proteusPy: A Python Package for Protein Structure and Disulfide Bond Modeling and Analysis

Eric G Suchanek [1*]

1 Flux-Frontiers * These authors contributed equally.

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Open Journals

Reviewers:

- @openjournals

Submitted: 01 January 1970
Published: unpublished

Summary

proteusPy is a Python package specializing in the modeling and analysis of proteins of known structure with an initial focus on Disulfide bonds. This package significantly extends the capabilities of the molecular modeling program **proteus**, (Pabo & Suchanek, 1986), and utilizes a new implementation of the Turtle3D class for disulfide and protein modeling. This initial implementation focuses on the Disulfide class, which implements methods to analyze the protein structure stabilizing element known as a **Disulfide Bond**.

The work has resulted in a freely-accessible database of over 120,494 disulfide bonds contained within 35,818 proteins in the RCSB Protein Databank. The routines within the library are capable of extracting, comparing, and visualizing the disulfides contained within the database, facilitating analysis and understanding. In addition, the package can readily model disulfide bonds of arbitrary conformation, facilitating predictive analysis.

General Capabilities

- Interactively display disulfides contained in the RCSB in a variety of display styles
- Calculate geometric and energetic properties about these disulfides
- Create binary and sextant structural classes by characterizing the disulfide torsional angles into *n* classes
- Build idealized disulfide bonds from disulfide dihedral angle input
- Find disulfide neighbors based on dihedral angles
- Overlap disulfides onto a common frame of reference for display
- Build protein backbones from backbone phi, psi dihedral angle templates
- More in development

See https://suchanek.github.io/proteusPy/proteusPy.html for the API documentation with examples

Statement of Need

Disulfide bonds represent the sole naturally occurring covalent bond in proteins, playing a pivotal role in structural stabilization within and between protein subunits. Moreover, they participate in enzymatic catalysis, regulate protein activities, and offer protection against oxidative stress. Establishing an accessible structural database of these disulfides would serve as an invaluable resource for exploring these critical structural elements. While the capability to visualize protein structures is well established with excellent protein visualization tools like Pymol, Chimera and the RCSB itself, the tools for disulfide bond analysis are more limited. (Wong & Hogg, 2010) describe a web-based disulfide visualization tool; this is currently unavailable.

Accordingly, I have developed the **proteusPy** package to delve into the RCSB Protein Data Bank, furnishing tools for visualizing and analyzing the disulfide bonds contained therein. This endeavor necessitated the creation of a python-based package containing data structures and algorithms capable loading, manipulating and analyzing these entities. Consequently, an object-oriented database has been crafted, facilitating introspection, analysis, and display. The package's API (Suchanek, 2023a) is accessible online at: proteusPy API, offering comprehensive details and numerous illustrative examples.

Requirements

1. PC running MacOS, Linux, Windows with git, git-lfs and make installed
2. 8 GB RAM
3. 1 GB disk space

Installation

It's simplest to clone the repo via GitHub since it contains all of the notebooks, data and test programs. Installation includes installing my Biopython fork. This is required to rebuild the database. I highly recommend using Miniforge since it includes mamba. The installation instructions below assume a clean install with no package manager or compiler installed.

MacOS/Linux

- Install Miniforge: https://github.com/conda-forge/miniforge (existing Anaconda installations are fine but please install mamba)
- Install git-lfs:
  - https://help.github.com/en/github/managing-large-files/installing-git-large-file-storage
- Install `make` on your system.
- From a shell prompt while sitting in your repo dir:

```
$ git clone https://github.com/suchanek/proteusPy.git
$ cd proteusPy
$ make pkg
$ mamba activate proteusPy
$ mamba install vtk
$ make install
```

Windows

- Install Miniforge: https://github.com/conda-forge/miniforge (existing Anaconda installations are fine but please install mamba)
- Install git for Windows and configure for Bash:
  - https://git-scm.com/download/win
- Install git-lfs:
  - https://git-lfs.github.com/
- Install GNU make:
  - https://gnuwin32.sourceforge.net/packages/make.htm
- Open a Miniforge prompt and cd into your repo dir:

```
(base) C:\Users\egs\repos> git clone https://github.com/suchanek/proteusPy.git
(base) C:\Users\egs\repos> cd proteusPy
(base) C:\Users\egs\repos\proteuspy> make pkg
(base) C:\Users\egs\repos>\proteuspy> conda activate proteusPy
(proteusPy) C:\Users\egs\repos> make install
```

Testing

I currently have `pytest` and docstring testing for the modules in place. To run them cd into the repository and run:

```
$ make tests
```

The modules will 1) run pytest for the main modules and 2) perform docstring tests. This will result in a number of disulfide visualization windows to open. Simply close them. If all goes normally there will be no errors. (you may need to install `pytest` via `pip install pytest`.

