ssbio Documentation

Release 0.9.9

Nathan Mih

Dec 20, 2017

Contents

CHAPTER 1

Introduction

This Python package provides a collection of tools for people with questions in the realm of structural systems biology. The main goals of this package are to:

1. Provide an easy way to map genes to their encoded proteins sequences and structures
2. Directly link structures to genome-scale SBML models
3. Prepare structures for downstream analyses, such as their use in molecular modeling software
4. Demonstrate fully-featured Python scientific analysis environments in Jupyter notebooks

Example questions you can (start to) answer with this package:

- How can I determine the number of protein structures available for my list of genes?
- What is the best, representative structure for my protein?
- Where, in a metabolic network, do these proteins work?
- Where do popular mutations show up on a protein?
- How can I compare the structural features of entire proteomes?
- How can I zoom in and visualize the interactions happening in the cell at the molecular level?
- How do structural properties correlate with my experimental datasets?
- How can I improve the contents of my model with structural data?
- and more...

CHAPTER 2

Installation

First install NGLview using pip:

```
pip install nglview
```

Then install ssbio:

```
pip install ssbio
```

Updating

```
pip install ssbio --upgrade
```

Uninstalling

```
pip uninstall ssbio
```

2.1 Dependencies

See: Software Installations for additional programs to install. Most of these additional programs are used to predict or calculate properties of proteins.

CHAPTER 3

Tutorials

Check out some Jupyter notebook tutorials for a single Protein and or for many in a GEM-PRO model.

CHAPTER 4

Citation

The manuscript for the `ssbio` package can be found and cited at[1].

[1] Mih N, Brunk E, Chen K, Catoiu E, Sastry A, Kavvas E, Monk JM, Zhang Z, Palsson BO. ssbio: A Python Framework for Structural Systems Biology. bioRxiv. 2017. p. 165506. https://doi.org/10.1101/165506

CHAPTER 5

Table of Contents

5.1 Getting Started

5.1.1 Introduction

This section will give a quick outline of the design of *ssbio* and the scientific topics behind it. If you would like to read a pre-print version of the manuscript, please see[1].

5.1.2 The basics

ssbio was developed with simplicity in mind - we wanted to make it as easy as possible to work with protein sequences and structures. Furthermore, we didn't want to reinvent the wheel wherever possible, thus systems models are treated as a direct extension of COBRApy, and Biopython classes and modules are used wherever possible. To best explain the utility of the package, we will outline its features from 2 different viewpoints: as a systems biologist used to looking at the "big picture"; and as a structural biologist where the "devil is in the details".

From a systems perspective

Systems biology is broadly concerned with the modeling and understanding of complex biological systems. What you may be taught in biochemistry 101 at this level will usually be reflected in a kind of interaction map, such as metabolic map shown here:

This map details the reactions needed to sustain the metabolic function of a cell. Typically, nodes will represent enzymes, and edges the metabolites they act upon (this is reversed in some graphical representations). There can be hundreds or thousands of reactions being modeled at once, *in silico*. These models can be stored in a single file, such as the Systems Biology Markup Language (SBML). *ssbio* can load SBML models, and so far we have mainly used it in the further annotation of genome-scale metabolic models, or GEMs. The goal of GEMs is to provide a comprehensive annotation of all the metabolic enzymes encoded within a genome, along with a generating a computable model (such

[1] Mih N, Brunk E, Chen K, Catoiu E, Sastry A, Kavvas E, et al. ssbio: A Python Framework for Structural Systems Biology. bioRxiv. 2017. p. 165506. doi:10.1101/165506

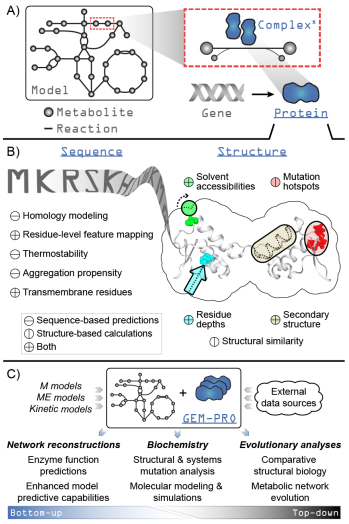


Fig. 5.1: Overview of the design and functionality of *ssbio*. Underlined fixed-width text in blue indicates added functionality to COBRApy for a genome-scale model loaded using *ssbio*. A) A simplified schematic showing the addition of a Protein to the core objects of COBRApy (fixed-width text in gray). A gene is directly associated with a protein, which can act as a monomeric enzyme or form an active complex with itself or other proteins (the asterisk denotes that methods for complexes are currently under development). B) Summary of properties and functions available for a protein sequence and structure. C) Uses of a GEM-PRO, from the bottom-up and the top-down. Once all protein sequences and structures are mapped to a genome-scale model, the resulting GEM-PRO has uses in multiple areas of study.

as at a steady state, using constraint-based modeling methods, a.k.a. COBRA). That brings us to our first class: the `GEMPRO` object.

The objectives of the GEM-PRO pipeline (genome-scale models integrated with protein structures) have previously been detailed[2]. A GEM-PRO directly integrates structural information within a curated GEM, and streamlines identifier mapping, representative object selection, and property calculation for a set of proteins. The pipeline provided in *ssbio* functions with an input of a GEM (or any other kind of network model that can be loaded with COBRApy), but if this is unfamiliar to you, do not fret! A GEM-PRO can be built simply from a list of gene/protein IDs, and can simply be treated as a way to easily analyze a large number of proteins at once.

See *GEMPRO* for a detailed explanation of this object, Jupyter notebook tutorials of prospective use cases, and an explanation of select functions.

From a structures perspective

Structural biology is broadly concerned with elucidating and understanding the structure and function of proteins and other macromolecules. Ribbons, molecules, and chemical interactions are the name of the game here:

An abundance of information is stored within structural data, and we believe that it should not be ignored even when looking at thousands of proteins at once within a systems model. To that end, the `Protein` object aims to integrate analyses on the level of a single protein's sequence (and related sequences) along with its available structures.

A `Protein` is representative of its associated gene's translated polypeptide chain (in other words, we are only considering monomers at this point). The object holds related amino acid sequences and structures, allowing for a single representative sequence and structure to be set from these. Multiple available structures such as those from PDB or homology models can be subjected to QC/QA based on set cutoffs such as sequence coverage and X-ray resolution.

[2] Brunk E, Mih N, Monk J, Zhang Z, O'Brien EJ, Bliven SE, et al. Systems biology of the structural proteome. BMC Syst Biol. 2016;10: 26. doi:10.1186/s12918-016-0271-6

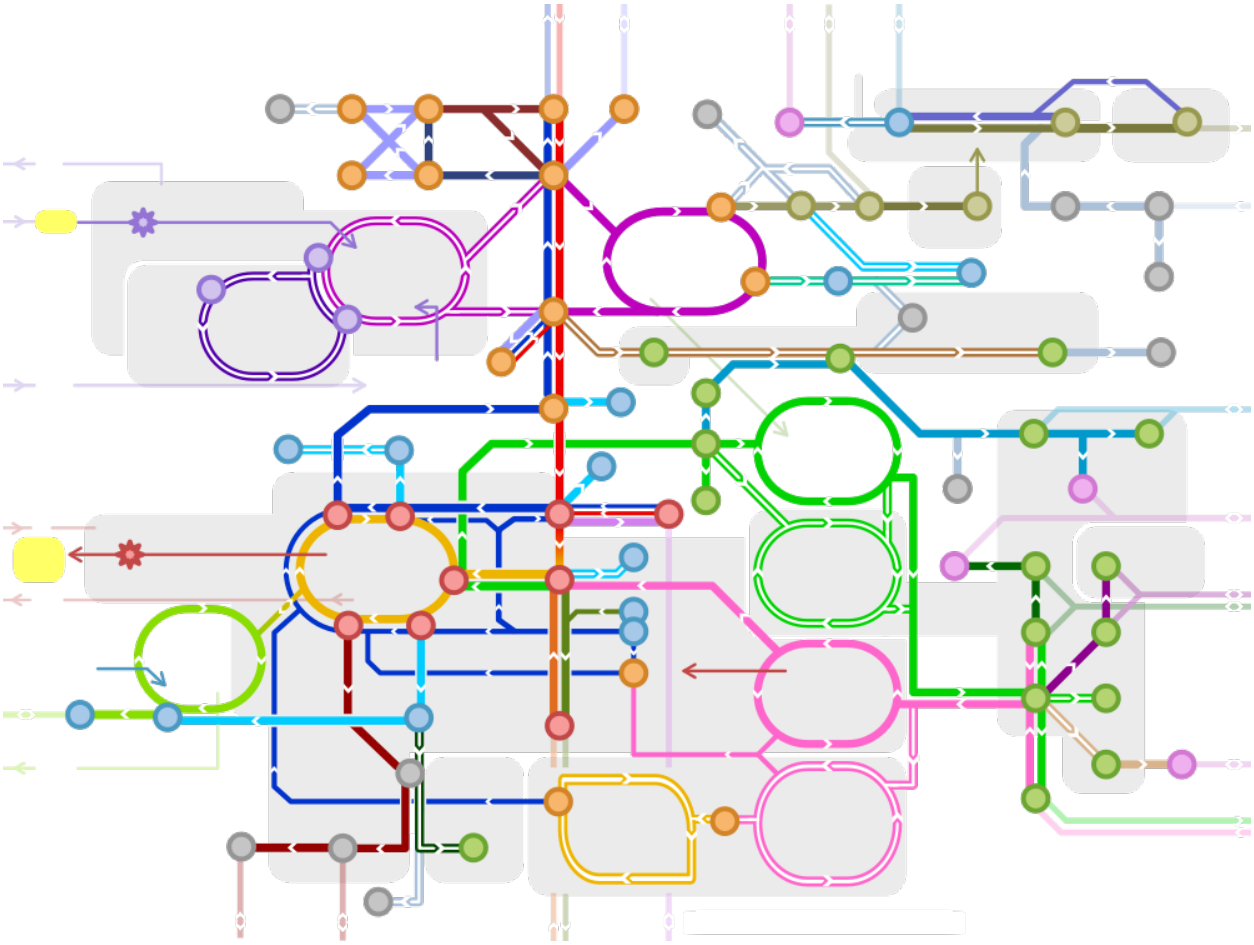


Fig. 5.2: A “metabolic metro map”. By Dctrzl, changed work of Chakazul [CC BY-SA 4.0], via Wikimedia Commons 1

