visualized, default: 0.

res: Residue, optional Residues passed in 'res' will be plotted with a bounding box around them. Defaults to None.

outname: string, optional save image of plot to specified filename. Will appear in data/img/ directory. Default: data/img/{self._out_path}_surface.

camera: pyvista.Camera, optional Pass a Pyvista Camera https://docs.pyvista.org/api/core/camera.html to manually set the camera position. If nothing/None is passed then the camera position will be set to [0, 3 * "max distance from the center", 0] (see: https://pro-vis.readthedocs.io/en/latest/tutorial.html for more detail). Default: None.

Returns:

Pyvista.Plotter window Window with interactive plot.

`plot_stick_point`(*box=0*, *res=None*, *outname=0*, *camera=None*)

Plot stick and point model of the protein. Atoms are spheres, bonds are tubes.

Parameters:

box: bool, optional ptional - If True bounding box also visualized, default: 0.

res: Residue, optional Residues passed in 'res' will be plotted with a bounding box around them. Defaults to None.

outname: string, optional Save image of plot to specified filename. Will appear in data/img directory. Defaults to data/img/{pdb_id}_stick_point.png.

camera: pyvista.Camera, optional Pass a Pyvista Camera https://docs.pyvista.org/api/core/camera.html to manually set the camera position. If nothing/None is passed then the camera position will be set to [0, 4 * "max distance from the center", 0] (see: https://pro-vis.readthedocs.io/en/latest/tutorial.html for more detail). Default: None.

Returns:

Pyvista.Plotter window Window with interactive plot.

`plot_structure`(*box=False*, *res=None*, *outname=None*, *camera=None*, *title='Structure'*, *atoms=0*, *bonds=0*, *vw=0*, *residues=0*, *bb=0*)

Plot the dynamic atom cloud.

The code in words: Create a Plotter. Loop through the models of the Protein stored in self._protein and perform the following tasks. Clear the plotter. Add the specified meshes to the subplotter. Set the camera position. Render the plotter.

Parameters:

box: bool, optional ptional - If True bounding box also visualized, default: 0.

res: Residue, optional Residues passed in 'res' will be plotted with a bounding box around them. Defaults to None.

outname: string, optional Save image of plot to specified filename. Will appear in data/img directory. Defaults to data/img/{pdb_id}_stick_point.png.

camera: pyvista.Camera, optional Pass a Pyvista Camera https://docs.pyvista.org/api/core/camera.html to manually set the camera position. If nothing/None is passed then the camera position will be set to [0, 4 * "max distance from the center", 0] (see: https://pro-vis.readthedocs.io/en/latest/tutorial.html for more detail). Default: None.

title: str, optional Title of the plotting window. Default: "Structure".

atoms: bool, optional Plot atoms. Default: None.

bonds: int, optional ptional - Plot bond. If zero or undefined then it does not plot the bonds, if 1 it plots all bonds uniformly, if 2 it plots colorful bonds (see data_handler). Default: None.

vw: bool, optional Plot Wan-der-Waals radii instead of atomic radii.

residues: bool, optional ptional - Plot residue. Default: None.

bb: bool, optional If True backbone of protein is plotted. Default: False.

Returns:

Pyvista.Plotter window Window with interactive plot

str - only if self._notebook is True The full path to where the .mp4 video file is stored.

`plot_surface`(*feature=None*, *patch=False*, *title='Surface'*, *box=None*, *res=None*, *outname=None*, *camera=None*)

Plot the dynamic surface of the molecule.

The code in words: Create a Plotter. Loop through the models of the Protein stored in self._protein and perform the following tasks. Clear the plotter. Add the specified meshes to the subplotter. Set the camera position. Render the plotter.

Parameters:

feature: str, optional Pass which feature (coloring) you want to plot. Options: hydrophob, shape, charge. Default: None (uniform coloring).

patch: bool, optional If True then coloring will be read in from "root directory"/data/tmp/{pdb_id}.pth file. Default: False.

title: str, optional Title of the plot window. Default: Surface.

box: bool, optional ptional - If True bounding box also visualized, default: 0.

res: Residue, optional Residues passed in 'res' will be plotted with a bounding box around them. Defaults to None.

outname: string, optional Save image of plot to specified filename. Will appear in data/img directory. Defaults to data/img/{pdb_id}_stick_point.png.

camera: pyvista.Camera, optional Pass a Pyvista Camera https://docs.pyvista.org/api/core/camera.html to manually set the camera position. If nothing/None is passed then the camera position will be set to [0, 4 * "max distance from the center", 0] (see: https://pro-vis.readthedocs.io/en/latest/tutorial.html for more detail). Default: None.

Returns:

Pyvista.Plotter window Window with interactive plot

str - only if self._notebook is True The full path to where the .mp4 video file is stored.

`plot_vw`(*box=0*, *res=0*, *outname=0*, *camera=None*)

Plot Van-der-Waals radius of atoms as spheres. Spheres have a wireframe style to be able to view inner structure as well. To plot Van-der-Waals radii as solid spheres use the manual_plot() member function.

Parameters:

box: bool, optional ptional - If True bounding box also visualized, default: 0.

res: Residue, optional Residues passed in 'res' will be plotted with a bounding box around them. Defaults to None.

outname: string, optional Save image of plot to specified filename. Will appear in data/img directory. Defaults to data/img/{pdb_id}_atoms.png.

camera: pyvista.Camera, optional Pass a Pyvista Camera https://docs.pyvista.org/api/core/camera.html to manually set the camera position. If nothing/None is passed then the camera position will be set to [0, 4 * "max distance from the center", 0] (see: https://pro-vis.readthedocs.io/en/latest/tutorial.html for more detail). Default: None.

Returns:

Pyvista.Plotter window Window with interactive plot.

provis.src.plotting.plotter module

`class provis.src.plotting.plotter.Plotter(prot: provis.src.processing.protein.Protein, prot2=None, msms=True, notebook=False, plot_solvent=False)`

Bases: `object`

The Plotter class is used for plotting structural and surface information of one or more proteins.

Add Proteins to the Plotter to plot them next to one another. While it is possible to add more than two proteins to one Plotter class it is discouraged, as the window will get cluttered.

The class can visualize two kinds of surfaces:

- a chemically accurate surface created by the MSMS binary.
- a good approximation of the surface computed natively with o3d and trimesh. (MSMS does not have to be installed for this option. It is fast, but less precise.)

Choose between the two by setting the msms Boolean variable. (Default: True, corresponding to the MSMS binary option.)

`add_protein(protein: provis.src.processing.protein.Protein)`

Add another Protein to the (internal list of the) Plotter object.

Parameters:

protein: Protein Instance of Protein class.

`manual_plot(box=False, res=None, outname=None, atoms=None, col_a=None, bonds=None, vw=0, residues=None, col_r=None, bb=None, camera=None)`

Plots list of meshes directly. One can get these meshes from the DataHandler class.

Parameters:

box: bool, optional If True bounding box also visualized. Default: False.

res: list, optional List of pyvista Shperes representing each residue. Default: None.

outname: string, optional save image of plot to specified filename. Will appear in data/img directory. Defaults to data/img/{self._out_path}_stick_point.

atoms: list, optional List of pyvista Shperes representing each atom. Default: None.

col_a: list, optional List of colors for each atom. Default: None.

bonds: list, optional List of pyvista Lines representing each bond. Default: None.

vw: bool, optional If True styling for Van-der-Waals plotting set. Vw atomic objects still have to be passed under 'atoms' variable.

col_r: list, optional List of colors for each residue. Default: None.

res: Residue, optional Specified residues will be plotted with a bounding box around them.

bb: pyvista.Spline, optional Spline describing the back-bone of the protein. Default: None.

camera: pyvista.Camera, optional Pass a Pyvista Camera https://docs.pyvista.org/api/core/camera.html to manually set the camera position. If nothing/None is passed then the camera position will be set to [0, 4 * "max distance from the center", 0] (see: https://pro-vis.readthedocs.io/en/latest/tutorial.html for more detail). Default: None.

Returns:

Pyvista.Plotter window Window with interactive plot

`plot_atoms`(*box=0*, *res=0*, *outname=None*, *camera=None*)
Plot the atoms as spheres. Each atom has a radius proportianal to its calculated atomic radius.

Consult https://en.wikipedia.org/wiki/CPK_coloring for the coloring.

Parameters:

box: bool, optional ptional - If True bounding box also visualized. Default: None.

res: Residue, optional Residues passed in 'res' will be plotted with a bounding box around them. Defaults to None.

outname: string, optional Save image of plot to specified filename. Will appear in data/img directory. Defaults to data/img/{pdb_id}_atoms.png.

camera: pyvista.Camera, optional Pass a Pyvista Camera https://docs.pyvista.org/api/core/camera.html to manually set the camera position. If nothing/None is passed then the camera position will be set to [0, 4 * "max distance from the center", 0] (see: https://pro-vis.readthedocs.io/en/latest/tutorial.html for more detail). Default: None.

Returns:

Pyvista.Plotter window Window with interactive plot.

`plot_backbone`(*box=0*, *res=0*, *outname=0*, *camera=None*)
Plots the backbone (roughly the amide bonds) of the protein.

Parameters:

box: bool, optional If True bounding box also visualized. Default: None.

res: Residue, optional Residues passed in 'res' will be plotted with a bounding box around them. Defaults to None.

outname: string, optional Save image of plot to specified filename. Will appear in data/img directory. Defaults to data/img/{pdb_id}_backbone.png.

camera: pyvista.Camera, optional Pass a Pyvista Camera https://docs.pyvista.org/api/core/camera.html to manually set the camera position. If nothing/None is passed then the camera position will be set to [0, 4 * "max distance from the center", 0] (see: https://pro-vis.readthedocs.io/en/latest/tutorial.html for more detail). Default: None.

Returns:

Pyvista.Plotter window Window with interactive plot.

`plot_bonds`(*box=0*, *res=0*, *outname=0*, *colorful=False*, *camera=None*)
Plot only the bonds. By default all bonds will be plotted uniformly.

If the difference in bond types is of interest set the "colorful" variable to True. Coloring: - Single bonds: white - Double bonds: blue - Triple bonds: green - Amide bonds: red - Aromatic bonds: purple - Undefined/Anything else: black

Parameters:

box: bool, optional If True bounding box also visualized. Default: None.

res: Residue, optional Residues passed in 'res' will be plotted with a bounding box around them. Defaults to None.

outname: string, optional Save image of plot to specified filename. Will appear in data/img directory. Defaults to data/img/{pdb_id}_atoms.png.

colorful: bool, optional If True bonds will be plotted in a colorful manner. If False all bonds are white. Default: False

camera: pyvista.Camera, optional Pass a Pyvista Camera https://docs.pyvista.org/api/core/camera.html to manually set the camera position. If nothing/None is passed then the camera position will be set to [0, 4 * "max distance from the center", 0] (see: https://pro-vis.readthedocs.io/en/latest/tutorial.html for more detail). Default: None.

Returns:

Pyvista.Plotter window Window with interactive plot.

`plot_charge`(*box=None*, *res=None*, *outname=None*, *camera=None*)
Plot the charge features of a protein.