General Usage

Once the package is installed one can use the existing notebooks for analysis of the RCSB Disulfide database.

The notebooks directory contains all of my Jupyter notebooks and is a good place to start:

- Analysis_2q7q.ipynb provides an example of visualizing the lowest energy Disulfide contained in the database and searching for nearest neighbors on the basis of conformational similarity.

The programs subdirectory contains the primary programs for downloading the RCSB disulfide-containing structure files:

- DisulfideDownloader.py: Downloads the raw RCSB structure files.
- DisulfideExtractor.py: Extracts the disulfides and creating the database loaders
- DisulfideClass_Analysis.py: Performs binary or sextant analysis on the disulfide database.

The first time one loads the database via Load_PDB_SS() the system will attempt to download the full and subset database from Google Drive. If this fails the system will attempt to rebuild the database from the repo's **data** subdirectory (not the package's). If you've downloaded from github this will work correctly. If you've installed from pyPi via **pip** it will fail.

Quickstart

After installation is complete, launch jupyter lab:

```
$ jupyter notebook
```

and open Analysis_2q7q. This notebook analyzes the disulfide bond with the lowest energy in the entire database and performs some searches in dihedral angle space to find similar conformations. There are several other notebooks in this directory that illustrate using the program. Some of these reflect active development work so may not be 'fully baked'.

Class Details

The primary classes developed for **proteusPy** are described briefly below. Please see the API for details.

Disulfide

This class provides a Python object and methods representing a physical disulfide bond either extracted from the RCSB protein databank or a virtual one built using the Turtle3D class. The disulfide bond is an important intramolecular stabilizing structural element and is characterized by:

- Atomic coordinates for the atoms N, C_α, C_β, C', S_γ for both amino acid residues. These are stored as both raw atomic coordinates as read from the RCSB file and internal local coordinates.
- The dihedral angles $\chi_1 - \chi_5$ for the disulfide bond
- A name, by default: {pdb_id}{prox_resnumb}{prox_chain}_{distal_resnum}{distal_chain}
- Proximal residue number
- Distal residue number
- Approximate bond torsional energy (kcal/mol):

$$E_{kcal/mol} \approx 2.0 * cos(3.0 * \chi_1) + cos(3.0 * \chi_5) + cos(3.0 * \chi_2)+$$
$$cos(3.0 * \chi_4) + 3.5 * cos(2.0 * \chi_3) + 0.6 * cos(3.0 * \chi_3) + 10.1$$

- Euclidean length of the dihedral angles (degrees) defined as:

$$\sqrt{(\chi_1^2 + \chi_2^2 + \chi_3^2 + \chi_4^2 + \chi_5^2)}$$

- $C_\alpha - C_\alpha$ distance (Å)
- $C_\beta - C_\beta$ distance (Å)
- The previous C' and next N coordinates for both the proximal and distal residues. These are needed to calculate the backbone dihedral angles ϕ, ψ.
- Backbone dihedral angles ϕ and ψ, when possible. Not all structures are complete and in those cases the atoms needed may be undefined. In this case the ϕ and ψ angles are set to -180°.

The class also provides 3D rendering capabilities using the excellent PyVista library, and can display disulfides interactively in a variety of display styles:

- 'sb' - Split Bonds style - bonds colored by their atom type
- 'bs' - Ball and Stick style - split bond coloring with small atoms
- 'pd' - Proximal/Distal style - bonds colored *Red* for proximal residue and *Green* for the distal residue.
- 'cpk' - CPK style rendering, colored by atom type:
  - Carbon - Grey
  - Nitrogen - Blue
  - Sulfur - Yellow
  - Oxygen - Red
  - Hydrogen - White

Individual renderings can be saved to a file and animations can be created. The *cpk* and *bs* styles are illustrated below:

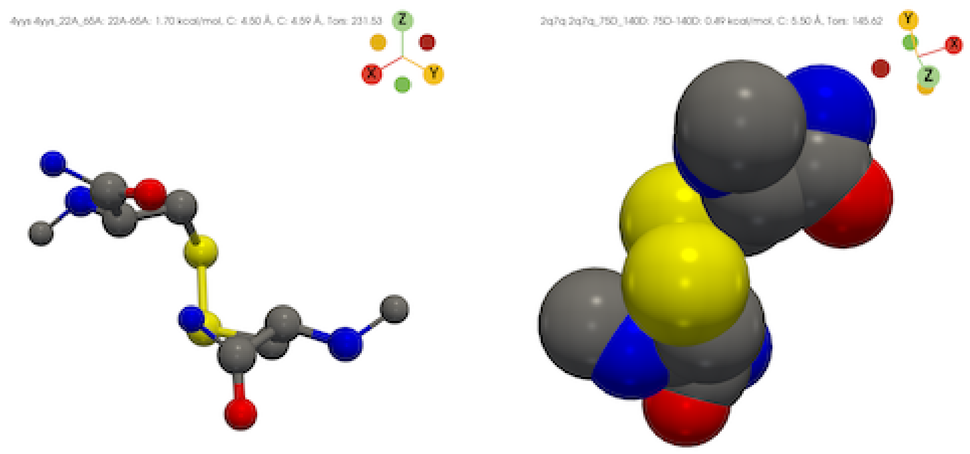
4yys 4yys_22A_65A: 22A-65A: 1.70 kcal/mol, C: 4.50 Å, C: 4.59 Å, Tors: 231.53
Z
X
Y
2q7q 2q7q_75D_140D: 75D-140D: 0.49 kcal/mol, C: 5.50 Å, Tors: 145.62
Y
X
Z

Figure 1: CPK & BS Disulfide Rendering

DisulfideLoader

This class encapsulates the disulfide database itself and is its primary means of accession. Instantiation takes 2 parameters: **subset** and **verbose**. Given the size of the database, one can use the **subset** parameter to load the first 1000 disulfides into memory. This facilitates quicker development and testing new functions. I recommend using a machine with 16GB or more to work with the full dataset.

The entirety of the RCSB disulfide database is stored within the class via a DisulfideList, a **Pandas** .csv file, and a **dict** of indices mapping the RCSB IDs into their respective list of disulfide bond objects. The datastructures allow simple, direct and flexible access to the disulfide structures contained within. This makes it possible to access the disulfides by array index, RCSB structure ID or disulfide name.

Example:

```
import proteusPy
from proteusPy import Disulfide, DisulfideLoader, DisulfideList

SS1 = DisulfideList([],'tmp1')
SS2 = DisulfideList([],'tmp2')

PDB_SS = DisulfideLoader(verbose=False, subset=True)

# Accessing by index value:
SS1 = PDB_SS[0]
SS1
<Disulfide 4yys_22A_65A, Source: 4yys, Resolution: 1.35 Å>

# Accessing by PDB_ID returns a list of Disulfides:
SS2 = PDB_SS['4yys']
SS2
[<Disulfide 4yys_22A_65A, Source: 4yys, Resolution: 1.35 Å>,
<Disulfide 4yys_56A_98A, Source: 4yys, Resolution: 1.35 Å>,
<Disulfide 4yys_156A_207A, Source: 4yys, Resolution: 1.35 Å>]

```

```
# Accessing individual disulfides by their name:
SS3 = PDB_SS['4yys_56A_98A']
SS3
<Disulfide 4yys_56A_98A, Source: 4yys, Resolution: 1.35 Å>

# Finally, we can access disulfides by regular slicing:
SSlist = SS2[:2]
[<Disulfide 4yys_56A_98A, Source: 4yys, Resolution: 1.35 Å>,
<Disulfide 4yys_156A_207A, Source: 4yys, Resolution: 1.35 Å>]
```

The class can also render Disulfides overlaid on a common coordinate system to a pyVista window using the DisulfideLoader.display_overlay() method.