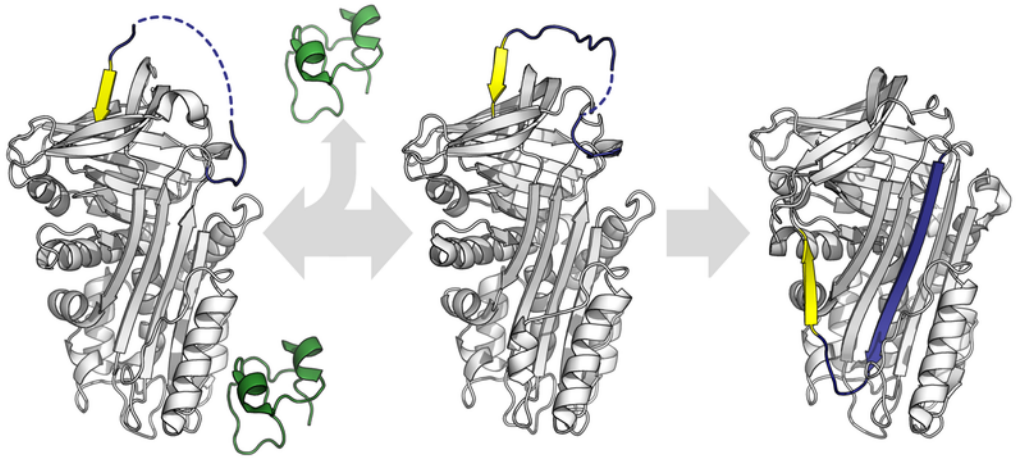


Fig. 5.3: A protein undergoing conformational changes. By Thomas Shafee (Own work) [CC BY 4.0], via Wikimedia Commons 2

Proteins with no structures available can be prepared for homology modeling through the I-TASSER platform. Biopython representations of sequences (`SeqRecord` objects) and structures (`Structure` objects) are utilized to allow access to analysis functions available for their respective objects.

See *Protein* for a detailed explanation of this object, Jupyter notebook tutorials of prospective use cases, and an explanation of select functions.

5.1.3 Modules & submodules

ssbio is organized into the following submodules for defined purposes. Please see the *Python API* for function documentation.

- `ssbio.databases`: modules that heavily depend on the Bioservices package[3] and custom code to enable pulling information from web services such as UniProt, KEGG, and the PDB, and to directly convert that information into sequence and structure objects to load into a protein.
- `ssbio.protein.sequence`: modules which allow a user to execute and parse sequence-based utilities such as sequence alignment algorithms or structural feature predictors.
- `ssbio.protein.structure`: modules that mirror the sequence module but instead work with structural information to calculate properties, and also to streamline the generation of homology models as well as to prepare structures for molecular modeling tools such as docking or molecular dynamics.
- `ssbio.pipeline.gempro`: a pipeline that simplifies the execution of these tools per protein while placing them into the context of a genome-scale model.

5.1.4 References

5.2 The GEM-PRO Pipeline

5.2.1 Introduction

The GEM-PRO pipeline is focused on annotating genome-scale models with protein structure information. Any SBML model can be used as input to the pipeline, although it is not required to have a one. Here are the possible starting points for using the pipeline:

1. An SBML model in `SBML` (`.sbml`, `.xml`), or `MATLAB` (`.mat`) formats
2. A list of gene IDs (`['b0001', 'b0002', ...]`)
3. A dictionary of gene IDs and their sequences (`{'b0001':'MSAVEVEEAP..', 'b0002':'AERAPLS', ...}`)

A GEM-PRO object can be thought of at a high-level as simply an annotation project. Creating a new project with any of the above starting points will create a new folder where protein sequences and structures will be downloaded to.

5.2.2 Tutorials

GEM-PRO - Calculating Protein Properties

This notebook gives an example of how to **calculate protein properties** for a list of proteins. The main features demonstrated are:

[3] Cokelaer, T, Pultz, D, Harder, LM, Serra-Musach, J, & Saez-Rodriguez, J. (2013). BioServices: a common Python package to access biological Web Services programmatically. Bioinformatics, 29/24: 3241–2. DOI: 10.1093/bioinformatics/btt547

1. Information retrieval from UniProt and linking residue numbering sites to structure
2. Calculating or predicting global protein sequence and structure properties
3. Calculating or predicting local protein sequence and structure properties

---

Input: List of gene IDs

Output: Representative protein structures and properties associated with them

---

Imports

```
In [1]: import sys
        import logging
```

```
In [2]: # Import the GEM-PRO class
        from ssbio.pipeline.gempro import GEMPRO
```

```
In [3]: # Printing multiple outputs per cell
        from IPython.core.interactiveshell import InteractiveShell
        InteractiveShell.ast_node_interactivity = "all"
```

Logging

Set the logging level in `logger.setLevel(logging.<LEVEL_HERE>)` to specify how verbose you want the pipeline to be. Debug is most verbose.

- `CRITICAL`
  - Only really important messages shown
- `ERROR`
  - Major errors
- `WARNING`
  - Warnings that don't affect running of the pipeline
- `INFO` (default)
  - Info such as the number of structures mapped per gene
- `DEBUG`
  - Really detailed information that will print out a lot of stuff

Warning: `DEBUG` mode prints out a large amount of information, especially if you have a lot of genes. This may stall your notebook!

```
In [4]: # Create logger
        logger = logging.getLogger()
        logger.setLevel(logging.INFO)  # SET YOUR LOGGING LEVEL HERE #
```

```
In [5]: # Other logger stuff for Jupyter notebooks
        handler = logging.StreamHandler(sys.stderr)
        formatter = logging.Formatter('[%(asctime)s] [%(name)s] %(levelname)s: %(message)s', datefmt=
        handler.setFormatter(formatter)
        logger.handlers = [handler]
```

Initialization

Set these three things:

- `ROOT_DIR`
  - The directory where a folder named after your `PROJECT` will be created
- `PROJECT`
  - Your project name
- `LIST_OF_GENES`
  - Your list of gene IDs

A directory will be created in `ROOT_DIR` with your `PROJECT` name. The folders are organized like so:

```
ROOT_DIR
└── PROJECT
    ├── data  # General storage for pipeline outputs
    ├── model  # SBML and GEM-PRO models are stored here
    ├── genes  # Per gene information
    │   ├── <gene_id1>  # Specific gene directory
    │   │   └── protein
    │   │       ├── sequences  # Protein sequence files, alignments, etc.
    │   │       └── structures  # Protein structure files, calculations, etc.
    │   └── <gene_id2>
    │       └── protein
    │           ├── sequences
    │           └── structures
    ├── reactions  # Per reaction information
    │   └── <reaction_id1>  # Specific reaction directory
    │       └── complex
    │           └── structures  # Protein complex files
    └── metabolites  # Per metabolite information
        └── <metabolite_id1>  # Specific metabolite directory
            └── chemical
                └── structures  # Metabolite 2D and 3D structure files
```

Note: Methods for protein complexes and metabolites are still in development.

```
In [6]: # SET FOLDERS AND DATA HERE
        import tempfile
        ROOT_DIR = tempfile.gettempdir()

        PROJECT = 'ssbio_protein_properties'
        LIST_OF_GENES = ['b1276', 'b0118']

In [7]: # Create the GEM-PRO project
        my_gempro = GEMPRO(gem_name=PROJECT, root_dir=ROOT_DIR, genes_list=LIST_OF_GENES, pdb_file_ty
```

```
[2017-11-21 19:21] [ssbio.pipeline.gempro] INFO: Creating GEM-PRO project directory in folder /tmp
[2017-11-21 19:21] [ssbio.pipeline.gempro] INFO: /tmp/ssbio_protein_properties: GEM-PRO project locat
[2017-11-21 19:21] [ssbio.pipeline.gempro] INFO: 2: number of genes
```

Mapping gene ID –> sequence

First, we need to map these IDs to their protein sequences. There are 2 ID mapping services provided to do this - through **KEGG** or **UniProt**. The end goal is to map a UniProt ID to each ID, since there is a comprehensive mapping (and some useful APIs) between UniProt and the PDB.

Note: You only need to map gene IDs using one service. However you can run both if some genes don't map in one service and do map in another!