Parameters:

box, optional: bool, optional If True bounding box also visualized. Default: None.

res: Residue, optional Residues passed in 'res' will be plotted with a bounding box around them. Defaults to None.

outname: string, optional save image of plot to specified filename. Will appear in data/img/ directory. Default: data/img/{self._out_path}_surface.

camera: pyvista.Camera, optional Pass a Pyvista Camera https://docs.pyvista.org/api/core/camera.html to manually set the camera position. If nothing/None is passed then the camera position will be set to [0, 3 * "max distance from the center", 0] (see: https://pro-vis.readthedocs.io/en/latest/tutorial.html for more detail). Default: None.

Returns:

Pyvista.Plotter window Window with interactive plot.

`plot_hydrophob`(*box=None*, *res=None*, *outname=None*, *camera=None*)
Plot the hydrophobic features of a protein.

Parameters:

box, optional: bool, optional If True bounding box also visualized. Default: None.

res: Residue, optional Residues passed in 'res' will be plotted with a bounding box around them. Defaults to None.

outname: string, optional save image of plot to specified filename. Will appear in data/img/ directory. Default: data/img/{self._out_path}_surface.

camera: pyvista.Camera, optional Pass a Pyvista Camera https://docs.pyvista.org/api/core/camera.html to manually set the camera position. If nothing/None is passed then the camera position will be set to [0, 3 * max_distance_from_center, 0]. Default: None.

Returns:

Pyvista.Plotter window Window with interactive plot.

`plot_residues`(*box=0*, *res=0*, *outname=0*, *camera=None*)
Plot the residues as Spheres. Each sphere is the approximate size of the radius of the given residue. This plot should only be used to get a general feel for the layout of the protein.

For coloring information please visit: http://acces.ens-lyon.fr/biotic/rastop/help/colour.htm

Parameters:

box: bool, optional ptional - If True bounding box also visualized. Default: None.

res: Residue, optional Residues passed in 'res' will be plotted with a bounding box around them. Defaults to None.

outname: string, optional Save image of plot to specified filename. Will appear in data/img directory. Defaults to data/img/{pdb_id}_residues.png.

camera: pyvista.Camera, optional Pass a Pyvista Camera https://docs.pyvista.org/api/core/camera.html to manually set the camera position. If nothing/None is passed then the camera position will be set to [0, 4 * "max distance from the center", 0] (see: https://pro-vis.readthedocs.io/en/latest/tutorial.html for more detail). Default: None.

Returns:

Pyvista.Plotter window Window with interactive plot.

`plot_shape`(*box=None*, *res=None*, *outname=None*, *camera=None*)
Plot the shape features of a protein.

Parameters:

box, optional: bool, optional If True bounding box also visualized. Default: None.

res: Residue, optional Residues passed in 'res' will be plotted with a bounding box around them. Defaults to None.

outname: string, optional save image of plot to specified filename. Will appear in data/img/ directory. Default: data/img/{self._out_path}_surface.

camera: pyvista.Camera, optional Pass a Pyvista Camera https://docs.pyvista.org/api/core/camera.html to manually set the camera position. If nothing/None is passed then the camera position will be set to [0, 3 * "max distance from the center", 0] (see: https://pro-vis.readthedocs.io/en/latest/tutorial.html for more detail). Default: None.

Returns:

Pyvista.Plotter window Window with interactive plot.

`plot_stick_point`(*box=0*, *res=None*, *outname=0*, *camera=None*)
Plot stick and point model of the protein. Atoms are spheres, bonds are tubes.

Parameters:

box: bool, optional ptional - If True bounding box also visualized. Default: None.

res: Residue, optional Residues passed in 'res' will be plotted with a bounding box around them. Defaults to None.

outname: string, optional Save image of plot to specified filename. Will appear in data/img directory. Defaults to data/img/{pdb_id}_stick_point.png.

camera: pyvista.Camera, optional Pass a Pyvista Camera https://docs.pyvista.org/api/core/camera.html to manually set the camera position. If nothing/None is passed then the camera position will be set to [0, 4 * "max distance from the center", 0] (see: https://pro-vis.readthedocs.io/en/latest/tutorial.html for more detail). Default: None.

Returns:

Pyvista.Plotter window Window with interactive plot.

`plot_structure`(*box=0*, *res=None*, *outname=0*, *atoms=0*, *bonds=0*, *vw=0*, *residues=0*, *bb=0*, *title='Structure'*, *camera=None*)

This member function is called by all the others. Using this function you can plot any combination of the results gotten from the specialized member functions. For example you could plot the atoms and the backbone of the protein in the same plot.

All information to be plotted is already computed. This function simply dictates what is to be plotted.

The code in words: Create a Plotter of size(1, {len(self._proteins)}). Loop through the Proteins in the self._proteins list and perform the following tasks. Add the specified meshes to the subplotter. Set the camera position. Show the plotter.

Parameters:

box: bool, optional ptional - If True bounding box also visualized. Default: None.

res: Residue, optional Residues passed in 'res' will be plotted with a bounding box around them. Defaults to None.

outname: string, optional Save image of plot to specified filename. Will appear in data/img directory. Defaults to {root directory}/data/img/{pdb_id}_{model_id}_stick_point.png. If Structure class was initialized with msms=True then output will have "_msms.png" as the ending.

atoms: bool, optional Plot atoms. Default: None.

bonds: int, optional ptional - Plot bond. If zero or undefined then it does not plot the bonds, if 1 it plots all bonds uniformly, if 2 it plots colorful bonds (see data_handler). Default: None.

vw: bool, optional Plot Wan-der-Waals radii instead of atomic radii.

residues: bool, optional ptional - Plot residue. Default: None.

bb: bool, optional If True backbone of protein is plotted. Default: False.

title: str, optional Title of the plot window. Default: Structure.

camera: pyvista.Camera, optional Pass a Pyvista Camera https://docs.pyvista.org/api/core/camera.html to manually set the camera position. If nothing/None is passed then the camera position will be set to [0, 4 * "max distance from the center", 0] (see: https://pro-vis.readthedocs.io/en/latest/tutorial.html for more detail). Default: None.

Returns:

Pyvista.Plotter window Window with interactive plot

`plot_surface`(*feature=None*, *title='Surface'*, *patch=False*, *box=None*, *res=None*, *outname=None*, *camera=None*)

Plot the surface of the protein. If the surface has already been computed and saved to the default file, then the surface will automatically be loaded from there. The surface can be computed either using the msms binary or natively. The msms binary is chemically accurate surface, while the native one is only for visualization purposes.

If you run into any sort of error concerning array size mismatching or of the sort delete all the temporary files and the mesh ({root directory}/data/meshes/{pdb_id}_{model_id}.obj). This will force everything to be recomputed and the dimension mismatch should disappear.

The code in words: Create a Plotter of size(1, {len(self._proteins)}). Loop through the Proteins in the self._proteins list and perform the following tasks. Add the specified meshes to the subplotter. Set the camera position. Show the plotter.

Parameters:

feature: str, optional Pass which feature (coloring) you want to plot. Options: hydrophob, shape, charge. Default: None (uniform coloring).

title: str, optional Title of the plot window. Default: Surface.

patch: bool, optional If True then coloring will be read in from "root directory"/data/tmp/{pdb_id}.pth file. Default: False.

box, optional: bool, optional If True bounding box also visualized. Default: None.

res: Residue, optional Residues passed in 'res' will be plotted with a bounding box around them. Defaults to None.

outname: string, optional Save image of plot to specified filename. Will appear in data/img directory. Defaults to {root directory}/data/img/{pdb_id}_{model_id}_surface.png. If Surface class was initialized with msms=True then output will have "_msms.png" as the ending.

camera: pyvista.Camera, optional Pass a Pyvista Camera https://docs.pyvista.org/api/core/camera.html to manually set the camera position. If nothing/None is passed then the camera position will be set to [0, 3 * "max distance from the center", 0] (see: https://pro-vis.readthedocs.io/en/latest/tutorial.html for more detail). Default: None.

Returns:

Pyvista.Plotter window Window with interactive plot.

`plot_vw`(*box=0*, *res=0*, *outname=0*, *camera=None*)

Plot Van-der-Waals radius of atoms as spheres. Spheres have a wireframe style to be able to view inner structure as well. To plot Van-der-Waals radii as solid spheres use the manual_plot_structure() member function.

Parameters:

box: bool, optional ptional - If True bounding box also visualized. Default: None.

res: Residue, optional Residues passed in 'res' will be plotted with a bounding box around them. Defaults to None.

outname: string, optional Save image of plot to specified filename. Will appear in data/img directory. Defaults to data/img/{pdb_id}_atoms.png.

camera: pyvista.Camera, optional Pass a Pyvista Camera https://docs.pyvista.org/api/core/camera.html to manually set the camera position. If nothing/None is passed then the camera position will be set to [0, 4 * "max distance from the center", 0] (see: https://pro-vis.readthedocs.io/en/latest/tutorial.html for more detail). Default: None.

Returns:

Pyvista.Plotter window Window with interactive plot.

provis.src.plotting.structure module

class provis.src.plotting.structure.**Structure**(*nc*, *dh=None*, *plot_solvent=False*, *notebook=False*, *msms=False*)

Bases: object

The Structure class is used to visualize the structural information of the given molecule. One can easily plot the atoms, residues, bonds or any combination of these structures.

manual_plot(*box=0*, *res=0*, *outname=0*, *atoms=0*, *col_a=0*, *bonds=0*, *vw=0*, *residues=0*, *col_r=0*, *bb=0*, *camera=None*)

Plot stick and point model. In this function one can pass all the desired meshes to be plotted. One can get these meshes from the DataHandler class.

Parameters:

box: bool, optional If True bounding box also visualized, default: 0.

res: list, optional List of pyvista Shperes representing each residue, default: 0.

outname: string, optional save image of plot to specified filename. Will appear in data/img directory. default: data/img/{self._out_path}_stick_point.

atoms: list, optional List of pyvista Shperes representing each atom, default: 0.

col_a: list, optional List of colors for each atom, default: 0.

bonds: list, optional List of pyvista Lines representing each bond, default: 0.

vw: bool, optional If True styling for Van-der-Waals plotting set. Vw atomic objects still have to be passed under 'atoms' variable.

col_r: list, optional List of colors for each residue, default: 0.

res: Residue, optional Specified residues will be plotted with a bounding box around them.

bb: bool, optional List of coordinates describing the back-bone of the protein, default: 0.

camera: pyvista.Camera, optional Pass a Pyvista Camera https://docs.pyvista.org/api/core/camera.html to manually set the camera position. If nothing/None is passed then the camera position will be set to [0, 4 * "max distance from the center", 0] (see: https://pro-vis.readthedocs.io/en/latest/tutorial.html for more detail). Default: None.

Returns:

Pyvista.Plotter window Window with interactive plot

plot(*box=0*, *res=None*, *outname=0*, *atoms=0*, *bonds=0*, *vw=0*, *residues=0*, *bb=0*, *title=None*, *camera=None*, *model_id=0*, *dynamic=False*)

This member function is called by all the others. Using this function you can plot any combination of the results gotten from the specialized member functions. For example you could plot the atoms and the backbone of the protein in the same plot.