NB: For typical usage one accesses the database via the **Load_PDB_SS()** function. This function loads the compressed database from its single source. Initializing a **DisulfideLoader** object will load the individual torsions and disulfide **.pkl** files, builds the classlist structures, and writes the completely built object to a single **.pkl** file. This requires the raw **.pkl** files created by the download process. These files are contained in the *repository* **data** directory, not in the **pyPi** distribution.

turtle3D

The **turtle3D** class represents an object that maintains a *local coordinate system* in three dimensional space. This coordinate system consists of:

- A Position in 3D space
- A Heading Vector
- A Left Vector
- An Up Vector

The *Heading*, *Left* and *Up* vectors are unit vectors that define the object's orientation in a *local* coordinate frame. The turtle developed in **proteusPy** is based on the excellent book by Abelson: (Abelson & DiSessa, 1986). The to_local and to_global methods convert between these two coordinate systems. These methods make it possible to readily compare different disulfides by:

1. Orienting the turtle at the disulfide's proximal residue in a standard orientation.
2. Converting the global coordinates of the disulfide as read from the RCSB into local coordinates.
3. Saving all of the local coordinates with the raw coordinates
4. Performing distance and angle calculations

By implementing the functions **Move**, **Roll**, **Yaw**, **Pitch** and **Turn** the turtle is capable of movement in a three-dimensional space. See (Pabo & Suchanek, 1986) for more details.

The turtle has several molecule-specific functions including orient_at_residue and orient_from_backbone. These routines make it possible to build protein backbones of arbitrary conformation and to readily add sidechains to modeled structures. These functions are currently used to build model disulfides from dihedral angle input.

Examples

I illustrate a few use cases for the package below. Use the **jupyter lab** command from your shell to launch jupyter. The examples illustrate the ease with which one can analyze and visualize some disulfides.

Find the lowest and highest energy disulfides present in the database

```python
from proteusPy import Load_PDB_SS, DisulfideList, Disulfide

# load the database
PDB_SS = Load_PDB_SS(verbose=True, subset=False)

# retrieve the minimum and maximum energy structures
ssMin, ssMax = PDB_SS.SSList.minmax_energy

# make a list to hold them
minmaxlist = DisulfideList([ssMin, ssMax], "minmax")

# display them as ball and stick style
minmaxlist.display(style="bs", light=True)
```

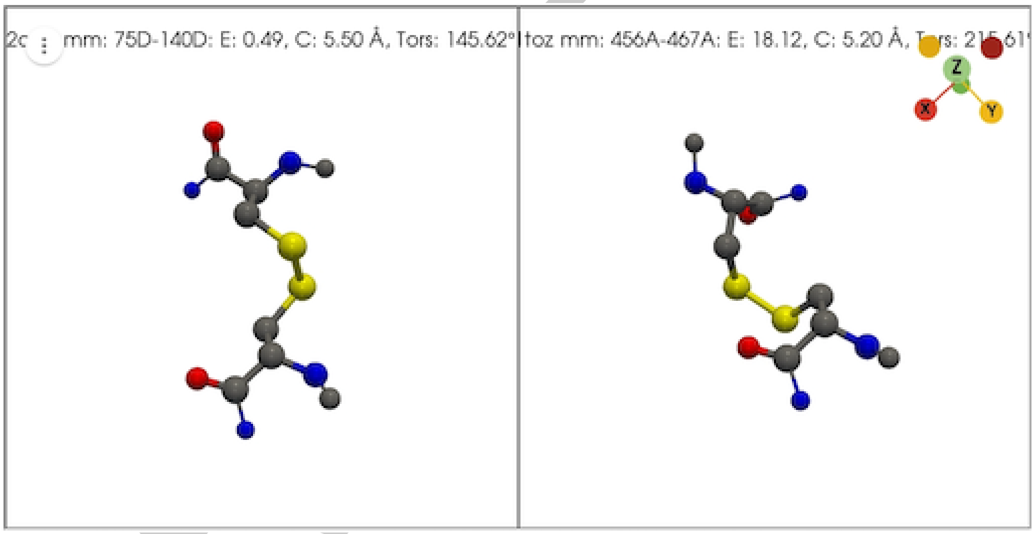
2c mm: 75D-140D: E: 0.49, C: 5.50 Å, Tors: 145.62°
ltoz mm: 456A-467A: E: 18.12, C: 5.20 Å, Tors: 215.61°
Z
X
Y

Figure 2: minmax

Find disulfides within 10 Å RMS in torsional space of the lowest energy structure