.. automethod:: ssbio.pipeline.gempro.GEMPRO.uniprot$_m apping_a nd_m etadata$: $noindex$:

```
In [8]: # UniProt mapping
        my_gempro.uniprot_mapping_and_metadata(model_gene_source='ENSEMBLGENOME_ID')
        print('Missing UniProt mapping: ', my_gempro.missing_uniprot_mapping)
        my_gempro.df_uniprot_metadata.head()
```

```
[2017-11-21 19:21] [root] INFO: getUserAgent: Begin
[2017-11-21 19:21] [root] INFO: getUserAgent: user_agent: EBI-Sample-Client/ (services.py; Python 3.6
[2017-11-21 19:21] [root] INFO: getUserAgent: End
```

```
HBox(children=(IntProgress(value=0, max=2), HTML(value='')))
```

```
[2017-11-21 19:21] [ssbio.pipeline.gempro] INFO: 2/2: number of genes mapped to UniProt
[2017-11-21 19:21] [ssbio.pipeline.gempro] INFO: Completed ID mapping --> UniProt. See the "df_unipro
```

```
Missing UniProt mapping:  []
```

```
Out[8]: uniprot  reviewed gene_name                 kegg  \
        gene
        b0118  P36683     False      acnB  ecj:JW0114;eco:b0118
        b1276  P25516     False      acnA  ecj:JW1268;eco:b1276

                                  refseq  pdbs                     pfam  \
        gene
        b0118  NP_414660.1;WP_001307570.1  1L5J  PF00330;PF06434;PF11791
        b1276  NP_415792.1;WP_000099535.1   NaN          PF00330;PF00694

                        description  entry_date  entry_version    seq_date  \
        gene
        b0118  Aconitate hydratase B  2017-10-25            163  1997-11-01
        b1276  Aconitate hydratase A  2017-10-25            151  2008-01-15

               seq_version sequence_file metadata_file
        gene
        b0118           3  P36683.fasta    P36683.xml
        b1276           3  P25516.fasta    P25516.xml
```

.. automethod:: ssbio.pipeline.gempro.GEMPRO.set$_r epresentative_s equence$: $noindex$:

```
In [9]: # Set representative sequences
        my_gempro.set_representative_sequence()
        print('Missing a representative sequence: ', my_gempro.missing_representative_sequence)
        my_gempro.df_representative_sequences.head()
```

```
HBox(children=(IntProgress(value=0, max=2), HTML(value='')))


[2017-11-21 19:21] [ssbio.pipeline.gempro] INFO: 2/2: number of genes with a representative sequence
[2017-11-21 19:21] [ssbio.pipeline.gempro] INFO: See the "df_representative_sequences" attribute for

Missing a representative sequence:  []
```

Out[9]:

	uniprot	kegg	pdbs	sequence_file	metadata_file
gene					
b0118	P36683	ecj:JW0114;eco:b0118	1L5J	P36683.fasta	P36683.xml
b1276	P25516	ecj:JW1268;eco:b1276	NaN	P25516.fasta	P25516.xml

Mapping representative sequence –> structure

These are the ways to map sequence to structure:

1. Use the UniProt ID and their automatic mappings to the PDB
2. BLAST the sequence to the PDB
3. Make homology models or
4. Map to existing homology models

You can only utilize option #1 to map to PDBs if there is a mapped UniProt ID set in the representative sequence. If not, you'll have to BLAST your sequence to the PDB or make a homology model. You can also run both for maximum coverage.

.. automethod:: ssbio.pipeline.gempro.GEMPRO.map$_u niprot_t o_p db$: $noindex$:

```
In [10]: # Mapping using the PDBe best_structures service
         my_gempro.map_uniprot_to_pdb(seq_ident_cutoff=.3)
         my_gempro.df_pdb_ranking.head()

[2017-11-21 19:21] [ssbio.pipeline.gempro] INFO: Mapping UniProt IDs --> PDB IDs...
[2017-11-21 19:21] [root] INFO: getUserAgent: Begin
[2017-11-21 19:21] [root] INFO: getUserAgent: user_agent: EBI-Sample-Client/ (services.py; Python 3.6
[2017-11-21 19:21] [root] INFO: getUserAgent: End

HBox(children=(IntProgress(value=0, max=2), HTML(value='')))


[2017-11-21 19:21] [ssbio.pipeline.gempro] INFO: 1/2: number of genes with at least one experimental
[2017-11-21 19:21] [ssbio.pipeline.gempro] INFO: Completed UniProt --> best PDB mapping. See the "df_
```

Out[10]:

	pdb_id	pdb_chain_id	uniprot	experimental_method	resolution	coverage	\
gene							
b0118	1l5j	A	P36683	X-ray diffraction	2.4	1	
b0118	1l5j	B	P36683	X-ray diffraction	2.4	1	

	start	end	unp_start	unp_end	rank
gene					
b0118	1	865	1	865	1
b0118	1	865	1	865	2

.. automethod:: ssbio.pipeline.gempro.GEMPRO.blast$_s eqs_t o_p db$: $noindex$:

```
In [11]: # Mapping using BLAST
         my_gempro.blast_seqs_to_pdb(all_genes=True, seq_ident_cutoff=.7, evalue=0.00001)
         my_gempro.df_pdb_blast.head(2)

HBox(children=(IntProgress(value=0, max=2), HTML(value='')))
```

```
[2017-11-21 19:21] [ssbio.pipeline.gempro] INFO: Completed sequence --> PDB BLAST. See the "df_pdb_bl
[2017-11-21 19:21] [ssbio.pipeline.gempro] INFO: 0: number of genes with additional structures added
[2017-11-21 19:21] [ssbio.pipeline.gempro] WARNING: Empty dataframe
```

```
Out[11]: Empty DataFrame
         Columns: []
         Index: []
```

.. automethod:: ssbio.pipeline.gempro.GEMPRO.get$_m anual_h omology_m odels$: $noindex$:

```
In [12]: import pandas as pd
         import os.path as op
```

```
In [13]: # Creating manual mapping dictionary for ECOLI I-TASSER models
         homology_models = '/home/nathan/projects_archive/homology_models/ECOLI/zhang/'
         homology_models_df = pd.read_csv('/home/nathan/projects_archive/homology_models/ECOLI/zhang_
         tmp = homology_models_df[['zhang_id','model_file','m_gene']].drop_duplicates()
         tmp = tmp[pd.notnull(tmp.m_gene)]

         homology_model_dict = {}

         for i,r in tmp.iterrows():
             homology_model_dict[r['m_gene']] = {r['zhang_id']: {'model_file':op.join(homology_models
                                                                 'file_type':'pdb'}}

         my_gempro.get_manual_homology_models(homology_model_dict)
```

```
HBox(children=(IntProgress(value=0, max=2), HTML(value='')))
```

```
[2017-11-21 19:21] [ssbio.pipeline.gempro] INFO: Updated homology model information for 2 genes.
```

```
In [14]: # Creating manual mapping dictionary for ECOLI SUNPRO models
         homology_models = '/home/nathan/projects_archive/homology_models/ECOLI/sunpro/'
         homology_models_df = pd.read_csv('/home/nathan/projects_archive/homology_models/ECOLI/sunpro
         tmp = homology_models_df[['sunpro_id','model_file','m_gene']].drop_duplicates()
         tmp = tmp[pd.notnull(tmp.m_gene)]

         homology_model_dict = {}

         for i,r in tmp.iterrows():
             homology_model_dict[r['m_gene']] = {r['sunpro_id']: {'model_file':op.join(homology_model
                                                                 'file_type':'pdb'}}

         my_gempro.get_manual_homology_models(homology_model_dict)
```

```
HBox(children=(IntProgress(value=0, max=2), HTML(value='')))
```

```
[2017-11-21 19:21] [ssbio.pipeline.gempro] INFO: Updated homology model information for 2 genes.
```

Downloading and ranking structures

.. automethod:: ssbio.pipeline.gempro.GEMPRO.pdb$_d ownloader_a nd_m etadata$: $noindex$:

Warning: Downloading all PDBs takes a while, since they are also parsed for metadata. You can skip this step and just set representative structures below if you want to minimize the number of PDBs downloaded.

```
In [15]: # Download all mapped PDBs and gather the metadata
         my_gempro.pdb_downloader_and_metadata()
         my_gempro.df_pdb_metadata.head(2)
```

```
HBox(children=(IntProgress(value=0, max=2), HTML(value='')))
```

```
[2017-11-21 19:21] [ssbio.pipeline.gempro] INFO: Updated PDB metadata dataframe. See the "df_pdb_meta
[2017-11-21 19:21] [ssbio.pipeline.gempro] INFO: Saved 1 structures total
```

```
Out[15]: pdb_id                                  pdb_title  \
         gene
         b0118   1l5j  CRYSTAL STRUCTURE OF E. COLI ACONITASE B.

                                      description experimental_method mapped_chains  \
         gene
         b0118  Aconitate hydratase 2 (E.C.4.2.1.3)   X-ray diffraction           A;B

                resolution chemicals     taxonomy_name structure_file
         gene
         b0118         2.4   F3S;TRA  Escherichia coli       1l5j.pdb
```

.. automethod:: ssbio.pipeline.gempro.GEMPRO.set$_r epresentative_s tructure$: $noindex$:

```
In [16]: # Set representative structures
         my_gempro.set_representative_structure()
         my_gempro.df_representative_structures.head()
```

```
HBox(children=(IntProgress(value=0, max=2), HTML(value='')))
```

```
[2017-11-21 19:21] [ssbio.pipeline.gempro] INFO: 2/2: number of genes with a representative structure
[2017-11-21 19:21] [ssbio.pipeline.gempro] INFO: See the "df_representative_structures" attribute for
```

```
Out[16]: id  is_experimental file_type  \
         gene
         b0118         REP-1l5j             True       pdb
         b1276  REP-ACON1_ECOLI            False       pdb

                                  structure_file
         gene
         b0118                       1l5j-A_clean.pdb
         b1276  ACON1_ECOLI_model1_clean-X_clean.pdb
```