All information to be plotted is already computed. This function simply dictates what is to be plotted.

Parameters:

box: bool, optional ptional - If True bounding box also visualized, default: 0.

res: Residue, optional Residues passed in 'res' will be plotted with a bounding box around them. Defaults to None.

outname: string, optional Save image of plot to specified filename. Will appear in data/img directory. Defaults to {root directory}/data/img/{pdb_id}_{model_id}_stick_point.png. If Structure class was initialized with msms=True then output will have "_msms.png" as the ending.

atoms: bool, optional Plot atoms, default: 0.

bonds: int, optional ptional - Plot bond. If zero or undefined then it does not plot the bonds, if 1 it plots all bonds uniformly, if 2 it plots colorful bonds (see data_handler). Default: 0.

vw: bool, optional Plot Wan-der-Waals radii instead of atomic radii.

residues: bool, optional ptional - Plot residue, default: 0.

bb: bool, optional If True backbone of protein is plotted. Default: False.

title: str, optional Title of the plot window. Defaults to None.

camera: pyvista.Camera, optional Pass a Pyvista Camera https://docs.pyvista.org/api/core/camera.html to manually set the camera position. If nothing/None is passed then the camera position will be set to [0, 4 * "max distance from the center", 0] (see: https://pro-vis.readthedocs.io/en/latest/tutorial.html for more detail). Default: None.

model_id: int, optional The dynamic model ID of the desired molecule. Count starts at 0. Leave default value for static molecules. Default: 0.

dynamic: bool, optional Set to True if you are plotting a dynamic model. Default: False.

Returns:

Pyvista.Plotter window Window with interactive plot

`plot_atoms`(*box=0*, *res=0*, *outname=None*, *camera=None*)

Plot the atoms as spheres. Each atom has a radius proportianal to its calculated atomic radius.

Consult https://en.wikipedia.org/wiki/CPK_coloring for the coloring.

Parameters:

box: bool, optional ptional - If True bounding box also visualized, default: 0.

res: Residue, optional Residues passed in 'res' will be plotted with a bounding box around them. Defaults to None.

outname: string, optional Save image of plot to specified filename. Will appear in data/img directory. Defaults to data/img/{pdb_id}_atoms.png.

camera: pyvista.Camera, optional Pass a Pyvista Camera https://docs.pyvista.org/api/core/camera.html to manually set the camera position. If nothing/None is passed then the camera position will be set to [0, 4 * "max distance from the center", 0] (see: https://pro-vis.readthedocs.io/en/latest/tutorial.html for more detail). Default: None.

Returns:

Pyvista.Plotter window Window with interactive plot.

`plot_backbone`(*box=0*, *res=0*, *outname=0*, *camera=None*)

Plots the backbone (roughly the amide bonds) of the protein.

Parameters:

box: bool, optional If True bounding box also visualized, default: 0.

res: Residue, optional Residues passed in 'res' will be plotted with a bounding box around them. Defaults to None.

outname: string, optional Save image of plot to specified filename. Will appear in data/img directory. Defaults to data/img/{pdb_id}_backbone.png.

camera: pyvista.Camera, optional Pass a Pyvista Camera https://docs.pyvista.org/api/core/camera.html to manually set the camera position. If nothing/None is passed then the camera position will

be set to [0, 4 * "max distance from the center", 0] (see: https://pro-vis.readthedocs.io/en/latest/tutorial.html for more detail). Default: None.

Returns:

Pyvista.Plotter window Window with interactive plot.

`plot_bonds`(*box=0*, *res=0*, *outname=0*, *colorful=False*, *camera=None*)
Plot only the bonds. By default all bonds will be plotted uniformly.

If the difference in bond types is of interest set the "colorful" variable to True. Coloring: - Single bonds: white - Double bonds: blue - Triple bonds: green - Amide bonds: red - Aromatic bonds: purple - Undefined/Anything else: black

Parameters:

box: bool, optional If True bounding box also visualized, default: 0.

res: Residue, optional Residues passed in 'res' will be plotted with a bounding box around them. Defaults to None.

outname: string, optional Save image of plot to specified filename. Will appear in data/img directory. Defaults to data/img/{pdb_id}_atoms.png.

colorful: bool, optional If True bonds will be plotted in a colorful manner. If False all bonds are white. Default: False

camera: pyvista.Camera, optional Pass a Pyvista Camera https://docs.pyvista.org/api/core/camera.html to manually set the camera position. If nothing/None is passed then the camera position will be set to [0, 4 * "max distance from the center", 0] (see: https://pro-vis.readthedocs.io/en/latest/tutorial.html for more detail). Default: None.

Returns:

Pyvista.Plotter window Window with interactive plot.

`plot_residues`(*box=0*, *res=0*, *outname=0*, *camera=None*)
Plot the residues as Spheres. Each sphere is the approximate size of the radius of the given residue. This plot should only be used to get a general feel for the layout of the protein.

For coloring information please visit: http://acces.ens-lyon.fr/biotic/rastop/help/colour.htm

Parameters:

box: bool, optional ptional - If True bounding box also visualized, default: 0.

res: Residue, optional Residues passed in 'res' will be plotted with a bounding box around them. Defaults to None.

outname: string, optional Save image of plot to specified filename. Will appear in data/img directory. Defaults to data/img/{pdb_id}_residues.png.

camera: pyvista.Camera, optional Pass a Pyvista Camera https://docs.pyvista.org/api/core/camera.html to manually set the camera position. If nothing/None is passed then the camera position will be set to [0, 4 * "max distance from the center", 0] (see: https://pro-vis.readthedocs.io/en/latest/tutorial.html for more detail). Default: None.

Returns:

Pyvista.Plotter window Window with interactive plot.

`plot_stick_point`(*box=0*, *res=None*, *outname=0*, *camera=None*)
Plot stick and point model of the protein. Atoms are spheres, bonds are tubes.

Parameters:

box: bool, optional ptional - If True bounding box also visualized, default: 0.

res: Residue, optional Residues passed in 'res' will be plotted with a bounding box around them. Defaults to None.

outname: string, optional Save image of plot to specified filename. Will appear in data/img directory. Defaults to data/img/{pdb_id}_stick_point.png.

camera: pyvista.Camera, optional Pass a Pyvista Camera https://docs.pyvista.org/api/core/camera.html to manually set the camera position. If nothing/None is passed then the camera position will be set to [0, 4 * "max distance from the center", 0] (see: https://pro-vis.readthedocs.io/en/latest/tutorial.html for more detail). Default: None.

Returns:

Pyvista.Plotter window Window with interactive plot.

`plot_vw`(*box=0*, *res=0*, *outname=0*, *camera=None*)

Plot Van-der-Waals radius of atoms as spheres. Spheres have a wireframe style to be able to view inner structure as well. To plot Van-der-Waals radii as solid spheres use the manual_plot() member function.

Parameters:

box: bool, optional ptional - If True bounding box also visualized, default: 0.

res: Residue, optional Residues passed in 'res' will be plotted with a bounding box around them. Defaults to None.

outname: string, optional Save image of plot to specified filename. Will appear in data/img directory. Defaults to data/img/{pdb_id}_atoms.png.

camera: pyvista.Camera, optional Pass a Pyvista Camera https://docs.pyvista.org/api/core/camera.html to manually set the camera position. If nothing/None is passed then the camera position will be set to [0, 4 * "max distance from the center", 0] (see: https://pro-vis.readthedocs.io/en/latest/tutorial.html for more detail). Default: None.

Returns:

Pyvista.Plotter window Window with interactive plot.

provis.src.plotting.surface module

`class` provis.src.plotting.surface.`Surface`(*nc*, *sh=None*, *msms=True*, *density=3.0*, *notebook=False*)

Bases: `object`

The Surface class is used to visualize the surface information of the given molecule.

The class can visualize two kinds of surfaces:

- a chemically accurate surface created by the MSMS binary.
- a good approximation of the surface computed natively with o3d and trimesh. (MSMS does not have to be installed for this option. It is fast, but less precise.)

Choose between the two by setting the msms Boolean variable. (Default: True, corresponding to the MSMS binary option.)

`plot`(*feature=None*, *title='Surface'*, *patch=False*, *box=None*, *res=None*, *outname=None*, *camera=None*, *model_id=0*)

Plot the surface of the protein. If the surface has already been computed and saved to the default file, then the surface will automatically be loaded from there. The surface can be computed either using the msms binary or natively. The msms binary is chemically accurate surface, while the native one is only for visualization purposes.

If you run into any sort of error concerning array size mismatching or of the sort delete all the temporary files and the mesh ({root directory}/data/meshes/{pdb_id}_{model_id}.obj). This will force everything to be recomputed and the dimension mismatch should disappear.

Parameters:

feature: str, optional Pass which feature (coloring) you want to plot. Options: hydrophob, shape, charge. Default: None (uniform coloring).

title: str, optional Title of the plot window. Default: Surface.

patch: bool, optional If True then coloring will be read in from "root directory"/data/tmp/{pdb_id}.pth file. Default: False.

box, optional: bool, optional If True bounding box also visualized, default: 0.

res: Residue, optional Residues passed in 'res' will be plotted with a bounding box around them. Defaults to None.

outname: string, optional Save image of plot to specified filename. Will appear in data/img directory. Defaults to {root directory}/data/img/{pdb_id}_{model_id}_surface.png. If Surface class was initialized with msms=True then output will have "_msms.png" as the ending.

camera: pyvista.Camera, optional Pass a Pyvista Camera https://docs.pyvista.org/api/core/camera.html to manually set the camera position. If nothing/None is passed then the camera position will be set to [0, 3 * "max distance from the center", 0] (see: https://pro-vis.readthedocs.io/en/latest/tutorial.html for more detail). Default: None.

model_id: int, optional The dynamic model ID of the desired molecule. Count starts at 0. Leave default value for static molecules. Default: 0.

Returns:

Pyvista.Plotter window Window with interactive plot.

`plot_charge`(*box=None*, *res=None*, *outname=None*, *camera=None*)
Plot the charge features of a protein.

Parameters:

box, optional: bool, optional If True bounding box also visualized, default: 0.

res: Residue, optional Residues passed in 'res' will be plotted with a bounding box around them. Defaults to None.

outname: string, optional save image of plot to specified filename. Will appear in data/img/ directory. Default: data/img/{self._out_path}_surface.

camera: pyvista.Camera, optional Pass a Pyvista Camera https://docs.pyvista.org/api/core/camera.html to manually set the camera position. If nothing/None is passed then the camera position will be set to [0, 3 * "max distance from the center", 0] (see: https://pro-vis.readthedocs.io/en/latest/tutorial.html for more detail). Default: None.

Returns:

Pyvista.Plotter window Window with interactive plot.

`plot_hydrophob`(*box=None*, *res=None*, *outname=None*, *camera=None*)
Plot the hydrophobic features of a protein.