In this example we load the disulfide database, find the disulfides with the lowest and highest energies, and then find the nearest conformational neighbors. Finally, we display the neighbors overlaid against a common reference frame. Note that the window title gives statistics about the list of disulfides being displayed, including list name, resolution, number, average energy, and average atom positional error.

```python
import proteusPy
from proteusPy Load_PDB_SS, DisulfideList, Disulfide

PDB_SS = None
PDB_SS = Load_PDB_SS(verbose=False, subset=False)
ss_list = DisulfideList([], "tmp")

# Return the minimum and maximum energy structures. We ignore the maximum in this case.
ssmin_enrg, _ = PDB_SS.SSList.minmax_energy

# Make an empty list and find the nearest neighbors within 10 degrees avg RMS in
# sidechain dihedral angle space.

low_energy_neighbors = DisulfideList([], "Neighbors")
low_energy_neighbors = ssmin_enrg.Torsion_neighbors(sslist, 10)

# Display the number found, and then display them overlaid onto their common reference f

tot = low_energy_neighbors.length
low_energy_neighbors.display_overlay()
```

```
18
```

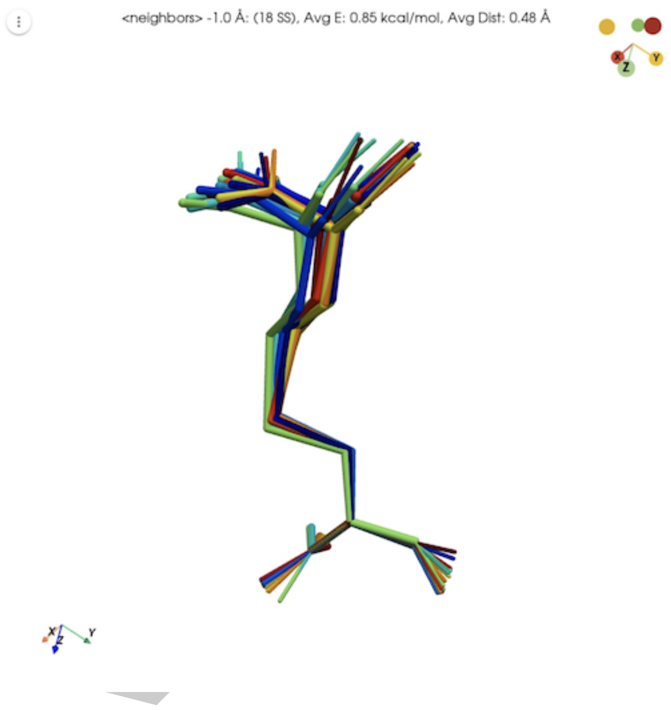
<neighbors> -1.0 Å: (18 SS), Avg E: 0.85 kcal/mol, Avg Dist: 0.48 Å
X
Y
Z

Figure 3: Low energy neighbors

Analyzing Disulfide Structural Class Distributions

The package includes the DisulfideClassConstructer class, which is used to create and manage Disulfide binary and sextant classes. A note about these structural classes is in order. (Schmidt, 2006) described a method of characterizing disulfide structures by describing each individual dihedral angle as either + or - based on its sign. This yields 2^5 or 32 possible classes. The author was then able to classify protein functional families into one of 20 remaining structural classes. Since the binary approach is very coarse and computers are much more capable than in 2006 I extended this formalism to a *Sextant* approach. In other words, I created *six* possible classes for each dihedral angle by dividing it into 60 degree segments. This yields a possible 6^5 or 7,776 possible classes. The notebook DisulfideClassesPlayground.ipynb contains some initial results. This work is ongoing.

Appendix

Database Creation Workflow

The following steps were performed to create the RCSB disulfide database:

1. Identify disulfide containing proteins in the RCSB: I generated a query using the web-based query tool for all proteins containing one or more disulfide bond. The resulting file consisted of 35,819 IDs. The file containing these is: ss_ids.txt.
2. Download the structure files to disk. This resulted in the program DisulfideDownloader.py. The download took approximately twelve hours.
3. Extract the disulfides from the downloaded structures and build the **DisulfideLoader** object. The program DisulfideExtractor.py was created and used to do this against the individual structure files. This seemingly simple task was complicated by several factors including:
    1. The PDB file parser contained in Bio.PDB described in (Hamelyrck & Manderick, 2003) lacked the ability to parse the **SSBOND** records in PDB files. As a result I forked the Biopython repository and updated the **parse_pdb_header.py** file. My fork is available at: https://github.com/suchanek/biopython
    2. Duplicate disulfides contained within a multi-chain protein file.
    3. Physically impossible disulfides, where the $C_\alpha - C_\alpha$ distance is $>$ 8 Å .
    4. Structures with disordered CYS atoms.