Computing and storing protein properties

.. automethod:: ssbio.pipeline.gempro.GEMPRO.get$_s equence_p roperties$: $noindex$:

```
In [17]: # Requires EMBOSS "pepstats" program
         # See the ssbio wiki for more information: https://github.com/SBRG/ssbio/wiki/Software-Insta
         # Install using:
         # sudo apt-get install emboss
         my_gempro.get_sequence_properties()
```

```
HBox(children=(IntProgress(value=0, max=2), HTML(value='')))
```

.. automethod:: ssbio.pipeline.gempro.GEMPRO.get$_s cratch_p redictions$: $noindex$:

```
In [18]: # Requires SCRATCH installation, replace path_to_scratch with own path to script
         # See the ssbio wiki for more information: https://github.com/SBRG/ssbio/wiki/Software-Insta
         my_gempro.get_scratch_predictions(path_to_scratch='/home/nathan/software/SCRATCH-1D_1.1/bin/
                                           results_dir=my_gempro.data_dir,
                                           num_cores=4)
```

```
[2017-11-21 19:21] [ssbio.pipeline.gempro] INFO: /tmp/ssbio_protein_properties/data/ssbio_protein_pro
```

```
###################################
#                                 #
#  SCRATCH-1D release 1.1 (2015)  #
#                                 #
###################################

[SCRATCH-1D_predictions.pl] 2 protein sequence(s) found
[SCRATCH-1D_predictions.pl] generating sequence profiles...
[SCRATCH-1D_predictions.pl] running SCRATCH-1D predictors...
[SCRATCH-1D_predictions.pl] running homology analysis...
[SCRATCH-1D_predictions.pl] writing SSpro predictions...
[SCRATCH-1D_predictions.pl] writing SSpro8 predictions...
[SCRATCH-1D_predictions.pl] writing ACCpro predictions...
[SCRATCH-1D_predictions.pl] writing ACCpro20 predictions...
[SCRATCH-1D_predictions.pl] job successfully completed!

HBox(children=(IntProgress(value=0, max=2), HTML(value='')))


[2017-11-21 19:30] [ssbio.pipeline.gempro] INFO: 2/2: number of genes with SCRATCH predictions loaded
```

.. automethod:: ssbio.pipeline.gempro.GEMPRO.find$_d isul fide_b ridges$: $noindex$:

```
In [19]: my_gempro.find_disulfide_bridges(representatives_only=False)
```

```
HBox(children=(IntProgress(value=0, max=2), HTML(value='')))
```

.. automethod:: ssbio.pipeline.gempro.GEMPRO.get$_d ssp_a nnotations$: $noindex$:

```
In [20]: # Requires DSSP installation
         # See the ssbio wiki for more information: https://github.com/SBRG/ssbio/wiki/Software-Insta
         my_gempro.get_dssp_annotations()
```

```
HBox(children=(IntProgress(value=0, max=2), HTML(value='')))
```

.. automethod:: ssbio.pipeline.gempro.GEMPRO.get$_m sms_a nnotations$: $noindex$:

```
In [21]: # Requires MSMS installation
         # See the ssbio wiki for more information: https://github.com/SBRG/ssbio/wiki/Software-Insta
         my_gempro.get_msms_annotations()
```

```
HBox(children=(IntProgress(value=0, max=2), HTML(value='')))
```

Global protein properties

Properties of the entire protein sequence/structure are stored in the `representative_sequence` and `representative_structure` attributes. These properties describe aspects of the entire protein, such as its molecular weight, the percentage of amino acids in a particular secondary structure, the percentage of charged or

```
In [22]: from pprint import pprint

In [23]: for g in my_gempro.genes_with_a_representative_structure:
             repseq = g.protein.representative_sequence
             repstruct = g.protein.representative_structure
             repchain = g.protein.representative_chain

             print('Gene: {}'.format(g.id))
             print('Number of structures: {}'.format(g.protein.num_structures))
             print('Representative sequence: {}'.format(repseq.id))
             print('Representative structure: {}'.format(repstruct.id))

             print('Global properties of the representative sequence:')
             pprint(repseq.annotations)

             print('Global properties of the representative structure:')
             pprint(repstruct.chains.get_by_id(repchain).seq_record.annotations)

             print('--------------------------------------------------')
```

```
Gene: b0118
Number of structures: 4
Representative sequence: P36683
Representative structure: REP-115j
Global properties of the representative sequence:
{'amino_acids_percent': {'A': 0.11213872832369942,
                         'C': 0.011560693641618497,
                         'D': 0.06358381502890173,
                         'E': 0.06589595375722543,
                         'F': 0.03352601156069364,
                         'G': 0.08786127167630058,
                         'H': 0.017341040462427744,
                         'I': 0.05433526011560694,
                         'K': 0.056647398843930635,
                         'L': 0.10173410404624278,
                         'M': 0.026589595375722544,
                         'N': 0.035838150289017344,
                         'P': 0.06242774566473988,
                         'Q': 0.028901734104046242,
                         'R': 0.04508670520231214,
                         'S': 0.04161849710982659,
                         'T': 0.05433526011560694,
                         'V': 0.06705202312138728,
                         'W': 0.009248554913294798,
                         'Y': 0.024277456647398842},
 'aromaticity': 0.06705202312138728,
 'instability_index': 32.79631213872841,
 'isoelectric_point': 5.23931884765625,
 'molecular_weight': 93497.01500000065,
 'monoisotopic': False,
 'percent_B-sspro8': 0.016184971098265895,
 'percent_C-sspro': 0.4254335260115607,
 'percent_C-sspro8': 0.2138728323699422,
```

```
 'percent_E-sspro': 0.15606936416184972,
 'percent_E-sspro8': 0.14335260115606938,
 'percent_G-sspro8': 0.027745664739884393,
 'percent_H-sspro': 0.4184971098265896,
 'percent_H-sspro8': 0.3895953757225434,
 'percent_I-sspro8': 0.0,
 'percent_S-sspro8': 0.07976878612716763,
 'percent_T-sspro8': 0.12947976878612716,
 'percent_acidic': 0.12948,
 'percent_aliphatic': 0.33526000000000006,
 'percent_aromatic': 0.08439,
 'percent_basic': 0.11907999999999999,
 'percent_buried-accpro': 0.6323699421965318,
 'percent_buried-accpro20': 0.6809248554913295,
 'percent_charged': 0.24855,
 'percent_exposed-accpro': 0.3676300578034682,
 'percent_exposed-accpro20': 0.3190751445086705,
 'percent_helix_naive': 0.29017341040462424,
 'percent_non-polar': 0.59075,
 'percent_polar': 0.40924999999999995,
 'percent_small': 0.53642,
 'percent_strand_naive': 0.3063583815028902,
 'percent_tiny': 0.30751,
 'percent_turn_naive': 0.22774566473988442}
Global properties of the representative structure:
{'percent_B-dssp': 0.016241299303944315,
 'percent_C-dssp': 0.20765661252900233,
 'percent_E-dssp': 0.14037122969837587,
 'percent_G-dssp': 0.034802784222737818,
 'percent_H-dssp': 0.38051044083526681,
 'percent_I-dssp': 0.0,
 'percent_S-dssp': 0.082366589327146175,
 'percent_T-dssp': 0.13805104408352667}
--------------------------------------------------
Gene: b1276
Number of structures: 3
Representative sequence: P25516
Representative structure: REP-ACON1_ECOLI
Global properties of the representative sequence:
{'amino_acids_percent': {'A': 0.08641975308641975,
                         'C': 0.007856341189674524,
                         'D': 0.06397306397306397,
                         'E': 0.06172839506172839,
                         'F': 0.025813692480359147,
                         'G': 0.08754208754208755,
                         'H': 0.020202020202020204,
                         'I': 0.04826038159371493,
                         'K': 0.04826038159371493,
                         'L': 0.09427609427609428,
                         'M': 0.028058361391694726,
                         'N': 0.037037037037037035,
                         'P': 0.05611672278338945,
                         'Q': 0.030303030303030304,
                         'R': 0.05723905723905724,
                         'S': 0.05723905723905724,
                         'T': 0.06060606060606061,
                         'V': 0.0819304152637486,
                         'W': 0.014590347923681257,
                         'Y': 0.03254769921436588},
```

```
 'aromaticity': 0.07295173961840629,
 'instability_index': 36.28239057239071,
 'isoelectric_point': 5.59344482421875,
 'molecular_weight': 97676.06830000057,
 'monoisotopic': False,
 'percent_B-sspro8': 0.010101010101010102,
 'percent_C-sspro': 0.43546576879910215,
 'percent_C-sspro8': 0.23905723905723905,
 'percent_E-sspro': 0.18855218855218855,
 'percent_E-sspro8': 0.1829405162738496,
 'percent_G-sspro8': 0.03254769921436588,
 'percent_H-sspro': 0.3759820426487093,
 'percent_H-sspro8': 0.3378226711560045,
 'percent_I-sspro8': 0.002244668911335578,
 'percent_S-sspro8': 0.0707070707070707,
 'percent_T-sspro8': 0.12457912457912458,
 'percent_acidic': 0.1257,
 'percent_aliphatic': 0.31089,
 'percent_aromatic': 0.09315,
 'percent_basic': 0.1257,
 'percent_buried-accpro': 0.5847362514029181,
 'percent_buried-accpro20': 0.6273849607182941,
 'percent_charged': 0.2514,
 'percent_exposed-accpro': 0.4152637485970819,
 'percent_exposed-accpro20': 0.372615039281706,
 'percent_helix_naive': 0.29741863075196406,
 'percent_non-polar': 0.56341,
 'percent_polar': 0.43659,
 'percent_small': 0.53872,
 'percent_strand_naive': 0.27048260381593714,
 'percent_tiny': 0.29966000000000004,
 'percent_turn_naive': 0.2379349046015713}
Global properties of the representative structure:
{'percent_B-dssp': 0.010101010101010102,
 'percent_C-dssp': 0.22222222222222221,
 'percent_E-dssp': 0.17396184062850731,
 'percent_G-dssp': 0.039281705948372617,
 'percent_H-dssp': 0.34567901234567899,
 'percent_I-dssp': 0.0056116722783389446,
 'percent_S-dssp': 0.094276094276094277,
 'percent_T-dssp': 0.10886644219977554}
--------------------------------------------------
```