Parameters:

box, optional: bool, optional If True bounding box also visualized, default: 0.

res: Residue, optional Residues passed in 'res' will be plotted with a bounding box around them. Defaults to None.

outname: string, optional save image of plot to specified filename. Will appear in data/img/ directory. Default: data/img/{self._out_path}_surface.

camera: pyvista.Camera, optional Pass a Pyvista Camera https://docs.pyvista.org/api/core/camera.html to manually set the camera position. If nothing/None is passed then the camera position will be set to [0, 3 * max_distance_from_center, 0]. Default: None.

Returns:

Pyvista.Plotter window Window with interactive plot.

`plot_shape`(*box=None*, *res=None*, *outname=None*, *camera=None*)
Plot the shape features of a protein.

Parameters:

box, optional: bool, optional If True bounding box also visualized, default: 0.

res: Residue, optional Residues passed in 'res' will be plotted with a bounding box around them. Defaults to None.

outname: string, optional save image of plot to specified filename. Will appear in data/img/ directory. Default: data/img/{self._out_path}_surface.

camera: pyvista.Camera, optional Pass a Pyvista Camera https://docs.pyvista.org/api/core/camera.html to manually set the camera position. If nothing/None is passed then the camera position will be set to [0, 3 * "max distance from the center", 0] (see: https://pro-vis.readthedocs.io/en/latest/tutorial.html for more detail). Default: None.

Returns:

Pyvista.Plotter window Window with interactive plot.

Module contents

4.1.2 provis.src.processing package

These classes constitute the "brain" of provis. All computation is done here (sometimes by calling the utils package).

The results/outputs of these classes are passed to the plotting classes for plotting.

provis.src.processing.data_handler module

`class provis.src.processing.data_handler.DataHandler`(*nc*, *fc=None*)
Bases: `object`

The 'brain' of provis, when it comes to handling atomic positions.

This class loads information from a variety of files and creates meshes to be plotted. Upper level classes - eg. Protein - use DataHandler objects to create the meshes.

The DataHandler class loads atom-positional information from a .pdb file and from this information computes the necessairy molecular structure mesh. It also loads pre-defined dictionaries from atminfo.py, that encode the size, coloring and mass of a given atom or residue. The member functions range from loading the atoms from the .pdb file and storing them by type, to creating the meshes from this information, as well as calculating bonds or the backbone.

`get_atom_mesh`(*atom_data*, *vw=0*, *probe=0*, *phi_res=10*, *theta_res=10*)

Create a list of Shperes and colors representing each atom for plotting. Can later be added to a mesh for plotting.

The code in words: Iterates through the atom_data dictionary by atom type (from get_atoms()). It creates uniform Spheres (same size and color) in the position specified by the coordinates list for each atom type. Also differentiates between Van der Waals and normal radii and handles unkown atoms.

Parameters:

atom_data: dict Dictionary of atoms and their coordinates, by atom type.

vw: bool, optional ptional - When set to True Van-der-Waals atomic radii used instead of empirical radii. Default: False.

probe: int, optional size of probe (representing the solvent size) needed for surface calculation. Default: 0.

phi_res: int, optional pyvista phi_resolution for Sphere objects representing atoms. Default: 10.

theta_res: int, optional pyvista theta_resolution for Sphere objects representing atoms. Default: 10.

Returns:

list List of pyvista Shperes representing each atom

list List of colors corresponding to each atom

list List of atom ID's for each atom

`get_atom_trimesh`(*atom_data*, *vw=False*, *probe=0*)

Create a list of shperes and colors representing each atom for plotting in a Trimesh format. Used for feature computation in the surface_handler class.

The code in words: Iterates through the atom_data dictionary by atom type (from get_atoms()). It creates uniform Spheres (same size and color) in the position specified by the coordinates list for each atom type. Also differentiates between Van der Waals and normal radii and handles unkown atoms.

Parameters:

atom_data: dict Dictionary of atoms and their coordinates, by atom type.

vw: bool, optional ptional - When set to True Van-der-Waals atomic radii used instead of empirical radii. Default: False.

probe: int, optional size of probe (representing the solvent size) needed for surface calculation. Default: 0.

Returns:

list List of pyvista Shperes representing each atom

list List of colors for each atom

list List of atom ID's for each atom

`get_atoms`(*show_solvent=False*, *model_id=0*)

Creates a dictionary that stores the 3D coordinates for each atom. The dictionary keys are the atom names. For each atom type in the given molecule the coordinates of the atoms of this type are stored in a list within the dictionary.

The code in words: The .pdb file (loaded in __init__()) is iterated through. For each atom it is checked if the type of this atom is already in the dictionary. If not then a new list is created with the coordinates of this atom and added to the dictionary with the name of the atom as the key. If the name of this atom is already

present then the coordinates of the current atom are added to the list of coordinates of this same type of atom. The dictionary is returned.

Parameters:

show_solvent: bool, optional If True solvent molecules also added to retrun dictionary. Default: False.

model_id: int, optional The dynamic model ID of the desired molecule. Count starts at 0. Leave default value for static molecules. Default: 0.

Returns:

dict Dictionary of atomic coordinates by atom type.

int Maximum coordinate in y axis. (Used to create default camera)

`get_atoms_IDs`(*model_id=0*)

Get dictionary of atomic coordinates (same as get_atoms()) and residue IDs (format from output_pdb_as_xyzrn()) from the xyzrn file. Also return a list of all the atomic coordinates in a list in the same order as in the .pdb file.

Used by the Surface class.

The code in words: The .xyzrn file is loaded and is iterated through. The relevant fields are stored to temporary variables. For each atom it is checked if the type of this atom is already in the dictionary. If not then a new list is created with the coordinates of this atom and added to the dictionary with the name of the atom as the key. If the name of this atom is already present then the coordinates of the current atom are added to the list of coordinates of this same type of atom. Regardless of the atom already being present in the dictionary the residue id and the coordinates are added to the two lists. The dictionary and the two lists are returned.

Parameters:

model_id: int, optional The dynamic model ID of the desired molecule. Count starts at 0. Leave default value for static molecules. Default: 0.

Returns:

dict Dictionary of atomic coordinates by atom type.

list List of unique residue IDs (format from output_pdb_as_xyzrn())

list Atomic coordinates (in same order as the residue IDs)

`get_backbone_mesh`(*model_id=0*)

Creates and returns a Spline object representing the backbone of the protein.

The code in words: Iterates through the res_data dictionary by atom type (from get_residues()). Calculates the center of each residue and returns these points as a numpy array. (Later used to create a Spline.)

Parameters:

model_id: int, optional The dynamic model ID of the desired molecule. Count starts at 0. Leave default value for static molecules. Default: 0.

Returns:

pyvista.Spline Spline running through coordinates representing the centre of mass of each residue.

`get_bond_mesh`(*model_id=0*)

Determine bonds from 3D information.

The color information is as follows: White for all single bonds, Blue for all double bonds, Green for all triple bonds, Red for all amide bonds, Purple for all aromatic bonds, Black for everything else.

The code in words: Parse mol2 file (also works on multi model file). Find where the boundaries of the current molecule are in the file. Extract the atomic and bond information by creating DataFrame. From the DataFrames get the information corresponding to the current bond: create a pyvista.Line() and store the bond type. Return the compiled lists of Lines and bond types (colors).

Parameters:

model_id: int, optional The dynamic model ID of the desired molecule. Count starts at 0. Leave default value for static molecules. Default: 0.

Returns:

list List of pyvista lines representing each bond.

list List of colors corresponding to the lines in the above list

list List of names of the bonds: single, double, triple, amide, aromatic, unkown

`get_residue_info`(*res*, *chain*, *option*)

Calculates information about specified residue from mol2 file. Depending on what is specified, either the center of mass (COM) or the charge is computed.

Parameters:

res: str Residue number of specified residue be looked at.

chain: choose what property of residue you want. com for Centre Of Mass, ch for charge Chain number of corresponding to residue be looked at. str - options: com, ch

Returns:

list List of COM coords of given (exact) residue.

`get_residue_mesh`(*res_data*, *phi_res=25*, *theta_res=25*)

Create a list of Shperes and colors representing each residue for plotting. Can later be added to a mesh for plotting.

The code in words: Iterates through the res_data dictionary by atom type (from get_residues()). It creates uniform Spheres (same size and color) in the position specified by the coordinates list for each residue type. Also differentiates between Van der Waals and normal radii and handles unkown residues.

Parameters:

res_data: dict Dictionary of residues and their coordinates by residue type.

phi_res: int, optional pyvista phi_resolution for Sphere objects representing residues. Defaul: 25.

theta_res: int, optional pyvista theta_resolution for Sphere objects representing residues. Default:25.

Returns:

list List of pyvista Shperes representing each residue

list List of colors for each residue

list List of residue names

`get_residues`(*model_id=0*, *show_solvent=False*)

Creates a dictionary of coordinates by residues from structure object

The code in words: The .pdb file (loaded in __init__()) is iterated through. For each residue it is checked if the type of this residue is already in the dictionary. If not then a new list is created with the coordinates of this residue and added to the dictionary with the type of the residue as the key. If the type of this residue is already present then the coordinates of the current residue are added to the list of coordinates of this same

type of residue. The coordinates of the residue are calculated as the arithmetic center of the coordinates. The dictionary is returned.

Parameters:

model_id: int, optional The dynamic model ID of the desired molecule. Count starts at 0. Leave default value for static molecules. Default: 0.

show_solvent: bool, optional If True solvent molecules also added to retrun dictionary. Default: False.

Returns:

dict Dictionary of atomic coordinates by residue type.

`get_structure()`
Return the loaded structure object

Returns: structure

provis.src.processing.file_converter module

`class provis.src.processing.file_converter.FileConverter(nc, density=3.0, convert_all=False)`
Bases: `object`

Class to create and destroy necessary files required in other parts of code. The member functions call the binaries and scripts to convert the files.

Also has a cleanup function that removes everything in the "root_directory"/data/tmp (and data/img if specified) directory. Best practice is to call this function at the end of your main file. However if you want to plot the same protein many times, then it is benificial to keep the temporary (data/tmp) files as if they exist provis will not recompute them.

`cleanup(delete_img: bool = False, delete_meshes: bool = False)`
Deletes all files related to the current .pdb id from data/tmp (and if specified, the data/img and data/meshes) directories.

CAUTION: provis does not recompute existing files. So if you have a molecule that you want to plot multiple times then do not delete the temporary files. WARNING: as provis does not recompute existing files it might occur that an old version of the file is stored in the temporary directories and this might cause provis to fail. If this is the case simply delete all temporary files (as well as meshes).

Parameters:

delete_img: bool, optional If True all files of the form {pdb_id}_{*} will be deleted from the data/meshes directory. (pdb_id is the name of the .pdb file without the .pdb extension and {*} represents that "anything"). Default: False.

delete_meshes: bool, optional If True all files of the form {pdb_id}_{*} will be deleted from the data/meshes directory. (pdb_id is the name of the .pdb file without the .pdb extension and {*} represents that "anything"). Default: False

`decompose_traj(path)`
As the MSMS binary is unable to work with trajectory .pdb files the large .pdb file containing all models needs to be decomposed to one file per model. This function completes this exact task.

Only executes decomposition if the first file ("{pdb_file_name}_0.pdb") does not exists to avoid unnecessairy recomputation.