The disulfide extraction process is time consuming, and is only needed if the underlying **Disulfide** class is changed.

I ultimately elected to only use a single example of a given disulfide from a multi-chain entry, and removed any disulfides with a $C_\alpha - C_\alpha$ distance $>$ 8 Å. This resulted in the current database consisting of 35,808 structures and 120,494 disulfide bonds. While there are many structure visualization and analysis packages available (PyMol, Chimera, RCSB) this is the only centralized, locally available Disulfide database available.

The Future

- I am writing up the first analysis paper which will provide an overall survey of the RCSB disulfide database in terms of structural statistics. This represents the outcome from my initial desire for building the system in the first place.
- I am exploring disulfide structural classes using the sextant class approach as time permits. This offers much higher class resolution than the binary approach and reveals subgroups within the binary structural classes. I'd also like to explore the catalytic and allosteric classes within the subgroups to look for common structural features at a higher level.
- I am working to deploy a Disulfide Database browser for further exploration and analysis. There are several iterations of the viewer in the **programs** directory. The issue is I am unable to refresh a **panel pyvista.plotter** object correctly into a single pane.

Miscellaneous

Performance

- Manipulating and searching through long lists of disulfides can take time. I've added progress bars for many of these operations.
- Rendering many disulfides in **pyvista** can also take time to load and may be slow to display in real time, depending on your hardware. I added optimization to reduce cylinder complexity as a function of total cylinders rendered, but it can still be less than perfect. The faster your GPU the better!

Visualizing Disulfides with pyVista

PyVista is an excellent 3D visualization framework and I've used it for the Disulfide visualization engine. It uses the VTK library on the back end and provides high-level access to 3d rendering.

The menu strip provided in the Disulfide visualization windows allows the user to turn borders, rulers, bounding boxes on and off and reset the orientations. Please try them out! There is also a button for *local* vs *server* rendering. *Local* rendering is usually much smoother. To manipulate: - Click and drag your mouse to rotate - Use the mouse wheel to zoom (3 finger zoom on trackpad)

Developer's Notes:

The .pkl files needed to instantiate this class and save it into its final .pkl file are defined in the proteusPy.data class and should not be changed. Upon initialization the class will load them and initialize itself.

NB: (Suchanek, 2023b) disulfide database creaton relies on my fork of the Biopython Python package to download and build the database, (https://github.com/suchanek/biopython). As a result, one can't download and create the database locally unless the BioPython patch is applied. Database analysis is unaffected without this, however.

Contributing/Reporting

I welcome anyone interested in collaborating on proteusPy! Feel free to contact me at suchanek@mac.com, fork the repo and get coding. Issues can be reported to https://github.com/suchanek/proteusPy/issues.

Bibliography

Abelson, H., & DiSessa, A. A. (1986). *Turtle geometry: The computer as a medium for exploring mathematics*. MIT Press.

Hamelyrck, T., & Manderick, B. (2003). PDB file parser and structure class implemented in python. *Bioinformatics*, *19*(17), 2308–2310. https://doi.org/10.1093/bioinformatics/btg2994

Pabo, C. O., & Suchanek, E. G. (1986). Computer-aided model-building strategies for protein design. *Biochemistry*, *25*(20), 5987–5991. https://doi.org/10.1021/bi00368a023

Schmidt, B. (2006). Multiple disulfide-bonded states of native proteins: Estimate of number using probabilities of disulfide bond formation. *Biochemistry*, *45*(24), 7429–74334. https://doi.org/10.1021/bi0603064

Suchanek, E. G. (2023a). proteusPy API. In *GitHub Documents*. GitHub. https://suchanek.github.io/suchanek/proteusPy/proteusPy.html

Suchanek, E. G. (2023b). proteusPy: A package for modeling and analyzing proteins of known structure. In *GitHub repository*. GitHub. https://github.com/suchanek/proteusPy

Wong, J. H., & Hogg, P. J. (2010). Analysis of disulfide bonds in protein structures. *Journal of Thrombosis and Haemostasis*, *8*(10), 2345. https://doi.org/10.1111/j.1538-7836.2010.03894.x