Local protein properties

```
In [24]: [x for x in g.protein.representative_sequence.features if 'site' in x.type]
```

```
Out[24]: [SeqFeature(FeatureLocation(ExactPosition(434), ExactPosition(435)), type='metal ion-binding
          SeqFeature(FeatureLocation(ExactPosition(500), ExactPosition(501)), type='metal ion-binding
          SeqFeature(FeatureLocation(ExactPosition(503), ExactPosition(504)), type='metal ion-binding
```

```
In [25]: for g in my_gempro.genes:
             for f in g.protein.representative_sequence.features:
                 if 'site' in f.type.lower():
                     print(f)
```

```
type: metal ion-binding site
location: [709:710]
qualifiers:
    Key: description, Value: Iron-sulfur (4Fe-4S)
    Key: evidence, Value: 5
    Key: type, Value: metal ion-binding site

type: metal ion-binding site
location: [768:769]
qualifiers:
    Key: description, Value: Iron-sulfur (4Fe-4S)
    Key: evidence, Value: 5
    Key: type, Value: metal ion-binding site

type: metal ion-binding site
location: [771:772]
qualifiers:
    Key: description, Value: Iron-sulfur (4Fe-4S)
    Key: evidence, Value: 5
    Key: type, Value: metal ion-binding site

type: binding site
location: [190:191]
qualifiers:
    Key: description, Value: Substrate
    Key: evidence, Value: 5
    Key: type, Value: binding site

type: binding site
location: [497:498]
qualifiers:
    Key: description, Value: Substrate
    Key: evidence, Value: 5
    Key: type, Value: binding site

type: binding site
location: [790:791]
qualifiers:
    Key: description, Value: Substrate
    Key: evidence, Value: 5
    Key: type, Value: binding site

type: binding site
location: [795:796]
qualifiers:
    Key: description, Value: Substrate
    Key: evidence, Value: 5
    Key: type, Value: binding site

type: mutagenesis site
location: [768:769]
qualifiers:
    Key: description, Value: Inhibits the dimer formation.
    Key: evidence, Value: 7
    Key: original, Value: C
    Key: type, Value: mutagenesis site
    Key: variation, Value: S

type: metal ion-binding site
```

```
location: [434:435]
qualifiers:
    Key: description, Value: Iron-sulfur (4Fe-4S)
    Key: evidence, Value: 1
    Key: type, Value: metal ion-binding site

type: metal ion-binding site
location: [500:501]
qualifiers:
    Key: description, Value: Iron-sulfur (4Fe-4S)
    Key: evidence, Value: 1
    Key: type, Value: metal ion-binding site

type: metal ion-binding site
location: [503:504]
qualifiers:
    Key: description, Value: Iron-sulfur (4Fe-4S)
    Key: evidence, Value: 1
    Key: type, Value: metal ion-binding site
```

.. automethod:: ssbio.core.protein.Protein.get_residue_annotations :noindex:

```
In [26]: metal_info = []

         for g in my_gempro.genes:
             for f in g.protein.representative_sequence.features:
                 if 'metal' in f.type.lower():
                     res_info = g.protein.get_residue_annotations(f.location.end, use_representatives
                     res_info['gene_id'] = g.id
                     res_info['seq_id'] = g.protein.representative_sequence.id
                     res_info['struct_id'] = g.protein.representative_structure.id
                     res_info['chain_id'] = g.protein.representative_chain
                     metal_info.append(res_info)

         cols = ['gene_id', 'seq_id', 'struct_id', 'chain_id',
                 'seq_residue', 'seq_resnum', 'struct_residue','struct_resnum',
                 'seq_SS-sspro','seq_SS-sspro8','seq_RSA-accpro','seq_RSA-accpro20',
                 'struct_SS-dssp','struct_RSA-dssp', 'struct_ASA-dssp',
                 'struct_PHI-dssp', 'struct_PSI-dssp', 'struct_CA_DEPTH-msms', 'struct_RES_DEPTH-msms

         pd.DataFrame.from_records(metal_info, columns=cols).set_index(['gene_id', 'seq_id', 'struct_
```

```
Out[26]: seq_residue struct_residue  \
         gene_id seq_id struct_id       chain_id seq_resnum
         b0118   P36683 REP-115j        A        710                 C             C
                                                 769                 C             C
                                                 772                 C             C
         b1276   P25516 REP-ACON1_ECOLI X        435                 C             C
                                                 501                 C             C
                                                 504                 C             C

                                                              struct_resnum  \
         gene_id seq_id struct_id       chain_id seq_resnum
         b0118   P36683 REP-115j        A        710                   710
                                                 769                   769
                                                 772                   772
         b1276   P25516 REP-ACON1_ECOLI X        435                   435
                                                 501                   501
                                                 504                   504
```

```
                                                  seq_SS-sspro seq_SS-sspro8  \
gene_id seq_id struct_id       chain_id seq_resnum
b0118   P36683 REP-1i5j        A        710                   C             T
                                        769                   C             C
                                        772                   H             G
b1276   P25516 REP-ACON1_ECOLI X        435                   H             H
                                        501                   C             C
                                        504                   H             G

                                                 seq_RSA-accpro  \
gene_id seq_id struct_id       chain_id seq_resnum
b0118   P36683 REP-1i5j        A        710                     -
                                        769                     -
                                        772                     -
b1276   P25516 REP-ACON1_ECOLI X        435                     -
                                        501                     -
                                        504                     -

                                                  seq_RSA-accpro20  \
gene_id seq_id struct_id       chain_id seq_resnum
b0118   P36683 REP-1i5j        A        710                    10
                                        769                     5
                                        772                     5
b1276   P25516 REP-ACON1_ECOLI X        435                    10
                                        501                     0
                                        504                     0

                                                 struct_SS-dssp  \
gene_id seq_id struct_id       chain_id seq_resnum
b0118   P36683 REP-1i5j        A        710                     T
                                        769                     -
                                        772                     G
b1276   P25516 REP-ACON1_ECOLI X        435                     H
                                        501                     S
                                        504                     G

                                                 struct_RSA-dssp  \
gene_id seq_id struct_id       chain_id seq_resnum
b0118   P36683 REP-1i5j        A        710               0.118519
                                        769               0.088889
                                        772               0.081481
b1276   P25516 REP-ACON1_ECOLI X        435               0.059259
                                        501               0.088889
                                        504               0.259259

                                                 struct_ASA-dssp  \
gene_id seq_id struct_id       chain_id seq_resnum
b0118   P36683 REP-1i5j        A        710                  16.0
                                        769                  12.0
                                        772                  11.0
b1276   P25516 REP-ACON1_ECOLI X        435                   8.0
                                        501                  12.0
                                        504                  35.0

                                                 struct_PHI-dssp  \
gene_id seq_id struct_id       chain_id seq_resnum
b0118   P36683 REP-1i5j        A        710                 -67.1
                                        769                 -67.8
```

```
                                          772                   -50.2
b1276   P25516 REP-ACON1_ECOLI X        435                   -61.1
                                          501                   -61.0
                                          504                   -56.0

                                                          struct_PSI-dssp  \
gene_id seq_id struct_id       chain_id seq_resnum
b0118   P36683 REP-115j        A        710                    -7.2
                                          769                   -28.3
                                          772                   -38.0
b1276   P25516 REP-ACON1_ECOLI X        435                   -26.6
                                          501                   -50.0
                                          504                   -45.6

                                                          struct_CA_DEPTH-msms  \
gene_id seq_id struct_id       chain_id seq_resnum
b0118   P36683 REP-115j        A        710                     10.148960
                                          769                      8.296585
                                          772                      8.282292
b1276   P25516 REP-ACON1_ECOLI X        435                      2.656722
                                          501                      1.999713
                                          504                      1.999634

                                                          struct_RES_DEPTH-msms
gene_id seq_id struct_id       chain_id seq_resnum
b0118   P36683 REP-115j        A        710                      10.009109
                                          769                       8.049832
                                          772                       8.239369
b1276   P25516 REP-ACON1_ECOLI X        435                       2.813536
                                          501                       2.409119
                                          504                       1.961484
```

Visualizing residues

.. automethod:: ssbio.protein.structure.structprop.StructProp.view$_{s}tructure$: $noindex$:

.. automethod:: ssbio.protein.structure.structprop.StructProp.add$_{r}esidues_{h}ighlight_{t}o_{n}glview$: $noindex$:

```
In [27]: for g in my_gempro.genes:

             # Gather residue numbers
             metal_binding_structure_residues = []
             for f in g.protein.representative_sequence.features:
                 if 'metal' in f.type.lower():
                     res_info = g.protein.get_residue_annotations(f.location.end, use_representatives
                     metal_binding_structure_residues.append(res_info['struct_resnum'])
             print(metal_binding_structure_residues)

             # Display structure
             view = g.protein.representative_structure.view_structure()
             g.protein.representative_structure.add_residues_highlight_to_nglview(view=view, structu

             view
```

```
[710, 769, 772]
```

```
[2017-11-21 19:31] [ssbio.protein.structure.structprop] INFO: Selection: ( :A ) and not hydrogen and
```

```
NGLWidget()
```

```
[435, 501, 504]
[2017-11-21 19:31] [ssbio.protein.structure.structprop] INFO: Selection: ( :X ) and not hydrogen and
NGLWidget()
```