Parameters:

path: str Name of input (pdb) file (without extension)

Returns:

int Number of models in the trajectory.

`msms`(*path*, *dens*)
Run the msms binary for given filename.

It takes the {path}.xyzrn file as input and output is written to {path}_out_{dens} Binary path is read in from environment variable: MSMS_BIN. If environment variable does not exist binary will be looked up in provis/binaries/msms.

Parameters:

path: str Path to the/Name of the .xyzrn file to be converted.

dens: float Density of triangulation

Returns:

void face and vert files

`pdb_to_mol2`(*path*, *outpath*)
Run openbabel, to convert pdb to mol2

Parameters:

path: str Name of input (pdb) file (without extension)

outpath: str Name of desired output file (without extension). It will add .mol2 to the given path.

Returns:

void mol2 file

`pdb_to_pqr`(*path*, *outpath*, *forcefield='swanson'*)
Run the pdb2pqr binary for given filename.

It takes the {path}.pdb file as input and output is written to {outpath}.pqr. Binary path is read in from environment variable: MSMS_BIN. If environment variable does not exist binary will be looked up in provis/binaries/msms.

Binary path is read in from environment variable: PDB2PQR_BIN. If environment variable does not exist binary will be looked up in binaries/pdb2pqr/pdb2pqr.

Parameters:

path: str Name of input (pdb) file (without extension)

outpath: str Name of desired output file (without extension). It will add .pqr to the given path.

forcefield: str, optional Force field used for charge computation, by binary. Default: swanson. Options: amber, charmm, parse, tyl06, peoepb and swanson

Returns:

void pqr file

`pdb_to_xyzrn`(*path*, *output*)
Converts .pdb to .xyzrn file.

Parameters:

path: str Name of input (pdb) file (without extension)

output: str Name of output (xyzrn) file (without extension)

Returns:

void xyzrn file

provis.src.processing.name_checker module

class provis.src.processing.name_checker.**NameChecker**(*name*, *base_path: Optional[str] = None*)
Bases: object

This class provides uniform names and path locations to all the other classes of provis. NameChecker has internal variables and a method to return these variables.

Class member variables: self._pdb_name - Full path to the pdb file without the .pdb extension. Usually PROVIS_PATH/data/pdb/{pdb_id}. self._out_path - Full path to the temporary files. The names of all temporary files are derived from this variable. Usually PROVIS_PATH/data/tmp/{pdb_id}. self._base_path - Full path of the provis directory or any directory that has the following directory structure within: {path}/data/data, {path}/data/img, {path}/data/tmp, {path}/data/pdb.

return_all()
Return all class variables. This function is used by all other provis classes to retrieve the paths to the files needed.

Returns:

str path to the input pdb id (file without the extension)

str path to the output pdb id - data/tmp/{pdb_id}

str path to the root of the provis directory. Following file structure HAS TO exist within: {path}/data/data, {path}/data/img, {path}/data/tmp, {path}/data/pdb.

provis.src.processing.protein module

class provis.src.processing.protein.**Protein**(*pdb_name*, *base_path=None*, *density=3.0*, *model_id=0*)
Bases: object

The protein class encapsulates every other class in the provis.src.processing package.

This class contains all the necessary information need for plotting.

provis.src.processing.residue module

class provis.src.processing.residue.**Residue**(*id=None*, *chain=0*, *padding=0*)
Bases: object

Residue class is used for plotting a bounding box around the specified residue.

add_residue(*id*, *chain=0*)
Add a new residue to the internal list of residues.

Parameters:

id: int, optional Residue id. Count starting at 0. Default: None.

chain: int, optional Chain id. Count starting at 0. Specify which chain the residue is on. Default 0 (in case of single chain). Default: 0.

get_res_info()
Returns all internal information of class

Returns:

list: list of the current residues

list: list of chain ID's corresponding to residues

int: padding for bounding box

`remove_residue`(*id*, *chain=0*)
Remove speciefied residue from internal list.

Parameters:

id: int, optional Residue id. Count starting at 0. Default: None.

chain: int, optional Chain id. Count starting at 0. Specify which chain the residue is on. Default 0 (in case of single chain). Default: 0.

provis.src.processing.surface_handler module

`class provis.src.processing.surface_handler.SurfaceHandler`(*nc*, *fc=None*, *dh=None*, *density=3.0*)
Bases: `object`

The 'brain' of provis, when it comes to handling surfaces.

This class loads information from a variety of files and creates meshes to be plotted. Upper level classes - eg. Surface - use SurfaceHandler objects to create the meshes.

The SurfaceHandler class loads atom-positional information from a .pdb file and from this information computes the surface mesh. The surface can be computed by the MSMS binary or natively. The MSMS version is chemically more accurate and faster, but the MSMS binary has to be downloaded for it to work.

`get_assignments`()
Get assignments (coloring) for the mesh. File has to exist, no way to produce it with provis. Loads "root directory"/data/tmp/{pdb_id}.pth and returns it.

Returns:

PyTourch object Coloring of surface.

`get_surface_features`(*mesh*, *feature*, *res_id=None*)
Get the coloring corresponding to a specific feature.

The code in words: Creates the pqr file if it does not exist. If the res_id variable is not empty retrieve the surface structure (list of specific surface-related information) using the get_surface() function. Else compile the surface structure from the mesh and the find_nearest_atom() function. Next, using the above mentioned surface structure compute all surface feature information. Finally return the coloring corresponding to the specified feature.

Parameters:

mesh: Trimesh The mesh

feature: str Name of feature we are interested in. Options: hydrophob, shape, charge, hbonds.

res_id: bool, optional List of unique residue IDs (format from output_pdb_as_xyzrn()). Default: False.

Returns:

numpy.ndarray Array of coloring corresponding to surface.

Raises:

NotImplementedError If unkown feature specified error is raised

`msms_mesh_and_color`(*feature=None*, *patch=False*)
Return the mesh and coloring created by the MSMS binary.

The code in words: If self._mesh_needed is set to True - if the mesh could not be loaded from a file - compute the mesh using the .face and .vert files created by the MSMS binary. If the mesh is needed compute the mesh from the .face and .vert files. Finally, if a feature is specified check if the color information could be loaded from a file (self._color_needed) and compute it if needed. If no feature specified set the variable that stores color information to None. This will result in a white mesh.

Parameters:

feature: str, optional Name of feature, same as in get_surface_features. Options: hydrophob, shape, charge, hbonds. Defaults to None.

patch: bool, optional Set coloring of mesh manually, from a file. If set to True get_assignments() will be called. Defaults to False.

`native_mesh_and_color`(*feature=None*)
Returns a mesh without the need for the MSMS binary. The mesh and coloring is also saved to the following file names: Mesh: "root directory"/data/meshes/{pdb_id}_{model_id}.obj Color: "root directory"/data/meshes/{pdb_id}_{feature}_{model_id} Always returns a pyvista.PolyData mesh of the surface and if a feature is specified it also returns the coloring according to that feature.

The code in words: If self._mesh_needed is set to True - if the mesh could not be loaded from a file - compute the mesh using the native tools. Get the atomic positional information from the DataHandler class. Extract the surface of the combined mesh and use the o3d.geometry.TriangleMesh.create_from_point_cloud_poisson() method to create a smooth surface. Finally, if a feature is specified check if the color information could be loaded from a file (self._color_needed) and compute it if needed. If no feature specified set the variable that stores color information to None. This will result in a white mesh.

Parameters:

feature: str, optional Name of feature, same as in get_surface_features. Options: hydrophob, shape, charge, hbonds. Defaults to "".

`return_mesh_and_color`(*msms=False*, *feature=None*, *patch=False*, *model_id=0*)
Wrapper function to choose between the msms surface visualization vs the native surface visualization. If you could not download the MSMS binary leave the msms variable as False.

If the mesh and appropriate coloring has already been stored to a file, then this information will be loaded and no computation will be done.

The code in words: First, set a few class member variables that are used in the {*}_mesh_and_color() member functions. Next check if the mesh and color information has already been computed. If the files already exist load them and return. No computation will be done. Otherwise, depending on what the msms input variable is set to, compute either the msms_mesh_and_color() ir the native_mesh_and_color(). Return the mesh and color information.

Parameters:

msms: bool, optional If true, surface generated by msms binary is returned, else the native mesh. Default: False.

feature: str, optional Name of feature, same as in get_surface_features. Options: hydrophob, shape, charge, hbonds. Defaults to None.

patch: bool, optional Set coloring of mesh manually. If set to True get_assignments() will be called. Defaults to False.

model_id: int, optional The dynamic model ID of the desired molecule. Count starts at 0. Leave default value for static molecules. Default: 0.

Returns:

trimesh.Trimesh The mesh corresponding to the surface of the protein.

numpy.ndarray Coloring map corresponding to specified feature.

Module contents

provis.src.processing.**get_residues**(*pdb_file*)

4.1.3 Module contents

4.2 Utilities for Provis

All of these files contain scripts and helper methods that are used by the core classes of **provis**.

4.2.1 provis.utils.atminfo module

provis.utils.atminfo.**import_atm_mass_info**()
Funtion to load dictionary storing atomic mass information by atom type.

dict dictionary of atomic mass by atom name

provis.utils.atminfo.**import_atm_size_info**(*vw=False*)
Funtion to load dictionaries storing atomic radii, color coding and Van-der-Waals radii by atom name.

Coloring from: https://sciencenotes.org/molecule-atom-colors-cpk-colors/

Parameters:

vw: bool, optional Option to return vanderwaals radius. Default: False.

Returns:

dict dictionary of atomic radius by atom name

dict dictionary of color by atom name

dict return dictionary of vw rdius by atom name

provis.utils.atminfo.**import_res_size_info**()
Funtion to load dictionaries storing residue radii and color coding by residue name.

Coloring from: http://acces.ens-lyon.fr/biotic/rastop/help/colour.htm

Parameters:

dict dictionary of radius by residue name

dict dictionary of color by residue name

4.2.2 provis.utils.charges_utils module

Utility functions for computing hydrogen bonds and electrostatics on protein surface.

provis.utils.charges_utils.**compute_angle_deviation**(*a: numpy.ndarray*, *b: numpy.ndarray*, *c: numpy.ndarray*, *theta: float*) → float

Computes the absolute angle deviation from theta. a, b, c form the three points that define the angle.

Parameters:

a: np.ndarray, Coordinate vector of the first point.

b: np.ndarray, Coordinate vector of the second point.

c: np.ndarray, Coordinate vector of the third point.

theta: float Angle to compute deviation with respect to.

Returns:

float absolute deviation of the angle formed by a, b, c with theta

provis.utils.charges_utils.**compute_angle_penalty**(*angle_deviation: float*) → float

Compute the angle penalty corresponding to angle of deviation.

Parameters:

angle_deviation: float Angle of deviation.

Returns:

float Angle penalty.

provis.utils.charges_utils.**compute_hbond_helper**(*atom_name: str*, *res: Bio.PDB.Residue.Residue*, *v: numpy.ndarray*) → float

Helper function. Computes the hydrogen bond for given atom.