Comparing features in different structures of the same protein

```
In [28]: # Run all sequence to structure alignments
         for g in my_gempro.genes:
             for s in g.protein.structures:
                 g.protein.align_seqprop_to_structprop(seqprop=g.protein.representative_sequence, str

In [29]: metal_info_compared = []

         for g in my_gempro.genes:
             for f in g.protein.representative_sequence.features:
                 if 'metal' in f.type.lower():
                     for s in g.protein.structures:
                         for c in s.mapped_chains:
                             res_info = g.protein.get_residue_annotations(seq_resnum=f.location.end,
                                                                          seqprop=g.protein.represent
                                                                          structprop=s, chain_id=c,
                                                                          use_representatives=False)
                             res_info['gene_id'] = g.id
                             res_info['seq_id'] = g.protein.representative_sequence.id
                             res_info['struct_id'] = s.id
                             res_info['chain_id'] = c
                             metal_info_compared.append(res_info)

         cols = ['gene_id', 'seq_id', 'struct_id', 'chain_id',
                 'seq_residue', 'seq_resnum', 'struct_residue','struct_resnum',
                 'seq_SS-sspro','seq_SS-sspro8','seq_RSA-accpro','seq_RSA-accpro20',
                 'struct_SS-dssp','struct_RSA-dssp', 'struct_ASA-dssp',
                 'struct_PHI-dssp', 'struct_PSI-dssp', 'struct_CA_DEPTH-msms', 'struct_RES_DEPTH-msms

         pd.DataFrame.from_records(metal_info_compared, columns=cols).sort_values(by=['seq_resnum','s

Out[29]: chain_id struct_residue  \
         gene_id seq_id seq_resnum seq_residue struct_id
         b1276   P25516 435        C           ACON1_ECOLI           X             C
                                                E01201                X             C
                                                REP-ACON1_ECOLI       X             C
                        501        C           ACON1_ECOLI           X             C
                                                E01201                X             C
                                                REP-ACON1_ECOLI       X             C
                        504        C           ACON1_ECOLI           X             C
                                                E01201                X             C
                                                REP-ACON1_ECOLI       X             C
         b0118   P36683 710        C           1l5j                  A             C
                                                1l5j                  B             C
                                                ACON2_ECOLI           X             C
                                                E00113                X             C
                                                REP-1l5j              A             C
                        769        C           1l5j                  A             C
                                                1l5j                  B             C
                                                ACON2_ECOLI           X             C
                                                E00113                X             C
                                                REP-1l5j              A             C
```

```
                772        C           1l5j                  A             C
                                       1l5j                  B             C
                                       ACON2_ECOLI           X             C
                                       E00113                X             C
                                       REP-1l5j              A             C

                                                                 struct_resnum  \
gene_id seq_id seq_resnum seq_residue struct_id
b1276   P25516 435        C           ACON1_ECOLI                        435
                                       E01201                             435
                                       REP-ACON1_ECOLI                    435
                501        C           ACON1_ECOLI                        501
                                       E01201                             501
                                       REP-ACON1_ECOLI                    501
                504        C           ACON1_ECOLI                        504
                                       E01201                             504
                                       REP-ACON1_ECOLI                    504
b0118   P36683 710        C           1l5j                               710
                                       1l5j                               710
                                       ACON2_ECOLI                        710
                                       E00113                             710
                                       REP-1l5j                           710
                769        C           1l5j                               769
                                       1l5j                               769
                                       ACON2_ECOLI                        769
                                       E00113                             769
                                       REP-1l5j                           769
                772        C           1l5j                               772
                                       1l5j                               772
                                       ACON2_ECOLI                        772
                                       E00113                             772
                                       REP-1l5j                           772

                                                                 seq_SS-sspro  \
gene_id seq_id seq_resnum seq_residue struct_id
b1276   P25516 435        C           ACON1_ECOLI                        H
                                       E01201                             H
                                       REP-ACON1_ECOLI                    H
                501        C           ACON1_ECOLI                        C
                                       E01201                             C
                                       REP-ACON1_ECOLI                    C
                504        C           ACON1_ECOLI                        H
                                       E01201                             H
                                       REP-ACON1_ECOLI                    H
b0118   P36683 710        C           1l5j                               C
                                       1l5j                               C
                                       ACON2_ECOLI                        C
                                       E00113                             C
                                       REP-1l5j                           C
                769        C           1l5j                               C
                                       1l5j                               C
                                       ACON2_ECOLI                        C
                                       E00113                             C
                                       REP-1l5j                           C
                772        C           1l5j                               H
                                       1l5j                               H
                                       ACON2_ECOLI                        H
                                       E00113                             H
                                       REP-1l5j                           H
```

```
                                                          seq_SS-sspro8  \
gene_id seq_id seq_resnum seq_residue struct_id
b1276   P25516 435        C           ACON1_ECOLI                 H
                                      E01201                      H
                                      REP-ACON1_ECOLI             H
               501        C           ACON1_ECOLI                 C
                                      E01201                      C
                                      REP-ACON1_ECOLI             C
               504        C           ACON1_ECOLI                 G
                                      E01201                      G
                                      REP-ACON1_ECOLI             G
b0118   P36683 710        C           1l5j                        T
                                      1l5j                        T
                                      ACON2_ECOLI                 T
                                      E00113                      T
                                      REP-1l5j                    T
               769        C           1l5j                        C
                                      1l5j                        C
                                      ACON2_ECOLI                 C
                                      E00113                      C
                                      REP-1l5j                    C
               772        C           1l5j                        G
                                      1l5j                        G
                                      ACON2_ECOLI                 G
                                      E00113                      G
                                      REP-1l5j                    G

                                                          seq_RSA-accpro  \
gene_id seq_id seq_resnum seq_residue struct_id
b1276   P25516 435        C           ACON1_ECOLI                 -
                                      E01201                      -
                                      REP-ACON1_ECOLI             -
               501        C           ACON1_ECOLI                 -
                                      E01201                      -
                                      REP-ACON1_ECOLI             -
               504        C           ACON1_ECOLI                 -
                                      E01201                      -
                                      REP-ACON1_ECOLI             -
b0118   P36683 710        C           1l5j                        -
                                      1l5j                        -
                                      ACON2_ECOLI                 -
                                      E00113                      -
                                      REP-1l5j                    -
               769        C           1l5j                        -
                                      1l5j                        -
                                      ACON2_ECOLI                 -
                                      E00113                      -
                                      REP-1l5j                    -
               772        C           1l5j                        -
                                      1l5j                        -
                                      ACON2_ECOLI                 -
                                      E00113                      -
                                      REP-1l5j                    -

                                                       seq_RSA-accpro20  \
gene_id seq_id seq_resnum seq_residue struct_id
b1276   P25516 435        C           ACON1_ECOLI                10
                                      E01201                     10
```

```
                                    REP-ACON1_ECOLI                10
               501        C         ACON1_ECOLI                     0
                                    E01201                          0
                                    REP-ACON1_ECOLI                 0
               504        C         ACON1_ECOLI                     0
                                    E01201                          0
                                    REP-ACON1_ECOLI                 0
b0118   P36683 710        C         1l5j                           10
                                    1l5j                           10
                                    ACON2_ECOLI                    10
                                    E00113                         10
                                    REP-1l5j                       10
               769        C         1l5j                            5
                                    1l5j                            5
                                    ACON2_ECOLI                     5
                                    E00113                          5
                                    REP-1l5j                        5
               772        C         1l5j                            5
                                    1l5j                            5
                                    ACON2_ECOLI                     5
                                    E00113                          5
                                    REP-1l5j                        5

                                                       struct_SS-dssp  \
gene_id seq_id seq_resnum seq_residue struct_id
b1276   P25516 435        C         ACON1_ECOLI                   NaN
                                    E01201                        NaN
                                    REP-ACON1_ECOLI                 H
               501        C         ACON1_ECOLI                   NaN
                                    E01201                        NaN
                                    REP-ACON1_ECOLI                 S
               504        C         ACON1_ECOLI                   NaN
                                    E01201                        NaN
                                    REP-ACON1_ECOLI                 G
b0118   P36683 710        C         1l5j                          NaN
                                    1l5j                          NaN
                                    ACON2_ECOLI                   NaN
                                    E00113                        NaN
                                    REP-1l5j                        T
               769        C         1l5j                          NaN
                                    1l5j                          NaN
                                    ACON2_ECOLI                   NaN
                                    E00113                        NaN
                                    REP-1l5j                        -
               772        C         1l5j                          NaN
                                    1l5j                          NaN
                                    ACON2_ECOLI                   NaN
                                    E00113                        NaN
                                    REP-1l5j                        G

                                                      struct_RSA-dssp  \
gene_id seq_id seq_resnum seq_residue struct_id
b1276   P25516 435        C         ACON1_ECOLI                   NaN
                                    E01201                        NaN
                                    REP-ACON1_ECOLI          0.059259
               501        C         ACON1_ECOLI                   NaN
                                    E01201                        NaN
                                    REP-ACON1_ECOLI          0.088889
               504        C         ACON1_ECOLI                   NaN
```

```
                                        E01201                        NaN
                                        REP-ACON1_ECOLI          0.259259
b0118   P36683 710        C            1l5j                          NaN
                                        1l5j                          NaN
                                        ACON2_ECOLI                   NaN
                                        E00113                        NaN
                                        REP-1l5j                 0.118519
               769        C            1l5j                          NaN
                                        1l5j                          NaN
                                        ACON2_ECOLI                   NaN
                                        E00113                        NaN
                                        REP-1l5j                 0.088889
               772        C            1l5j                          NaN
                                        1l5j                          NaN
                                        ACON2_ECOLI                   NaN
                                        E00113                        NaN
                                        REP-1l5j                 0.081481

                                                          struct_ASA-dssp  \
gene_id seq_id seq_resnum seq_residue struct_id
b1276   P25516 435        C            ACON1_ECOLI                   NaN
                                        E01201                        NaN
                                        REP-ACON1_ECOLI               8.0
               501        C            ACON1_ECOLI                   NaN
                                        E01201                        NaN
                                        REP-ACON1_ECOLI              12.0
               504        C            ACON1_ECOLI                   NaN
                                        E01201                        NaN
                                        REP-ACON1_ECOLI              35.0
b0118   P36683 710        C            1l5j                          NaN
                                        1l5j                          NaN
                                        ACON2_ECOLI                   NaN
                                        E00113                        NaN
                                        REP-1l5j                     16.0
               769        C            1l5j                          NaN
                                        1l5j                          NaN
                                        ACON2_ECOLI                   NaN
                                        E00113                        NaN
                                        REP-1l5j                     12.0
               772        C            1l5j                          NaN
                                        1l5j                          NaN
                                        ACON2_ECOLI                   NaN
                                        E00113                        NaN
                                        REP-1l5j                     11.0

                                                          struct_PHI-dssp  \
gene_id seq_id seq_resnum seq_residue struct_id
b1276   P25516 435        C            ACON1_ECOLI                   NaN
                                        E01201                        NaN
                                        REP-ACON1_ECOLI             -61.1
               501        C            ACON1_ECOLI                   NaN
                                        E01201                        NaN
                                        REP-ACON1_ECOLI             -61.0
               504        C            ACON1_ECOLI                   NaN
                                        E01201                        NaN
                                        REP-ACON1_ECOLI             -56.0
b0118   P36683 710        C            1l5j                          NaN
                                        1l5j                          NaN
                                        ACON2_ECOLI                   NaN
```

```
                                    E00113                      NaN
                                    REP-1l5j                  -67.1
                769        C        1l5j                        NaN
                                    1l5j                        NaN
                                    ACON2_ECOLI                 NaN
                                    E00113                      NaN
                                    REP-1l5j                  -67.8
                772        C        1l5j                        NaN
                                    1l5j                        NaN
                                    ACON2_ECOLI                 NaN
                                    E00113                      NaN
                                    REP-1l5j                  -50.2

                                                    struct_PSI-dssp  \
gene_id seq_id seq_resnum seq_residue struct_id
b1276   P25516 435        C        ACON1_ECOLI                 NaN
                                    E01201                      NaN
                                    REP-ACON1_ECOLI           -26.6
                501        C        ACON1_ECOLI                 NaN
                                    E01201                      NaN
                                    REP-ACON1_ECOLI           -50.0
                504        C        ACON1_ECOLI                 NaN
                                    E01201                      NaN
                                    REP-ACON1_ECOLI           -45.6
b0118   P36683 710        C        1l5j                        NaN
                                    1l5j                        NaN
                                    ACON2_ECOLI                 NaN
                                    E00113                      NaN
                                    REP-1l5j                   -7.2
                769        C        1l5j                        NaN
                                    1l5j                        NaN
                                    ACON2_ECOLI                 NaN
                                    E00113                      NaN
                                    REP-1l5j                  -28.3
                772        C        1l5j                        NaN
                                    1l5j                        NaN
                                    ACON2_ECOLI                 NaN
                                    E00113                      NaN
                                    REP-1l5j                  -38.0

                                                    struct_CA_DEPTH-msms  \
gene_id seq_id seq_resnum seq_residue struct_id
b1276   P25516 435        C        ACON1_ECOLI                      NaN
                                    E01201                           NaN
                                    REP-ACON1_ECOLI             2.656722
                501        C        ACON1_ECOLI                      NaN
                                    E01201                           NaN
                                    REP-ACON1_ECOLI             1.999713
                504        C        ACON1_ECOLI                      NaN
                                    E01201                           NaN
                                    REP-ACON1_ECOLI             1.999634
b0118   P36683 710        C        1l5j                             NaN
                                    1l5j                             NaN
                                    ACON2_ECOLI                      NaN
                                    E00113                           NaN
                                    REP-1l5j                   10.148960
                769        C        1l5j                             NaN
                                    1l5j                             NaN
                                    ACON2_ECOLI                      NaN
```

```
                                        E00113                          NaN
                                        REP-1l5j                   8.296585
                  772        C          1l5j                            NaN
                                        1l5j                            NaN
                                        ACON2_ECOLI                     NaN
                                        E00113                          NaN
                                        REP-1l5j                   8.282292

                                                          struct_RES_DEPTH-msms
gene_id seq_id seq_resnum seq_residue struct_id
b1276   P25516 435        C           ACON1_ECOLI                       NaN
                                       E01201                            NaN
                                       REP-ACON1_ECOLI              2.813536
                  501        C          ACON1_ECOLI                       NaN
                                       E01201                            NaN
                                       REP-ACON1_ECOLI              2.409119
                  504        C          ACON1_ECOLI                       NaN
                                       E01201                            NaN
                                       REP-ACON1_ECOLI              1.961484
b0118   P36683 710        C           1l5j                              NaN
                                       1l5j                              NaN
                                       ACON2_ECOLI                       NaN
                                       E00113                            NaN
                                       REP-1l5j                     10.009109
                  769        C          1l5j                              NaN
                                       1l5j                              NaN
                                       ACON2_ECOLI                       NaN
                                       E00113                            NaN
                                       REP-1l5j                     8.049832
                  772        C          1l5j                              NaN
                                       1l5j                              NaN
                                       ACON2_ECOLI                       NaN
                                       E00113                            NaN
                                       REP-1l5j                     8.239369
```

GEM-PRO - Genes & Sequences

This notebook gives an example of how to run the GEM-PRO pipeline with a **dictionary of gene IDs and their protein sequences**.

Input: Dictionary of gene IDs and protein sequences

Output: GEM-PRO model

Imports

```
In [1]: import sys
        import logging
```

```
In [2]: # Import the GEM-PRO class
        from ssbio.pipeline.gempro import GEMPRO
```

```
In [3]: # Printing multiple outputs per cell
        from IPython.core.interactiveshell import InteractiveShell
        InteractiveShell.ast_node_interactivity = "all"
```

Logging

Set the logging level in `logger.setLevel(logging.<LEVEL_HERE>)` to specify how verbose you want the pipeline to be. Debug is most verbose.

- `CRITICAL`
  - Only really important messages shown
- `ERROR`
  - Major errors
- `WARNING`
  - Warnings that don't affect running of the pipeline
- `INFO` (default)
  - Info such as the number of structures mapped per gene
- `DEBUG`
  - Really detailed information that will print out a lot of stuff

Warning: `DEBUG` mode prints out a large amount of information, especially if you have a lot of genes. This may stall your notebook!

```
In [4]: # Create logger
        logger = logging.getLogger()
        logger.setLevel(logging.INFO)  # SET YOUR LOGGING LEVEL HERE #

In [5]: # Other logger stuff for Jupyter notebooks
        handler = logging.StreamHandler(sys.stderr)
        formatter = logging.Formatter('[%(asctime)s] [%(name)s] %(levelname)s: %(message)s', datefmt=
        handler.setFormatter(formatter)
        logger.handlers = [handler]
```

Initialization of the project

Set these three things:

- `ROOT_DIR`
  - The directory where a folder named after your `PROJECT` will be created
- `PROJECT`
  - Your project name
- `LIST_OF_GENES`
  - Your list of gene IDs

A directory will be created in `ROOT_DIR` with your `PROJECT` name. The folders are organized like so:

```
ROOT_DIR
└── PROJECT
    ├── data  # General storage for pipeline outputs
    ├── model  # SBML and GEM-PRO models are stored here
    ├── genes  # Per gene information
    │   ├── <gene_id1>  # Specific gene directory
    │   │   └── protein
    │   │       ├── sequences  # Protein sequence files, alignments, etc.
    │   │       └── structures  # Protein structure files, calculations, etc.
    │   └── <gene_id2>
    │       └── protein
    │           ├── sequences
    │           └── structures
    ├── reactions  # Per reaction information
    │   └── <reaction_id1>  # Specific reaction directory
    │       └── complex
    │           └── structures  # Protein complex files
    └── metabolites  # Per metabolite information
        └── <metabolite_id1>  # Specific metabolite directory
            └── chemical
                └── structures  # Metabolite 2D and 3D structure files
```

Note: Methods for protein complexes and metabolites are still in development.

```
In [6]: # SET FOLDERS AND DATA HERE
        import tempfile
        ROOT_DIR = tempfile.gettempdir()

        PROJECT = 'genes_and_sequences_GP'
        GENES_AND_SEQUENCES = {'b0870': 'MIDLRSDTVTRPSRAMLEAMMAAPVGDDVYGDDPTVNALQDYAAELSGKEAAIFLPTGT
                               'b3041': 'MNQTLLSSFGTPFERVENALAALREGRGVMVLDDEDRENEGDMIFPAETMTVEQMALTI
        PDB_FILE_TYPE = 'mmtf'

In [7]: # Create the GEM-PRO project
        my_gempro = GEMPRO(gem_name=PROJECT, root_dir=ROOT_DIR, genes_and_sequences=GENES_AND_SEQUENC

[2017-11-21 19:21] [ssbio.pipeline.gempro] INFO: Creating GEM-PRO project directory in folder /tmp
[2017-11-21 19:21] [ssbio.pipeline.gempro] INFO: /tmp/genes_and_sequences_GP: GEM-PRO project locati
[2017-11-21 19:21] [ssbio.pipeline.gempro] INFO: Loaded in 2 sequences
[2017-11-21 19:21] [ssbio.pipeline.gempro] INFO: 2: number of genes
```

Mapping sequence –> structure

Since the sequences have been provided, we just need to BLAST them to the PDB.