Parameters:

atom_name: str Name of atom.

res: Residue Residue corresponding to atom.

v: np.ndarray Vertices.

Returns:

float hydrogen bonds of the atom

provis.utils.charges_utils.**compute_plane_deviation**(*a: numpy.ndarray*, *b: numpy.ndarray*, *c: numpy.ndarray*, *d: numpy.ndarray*) → float

Computes the absolute plane deviation from theta. a, b, c form the three points that define the angle.

Parameters:

a: np.ndarray, Coordinate vector of the first point.

b: np.ndarray, Coordinate vector of the second point.

c: np.ndarray, Coordinate vector of the third point.

theta: float Angle to compute deviation with respect to.

Returns:

float absolute deviation of the angle formed by a, b, c with theta

provis.utils.charges_utils.**compute_satisfied_CO_HN**(*atoms*)
Compute the list of backbone C=O:H-N that are satisfied. These will be ignored.

Parameters:

atoms: BioPython atoms list of atoms to be checked.

Returns:

set set of C=O bonds

set set of H-N bonds

provis.utils.charges_utils.**is_acceptor_atom**(*atom_name: str*, *res: Bio.PDB.Residue.Residue*) → bool
Check if atom is acceptor atom.

Parameters:

atom_name: str Name of the atom

res: Residue The corresponding residue

Returns:

bool True if conditions met

provis.utils.charges_utils.**is_polar_hydrogen**(*atom_name: str*, *res_name: str*) → bool
Check if the atom in a given residue has polar hydrogens.

Parameters:

atom_name: str Name of the atom

res_name: str Residue name

provis.utils.charges_utils.**normalize_electrostatics**(*in_elec: numpy.ndarray*) → numpy.ndarray
Normalizing charges on the surface, by clipping to upper and lower thresholds and converting all values to a -1/1 scale.

Parameters:

in_elec: np.ndarray Input charges for all surface vertices

Returns:

np.ndarray Normalized surface vertex charges

4.2.3 provis.utils.surface_feat module

https://github.com/bunnech/holoprot/blob/main/holoprot/feat/surface.py is the base for this file. Modifications were made.

Functions to compute features for a patch on the protein surface.

Some of these are borrowed from https://github.com/LPDI-EPFL/masif under the https://github.com/LPDI-EPFL/masif/blob/master/LICENSE license

provis.utils.surface_feat.**assign_props_to_new_mesh**(*new_vertices*, *old_vertices: numpy.ndarray*, *old_props: numpy.ndarray*, *feature_interpolation: bool = True*) → numpy.ndarray
Assign properties to vertices in modified mesh given the initial mesh. The assignment is carried using a KDTree data structure to query nearest points.

Parameters:

new_vertices: np.ndarray Vertices on the modified mesh

old_vertices: np.ndarray Vertices on the original mesh

old_props: np.ndarray Property values for each vertex on the original mesh

feature_interpolation: bool, (default True) If set to True interpolates features to new vertices.

Returns:

np.ndarray Property values for vertices on the modified mesh

provis.utils.surface_feat.**compute_charges**(*vertices: numpy.ndarray*, *pdb_id: str*, *path: str*) → numpy.ndarray

Computes electrostatics for the surface vertices. The function first calls the PDB2PQR executable to prepare the pdb file for electrostatics. Poisson- Boltzmann electrostatics are computed using APSB executable. Multivalue, provided within APSB suite is used to assign charges to each vertex. The charges are further normalized.

Parameters:

vertices: np.ndarray Surface vertex coordinates

pdb_id: str PDB ID of the protein

path: str Path of the pqr file in the form {path}.pqr

Returns:

np.ndarray Charge values for each vertex

provis.utils.surface_feat.**compute_hbonds**(*vertices: numpy.ndarray*, *residues: List[Bio.PDB.Residue.Residue]*, *names: List[str]*) → numpy.ndarray

Compute H-bond (hydrogen-bond) induced charges at every vertex.

Parameters:

vertices: np.ndarray Vertices of mesh

residues: List[Residue] List of residues to compute

names: List[str] List of custom names created by output_pdb_as_xyzrn()

Returns:

np.ndarray Array of bonds, by vertex

provis.utils.surface_feat.**compute_hydrophobicity**(*names: List[str]*) → numpy.ndarray

Compute hydrophobicity value for all vertices on the surface. Each surface vertex has a mapping to the corresponding residue from the original protein. This is used to assign a hydrophobicity value to each vertex using the Kyte- Doolittle scale.

Parameters:

names: List[str] Identifier names for each vertex in the surface

Returns:

np.ndarray Hydrophobicity values for each surface vertex

provis.utils.surface_feat.**compute_shape_index**(*mesh*) → numpy.ndarray

Computes shape index for the patches. Shape index characterizes the shape around a point on the surface, computed using the local curvature around each point. These values are derived using Trimesh's available geometric processing functionality.

Parameters:

mesh: Trimesh The mesh is constructed using information about vertices and faces.

Returns:

np.ndarray Shape index for each vertex

provis.utils.surface_feat.**compute_surface_features**(*surface: Tuple[numpy.ndarray, numpy.ndarray, numpy.ndarray, List[str], Dict[str, str]]*, *pdb_file: str*, *path: str*, *mesh=None*, *fix_mesh: bool = False*, *return_mesh: bool = False*, *pdb_id: Optional[str] = None*) → Tuple[numpy.ndarray]

Computes all patch features.

Parameters:

surface: Surface Tuple of attributes characterizing the surface. These include vertices, faces, normals to each vertex, areas, fdue identifiers for vertices.

pdb_file: str PDB File containing the atomic coordinates

path: str Path to files without extensions. Usually data/tmp/{pdb_id}

mesh: trimesh.Trimesh The mesh.

fix_mesh: bool, optional Whether to fix the mesh by collapsing nodes and edges. Default: False.

return_mesh: bool, optional Whether to return the mesh. Default: False.

pdb_id: str, optional PDB id of the associated protein. Default: None

Returns:

np.ndarray Shape index

np.ndarray Hydrogen bond induced charges

np.ndarray Hydrophobicity of each surface vertex

np.ndarray Electrostatics of each surface vertex

4.2.4 provis.utils.surface_utils module

https://github.com/bunnech/holoprot/blob/main/holoprot/utils/surface.py is the base for this file. Modifications were made.

Utilities for preparing and computing features on molecular surfaces.

provis.utils.surface_utils.**compute_normal**(*vertices: numpy.ndarray*, *faces: numpy.ndarray*) → numpy.ndarray

Compute normals for the vertices and faces

Parameters:

vertices: np.ndarray Vertices of the mesh

faces: np.ndarray Faces of the mesh

Returns:

np.ndarray: Normals of the mesh

provis.utils.surface_utils.**crossp**(*x: numpy.ndarray*, *y: numpy.ndarray*) → numpy.ndarray

Creates the cross product of two numpy arrays

Parameters:

x: np.ndarray Array 1

y: np.ndarray Array 2

Returns:

np.ndarray: (Array 1) x (Array 2)

provis.utils.surface_utils.**find_nearest_atom**(*coords*, *res_id*, *new_verts*)

provis.utils.surface_utils.**fix_trimesh**(*mesh*, *resolution: float = 1.0*)

Applies a predefined set of fixes to the mesh, and converts it to a specified resolution. These fixes include removing duplicated vertices wihin a certain threshold, removing degenerate triangles, splitting longer edges to a given target length, and collapsing shorter edges.

Parameters:

mesh: trimesh.Trimesh Mesh

resolution: float Maximum size of edge in the mesh

Returns:

trimesh.Trimesh: mesh with all fixes applied

provis.utils.surface_utils.**get_surface**(*out_path: str*, *density: float*, *center=[0, 0, 0]*)

Wrapper function that reads in the output from the MSMS executable to build the protein surface.

out_path: str path to output (output path from namechecker) directory. Usually data/tmp

density: bool Need to pass same density as used by the MSMS binary, as the face and vert files have the density included in their names. The variable is needed for loading these files.

center: List[float], optional Center of the atom cloud. Easily passed from DataHandler._centroid. Default: [0, 0, 0].

Returns:

numpy.ndarray: vertices

numpy.ndarray: faces

numpy.ndarray: vertex normals

list: list of res_id's from output_pdb_as_xyzrn()

dict: dictionary: residues as keys, areas as values

provis.utils.surface_utils.**output_pdb_as_xyzrn**(*pdb_file: str*, *xyzrn_file: str*) → None

Converts a .pdb file to a .xyzrn file.

Parameters:

pdb_file: str path to PDB File to convert (with extension)

xyzrn_file: str path to the xyzrn File (with extension)

provis.utils.surface_utils.**prepare_trimesh**(*vertices: numpy.ndarray*, *faces: numpy.ndarray*, *normals: Optional[numpy.ndarray] = None*, *apply_fixes: bool = False*)

Prepare the mesh surface given vertices and faces. Optionally, compute normals and apply fixes to mesh.

Parameters:

vertices: np.ndarray Surface vertices

faces: np.ndarray Triangular faces on the mesh

normals: np.ndarray Normals for each vertex

apply_fixes: bool Optional application of fixes to mesh. Check fix_mesh for details on fixes. Default: False,

Returns:

trimesh.Trimesh: Mesh

provis.utils.surface_utils.**read_msms**(*file_root: str*) → Tuple[numpy.ndarray, numpy.ndarray, numpy.ndarray, List[str]]

Read surface constituents from output files generated using MSMS.

file_root: str Root name for loading .face and .vert files (produced by MSMS). Default location is data/tmp/{pdb_id}s

Parameters:

numpy.ndarray: vertices

numpy.ndarray: faces

numpy.ndarray: vertex normals

list: list of res_id's from output_pdb_as_xyzrn()

4.2.5 Module contents

provis.utils.**get_residues**(*pdb_file*)

provis.utils.**str2bool**(*v: str*) → bool

Converts str to bool.

Parameters

- **name** – v - String element
- **type** – str

Returns boolean version of v

4.3 Module contents

CHAPTER

FIVE

GENERAL INFORMATION ABOUT PROVIS

This section describes the general code architecture and design decisions of the **provis** library.

5.1 MSMS

As explained in the Getting started section, there are two ways to compute the surface. Using the **msms** binary or natively.

Therefore, it is very important to specify which version you want to use. This can be done by setting the **msms** input variable in the constructor of the class that handles the surface plotting. These classes are: **Surface**, **Protein** and **DynamicPlotter** (all three classes ultimately rely on the **Surface** class in the background).

5.2 Classes

5.2.1 Protein

The **Protein** class encapsulates the whole processing package of the **provis** library. You can initialize it with a simple pdb filename and the class can then be used to retrieve all the necessairy information for plotting.

5.2.2 NameChecker

The **NameChecker** class is responsible for storing the paths to all the files used by **provis**. All other classes require a **NameChecker** class instance to be passed. Passing this **NameChecker** instance ensures that all other classes will work with the same molecule.

5.2.3 FileConverter

The **FileConverter** class is responsible for creating the required files from the *.pdb* file. This class is hidden in other classes and only called when a file is needed/missing.

IMPORTANT: the **FileConverter** class will not overwrite any existing files. This decision was made to not create the exact same file again and again. But this also means that if you have temporary file (used by the **FileConverter** class) that is corrupted or has bad information it will be used by **provis**. If you have any such problems you can always just delete all temporary files and then they will simply be newly created.

5.2.4 DataHandler

The **DataHandler** class processes the structural information of the protein and creates the meshes for plotting.

5.2.5 SurfaceHandler

The **SurfaceHandler** class processes the structural information of the protein and creates the meshes for plotting.

5.2.6 Plotter

The **Plotter** class is responsible for static plotting. It plots the internal data of the passed **Protein** class(es). If multiple **Protein** classes are passed, then they will be plotted in a side-by-side manner.

5.2.7 DynamicPlotter

Similar to the **Plotter** class, the **DynamicPlotter** class plots instances of the **Protein** class. This class however, focuses on dynamic plotting. It plots the trajectory of a molecule.

See: *Dynamic Structures*.

5.2.8 Residue

The **Residue** class is a very basic class. With this class you can specify a given residue on a given chain that you want to mark in the plot. It is passed to the plotting classes and a red box is drawn around the specified residue.

5.3 Design decisions

Since creating meshes requires a lot of computation it was decided to store the results of these computations to files. This way if a protein is analysed with **provis** for the second time, then results of the expensive computation will already be known. This way one can simply load the information from the files and plot right away.

Some classes of **provis** need the same information and need access to the same methods. For this reason, most classes of **provis** can be initialised with already existing instances of the internal variables. This way the exact same computations (for example: initialising the passed class) will be mitigated. Passing initialised classes also insures that the same molecule is being considered throughout the whole program.

To ensure that only necessary computations are run the existence of the object to be computed is always checked. Both files and class instance variables are checked. If the given object already exists (and possible other checks are passed) then this object will not be recomputed. **WARNING: this decision can result in an error if a new file with a previously existing name is analysed. **Provis** will not know that the molecule is different and errors might occur. To fix this simply delete (or move) all temporary files related to this molecule.** This is a very rare problem, if you use **provis** normally and do not modify existing files then you will not run into this problem.d

5.4 Code architecture

In the following will show how the above mentioned classes depend on one another.

One can easily observe that all classes depend on the **NameChecker** class. And since some classes have other dependencies as well, duplication of the exact same class instance would seem quite likely. This is the reason (as described above) why all classes can be initialised with pre-existing instances of the necessary classes, so instead of duplication they will be shared. (**Disclaimer**: the classes can also be initialised empty, but then the above mentioned duplication occurs.)

```
FileConverter
    └── NameChecker
```

```
DataHandler
    ├── NameChecker
    └── FileConverter
```

```
SurfaceHandler
    ├── NameChecker
    ├── FileConverter
    └── DataHandler
```

```
Protein
    ├── NameChecker
    ├── FileConverter
    ├── DataHandler
    └── SurfaceHandler
```

```
Plotter
    └── Protein
```

```
DynamicPlotter
    └── Protein
```

CHAPTER

SIX

HOW TO USE PROVIS

This section will explain how to use provis and how to get your desired plots.

In short you have to initialize a **Protein** class instance with the desired *.pdb* file. Then you have to create a **Plotter** or a **DynamicPlotter** class instance and pass the previously created **Protein** to it. Once the **(Dynamic)Plotter** is initialized you can plot using its plotting methods.

6.1 Loading a pdb

Provis uses the information in *.pdb* files to plot your desired protein. The most straightforward way is to pass the full path of the file to provis, as the *.pdb* file can be saved anywhere if you pass the full path. For example save the path to the **name** variable:

```
name = "/home/username/provis/data/pdb/2fd7.pdb"
```

As explained in *Setting up Provis*, provis requires a specific directory structure. If you have your *.pdb* file stored in the **/data/pdb** directory you do not have to specify a full path to the pdb file, but simply the name of the file:

```
name = "2fd7.pdb" # "2fd7" also works
```

6.2 Initializing the Protein class

The Protein class encompasses and combines all classes of the **provis.src.processing** package.

It first figures out the location of the *.pdb* file using the **NameChecker** class. Then it instantiates a **FileConverter** class so the necessairy temporary files can be converted from our *.pdb* file.

Then a **DataHandler** class is created. This class calculates all the necessairy non-surface related meshes, such as the Spheres corresponding to each atom or the backbone of the molecule. Next, a **SurfaceHandler** class handles all the surface related computation.

```
prot = Protein(name, base_path=None, density=3.0)
```

Specify the **name** of the *.pdb* file.

The path to the special direcotry explained in *Setting up Provis* has to also be provided. This is passed in the **base_path** variable and should point to the root directory of the **/data** and **/binaries** directories.

6.3 Plotter class

Use the Plotter class to plot. At least one Protein has to be passed. If two proteins are passed then they will be plotted side-by-side. It is possible to add more proteins later using the Plotter.add_protein(Protein) method.

```
prot2 = Protein(name, model_id=30)
plot = Plotter(prot, prot2, msms=msms, notebook=notebook)
```

6.3.1 MSMS

You will have to also specify if you want to plot the **msms** binary version of the surface or the simpler native mesh. If the **msms** option is chosen you can also specify the **density** of the triangulation (to be passed to the **msms** binary).

Solvent atoms can also be plotted by setting the **plot_solvent** variable to *True*.

And finally if you are working in a Jupyter Notebook like environment then set the **notebook** varaible to *True*.

6.3.2 Plotting

Plotting can be achieved by calling the member functions of the **Structure** and the **Surface** classes. For example for the **prot** class instance defined above the bonds of the molecule can be plotted as follows:

```
plot.plot_bonds()
```

All plotting functions have the following input variables:

- box (bool): If True bounding box will be plotted around molecule.
- res (Residue): Specified residue will be marked with a red box.
- outname (str): Save image of the plot to the file passed in this variable (otherwise saved in data/img).
- camera (pyvista.camera): A pyvista camera object to be make it easier to set a fixed camera position to compare two molecules.

Some of the plotting functions have additional input variables. One example; **plot_bonds**():

- colorful (bool): If True different bond types will be plotted in different colors.

  Single bonds: white Double bonds: blue Triple bonds: green Amide bonds: red Aromatic bonds: purple Undefined/Anything else: black

6.3.3 Camera

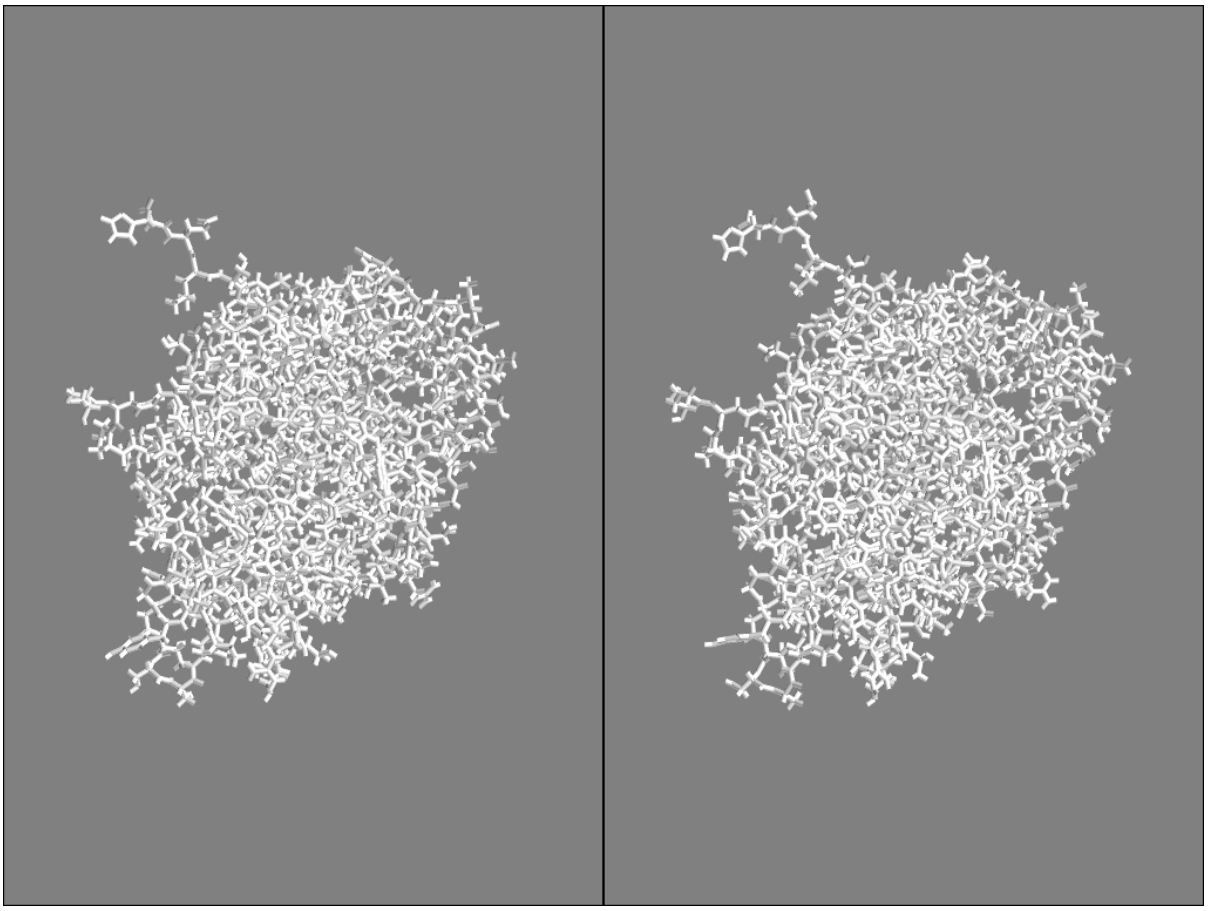
Setting a good camera position is very important. By default the camera portion is set to **[0, max * 3, 0]**, where **max** is the largest deviation of the coordinates from the center of the molecule. This ensures that the whole molecule is visible in the plot window and that the camera will always face the same direction when plotting dynamically.

To set the camera position manually the **DataHandler** class' instance variables named **DataHandler._cam_pos** (the default camera position) and **DataHandler._max_coords** (the maximum deviation, as explained above) might be helpful.

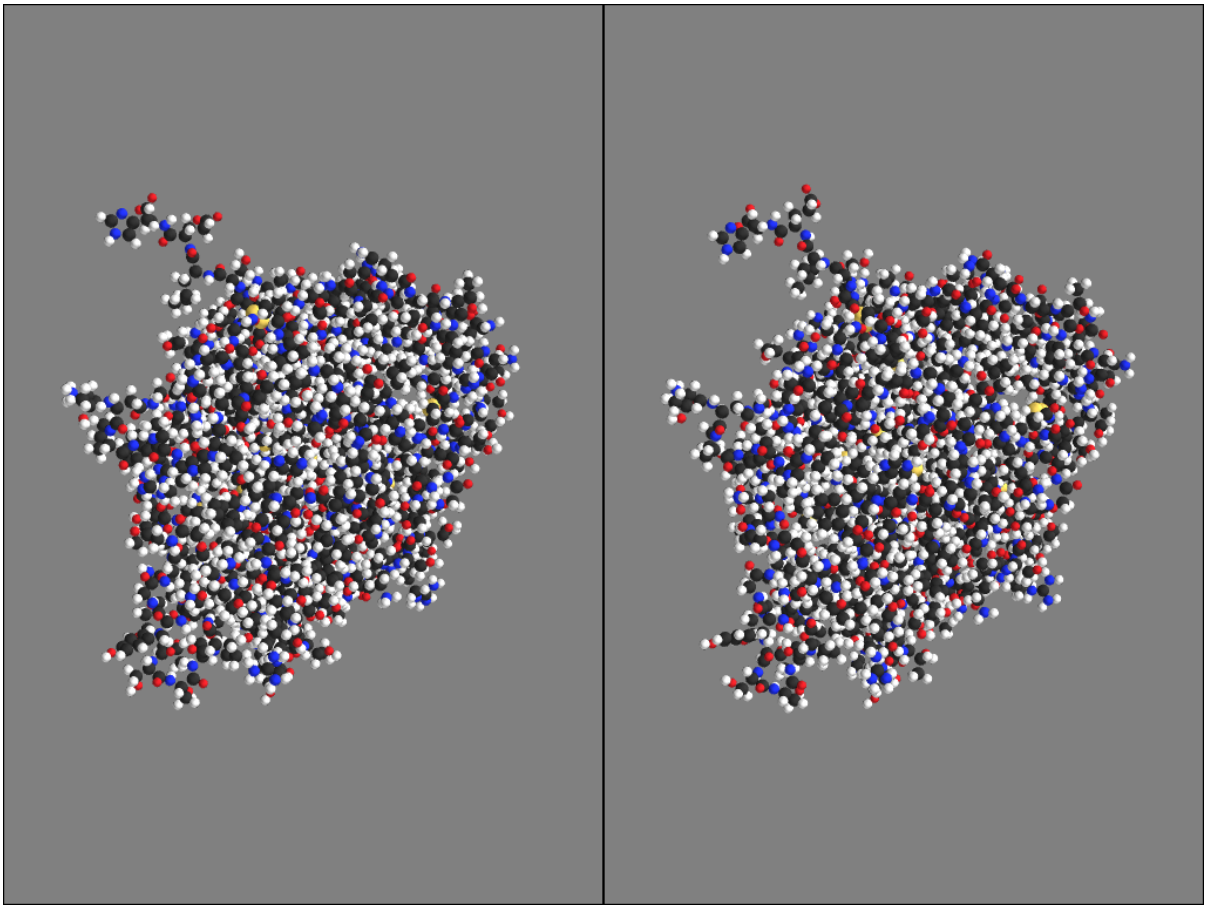
6.3.4 The output

The output will be an interactive **vtk.Window** window. The output will also be saved as an image to *"root directory"/data/img*.

The following image are the bonds of the 1st and 31st model of a given dynamic trajectory.



The following image are the atoms of the 1st and 31st model of a given dynamic trajectory.



CHAPTER

SEVEN

EXAMPLE

This is an example file to showcase the easiest way to run provis in particular how to plot a single protein. For this you should have this file in the root directory of the special directroy structure specified in the setup section of the documentation. Otherwise set the **base_path** variable of the **NameChecker** object.

If this is fullfilled path to the *"root directory"/data/tmp* will automatically be found. This way you can have your pdb files nicely organized in the data/pdb directory (or simply have them in the root directory). Your temporary files will be in the *data/tmp* directory and the screenshots of the plots in the *data/img* directory.

Import the necessairy files.

```
from provis.src.processing.protein import Protein
from provis.src.processing.residue import Residue
```

First: Define variables needed later:

```
name = "2fd7"
density = 3.0
plot_solvent = False
msms = True
notebook = False
```

Second: Create a **Protein** class instance. Initialize it with your *.pdb* file name and other parameters.

If you want to plot multiple proteins (or different models of the same trajectory) this is also possible. Simply create a second **Protein** class instance and pass it to the **Plotter**

```
prot = Protein(name, base_path=None, density=density)
prot2 = Protein(name, base_path=None, density=density, model_id=30)
```

Initialize the **Plotter** class. This creates all the necessairy classes in the background and you are already good to go!

```
plot = Plotter(prot, prot2, msms=msms, notebook=notebook, plot_solvent=plot_solvent)
```

Third: Plot!

Use the **Plotter** to plot.

```
plotter.plot_backbone()
plotter.plot_atoms()
plotter.plot_bonds()
plotter.plot_vw()
plotter.plot_stick_point()
plotter.plot_residues()
```

(continues on next page)

(continued from previous page)

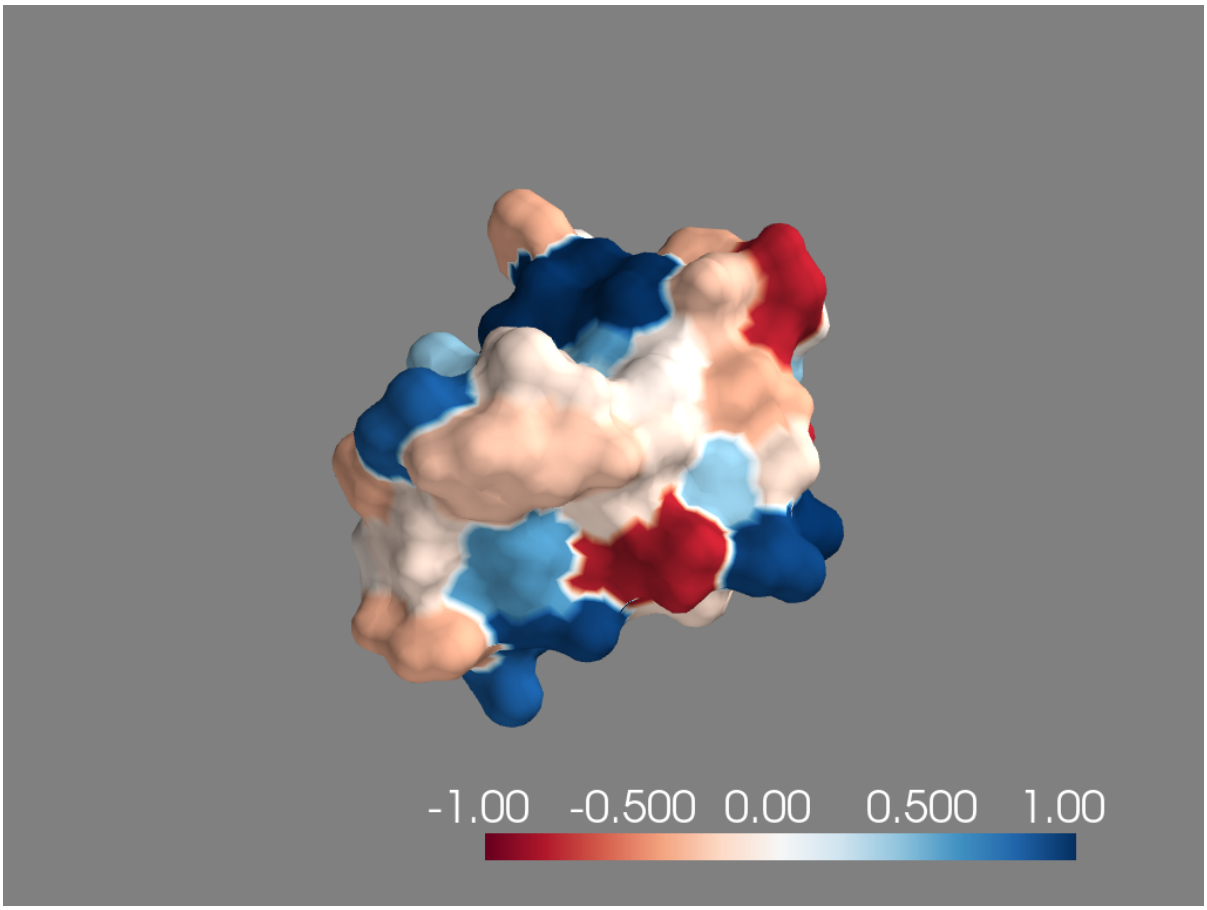
```
r = Residue(1)
r.add_residue(3)
r. add_residue(1, 1)
r.remove_residue(1, 1)
plotter.plot_structure(atoms=1, box=1, bonds=1, vw=0, residues=0, res=r, bb=0)

plotter.plot_surface()
plotter.plot_hydrophob()
plotter.plot_shape()
plotter.plot_charge()
```

And finally clean up everything with the "cleanup" function of the **Protein.file_converter** (**FileConverter** class) member variable.

```
prot.file_converter.cleanup()
```

The following image shows the hydrophobicity of the 2fd7 protein.



CHAPTER

EIGHT

DYNAMIC STRUCTURES

Dynamic structures showcase the transformation of a molecule in time. This trajectory of change can also be plotted with provis.

The **DynamicPlotter** class achieves exactly that. Unlike the **Protein** class, the **DynamicPlotter** class has its own member functions used for plotting. This is due to the fact that each model (molecule at a given time) has to be plotted seperately and this is achieved by a loop within the member function.

For large molecules and a long trajectory the surface meshes can take a while to compute. It is advised to first precompute the meshes (they will be stored in *.obj* files in the **"root directory"/data/meshes** directory). Once the *.obj* files exist plotting will be seamless.

The result of the plot will be saved to the **"root directory"/data/img** folder as an *.mp4* file.

8.1 How to use the DynamicPlotter class?

The **DynamicPlotter** class is very similar to the **Protein** class.

Import the necessairy files.

First: Define variables needed later:

```
name = "2fd7"
density = 3.0
```

Second: Create a **Protein** class instance. Initialize it with your *.pdb* file name and other parameters.

```
prot = Protein(name, base_path=None, density=density)
```

Initialize the **DynamicPlotter** class. This creates all the necessairy classes in the background and you are already good to go!

```
dp = DynamicPlotter(prot, msms=msms, notebook=notebook, plot_solvent=plot_solvent)
```

Third: Plot!

This class has its own plotting methods. Here is a complete list:

```
dp.plot_backbone()
dp.plot_atoms()
dp.plot_bonds()
dp.plot_vw()
dp.plot_stick_point()
```

(continues on next page)

(continued from previous page)

```
dp.plot_residues()
dp.plot_structure(atoms=1, box=1, bonds=1, vw=0, residues=0, res=r, bb=0)

dp.plot_surface()
dp.plot_hydrophob()
dp.plot_shape()
dp.plot_charge()
```

And finally, cleaning up is also possible with the "cleanup" function of the Protein.file_converter (**FileConverter** class) member variable.

```
prot.file_converter.cleanup()
```

CHAPTER

NINE

CONTACT AND MOTIVATION

Provis was developed by Kristof Czirjak as his Bachelor's Thesis at ETH Zurich at the Learning and Adaptive Systems laboratory.

If you have any questions feel free to get in touch.

Email: czirjakk@student.ethz.ch

CHAPTER

TEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

INDEX

I

M

N

O

P

R

S