Note: These methods do not download any 3D structure files.

Methods

.. automethod:: ssbio.pipeline.gempro.GEMPRO.blast$_seqs_to_pdb$: $noindex$:

```
In [8]: # Mapping using BLAST
        my_gempro.blast_seqs_to_pdb(all_genes=True, seq_ident_cutoff=.9, evalue=0.00001)
        my_gempro.df_pdb_blast.head(2)
```

```
HBox(children=(IntProgress(value=0, max=2), HTML(value='')))
```

```
[2017-11-21 19:21] [ssbio.pipeline.gempro] INFO: Completed sequence --> PDB BLAST. See the "df_pdb_bl
[2017-11-21 19:21] [ssbio.pipeline.gempro] INFO: 2: number of genes with additional structures added
```

```
Out[8]: pdb_id pdb_chain_id  hit_score  hit_evalue  hit_percent_similar  \
        gene
        b0870   3wlx            A     1713.0         0.0                  1.0
        b0870   3wlx            B     1713.0         0.0                  1.0

               hit_percent_ident  hit_num_ident  hit_num_similar
        gene
        b0870                1.0            333              333
        b0870                1.0            333              333
```

Downloading and ranking structures

Methods

.. automethod:: ssbio.pipeline.gempro.GEMPRO.pdb$_{d}ownloader_{a}nd_{m}etadata$: $noindex$:

Warning: Downloading all PDBs takes a while, since they are also parsed for metadata. You can skip this step and just set representative structures below if you want to minimize the number of PDBs downloaded.

```
In [9]: # Download all mapped PDBs and gather the metadata
        my_gempro.pdb_downloader_and_metadata()
        my_gempro.df_pdb_metadata.head(2)
```

```
HBox(children=(IntProgress(value=0, max=2), HTML(value='')))
```

```
[2017-11-21 19:21] [ssbio.pipeline.gempro] INFO: Updated PDB metadata dataframe. See the "df_pdb_meta
[2017-11-21 19:21] [ssbio.pipeline.gempro] INFO: Saved 11 structures total
```

```
Out[9]: pdb_id                                          pdb_title  \
        gene
        b0870   3wlx  Crystal structure of low-specificity L-threoni...
        b0870   4lnj  Structure of Escherichia coli Threonine Aldola...

                                                 description experimental_method  \
        gene
        b0870  Low specificity L-threonine aldolase (E.C.4.1....   X-RAY DIFFRACTION
        b0870  Low-specificity L-threonine aldolase (E.C.4.1....   X-RAY DIFFRACTION

              mapped_chains  resolution   chemicals     taxonomy_name structure_file
        gene
        b0870           A;B        2.51         PLG  Escherichia coli      3wlx.mmtf
        b0870           A;B        2.10  EPE;MG;PLR  Escherichia coli      4lnj.mmtf
```

.. automethod:: ssbio.pipeline.gempro.GEMPRO.set$_{r}epresentative_{s}tructure$: $noindex$:

```
In [10]: # Set representative structures
         my_gempro.set_representative_structure()
         my_gempro.df_representative_structures.head()
```

```
HBox(children=(IntProgress(value=0, max=2), HTML(value='')))
```

```
[2017-11-21 19:21] [ssbio.pipeline.gempro] INFO: 2/2: number of genes with a representative structure
[2017-11-21 19:21] [ssbio.pipeline.gempro] INFO: See the "df_representative_structures" attribute for
```

```
Out[10]: id  is_experimental file_type    structure_file
         gene
         b0870  REP-3wlx             True       pdb  3wlx-A_clean.pdb
         b3041  REP-1iez             True       pdb  1iez-A_clean.pdb
```

```
In [11]: # Looking at the information saved within a gene
         my_gempro.genes.get_by_id('b0870').protein.representative_structure
         my_gempro.genes.get_by_id('b0870').protein.representative_structure.get_dict()
```

```
Out[11]: <StructProp REP-3wlx at 0x7f4572afceb8>
```

```
Out[11]: {'_structure_dir': '/tmp/genes_and_sequences_GP/genes/b0870/b0870_protein/structures',
          'chains': [<ChainProp A at 0x7f4571032e48>],
          'date': None,
          'description': 'Low specificity L-threonine aldolase (E.C.4.1.2.48)',
          'file_type': 'pdb',
          'id': 'REP-3wlx',
          'is_experimental': True,
          'mapped_chains': ['A'],
          'notes': {},
          'original_structure_id': '3wlx',
          'resolution': 2.51,
          'structure_file': '3wlx-A_clean.pdb',
          'taxonomy_name': 'Escherichia coli'}
```

Creating homology models

For those proteins with no representative structure, we can create homology models for them. `ssbio` contains some built in functions for easily running I-TASSER locally or on machines with `SLURM` (ie. on NERSC) or `Torque` job scheduling.

You can load in I-TASSER models once they complete using the `get_itasser_models` later.

Info: Homology modeling can take a long time - about 24-72 hours per protein (highly dependent on the sequence length, as well as if there are available templates).

Methods

.. automethod:: ssbio.pipeline.gempro.GEMPRO.prep$_{i}tasser_{m}odels$: $noindex$:

```
In [12]: # Prep I-TASSER model folders
         my_gempro.prep_itasser_modeling('~/software/I-TASSER4.4', '~/software/ITLIB/', runtype='loca
```

```
[2017-11-21 19:21] [ssbio.pipeline.gempro] INFO: Prepared I-TASSER modeling folders for 0 genes in fo
```

Saving your GEM-PRO

Warning: Saving is still experimental. For a full GEM-PRO with sequences & structures, depending on the number of genes, saving can take >5 minutes.

.. automethod:: ssbio.pipeline.gempro.GEMPRO.save$_json$: $noindex$:

```
In [13]: import os.path as op
         my_gempro.save_json(op.join(my_gempro.model_dir, '{}.json'.format(my_gempro.id)), compressio

[2017-11-21 19:21] [root] WARNING: json-tricks: numpy scalar serialization is experimental and may wo
[2017-11-21 19:21] [ssbio.core.io] INFO: Saved <class 'ssbio.pipeline.gempro.GEMPRO'> (id: genes_and_
```

GEM-PRO - List of Gene IDs

This notebook gives an example of how to run the GEM-PRO pipeline with a **list of gene IDs**.

Input: List of gene IDs

Output: GEM-PRO model

Imports

```
In [1]: import sys
        import logging
```

```
In [2]: # Import the GEM-PRO class
        from ssbio.pipeline.gempro import GEMPRO
```

```
In [3]: # Printing multiple outputs per cell
        from IPython.core.interactiveshell import InteractiveShell
        InteractiveShell.ast_node_interactivity = "all"
```

Logging

Set the logging level in `logger.setLevel(logging.<LEVEL_HERE>)` to specify how verbose you want the pipeline to be. Debug is most verbose.

- `CRITICAL`
  - Only really important messages shown
- `ERROR`
  - Major errors
- `WARNING`
  - Warnings that don't affect running of the pipeline
- `INFO` (default)
  - Info such as the number of structures mapped per gene

- DEBUG
  - Really detailed information that will print out a lot of stuff

Warning: DEBUG mode prints out a large amount of information, especially if you have a lot of genes. This may stall your notebook!

```
In [4]: # Create logger
        logger = logging.getLogger()
        logger.setLevel(logging.INFO)  # SET YOUR LOGGING LEVEL HERE #

In [5]: # Other logger stuff for Jupyter notebooks
        handler = logging.StreamHandler(sys.stderr)
        formatter = logging.Formatter('[%(asctime)s] [%(name)s] %(levelname)s: %(message)s', datefmt=
        handler.setFormatter(formatter)
        logger.handlers = [handler]
```

Initialization of the project

Set these three things:

- ROOT_DIR
  - The directory where a folder named after your PROJECT will be created
- PROJECT
  - Your project name
- LIST_OF_GENES
  - Your list of gene IDs

A directory will be created in ROOT_DIR with your PROJECT name. The folders are organized like so:

```
ROOT_DIR
└── PROJECT
    ├── data  # General storage for pipeline outputs
    ├── model  # SBML and GEM-PRO models are stored here
    ├── genes  # Per gene information
    │   ├── <gene_id1>  # Specific gene directory
    │   │   └── protein
    │   │       ├── sequences  # Protein sequence files, alignments, etc.
    │   │       └── structures  # Protein structure files, calculations, etc.
    │   └── <gene_id2>
    │       └── protein
    │           ├── sequences
    │           └── structures
    ├── reactions  # Per reaction information
    │   └── <reaction_id1>  # Specific reaction directory
    │       └── complex
    │           └── structures  # Protein complex files
    └── metabolites  # Per metabolite information
        └── <metabolite_id1>  # Specific metabolite directory
            └── chemical
                └── structures  # Metabolite 2D and 3D structure files
```

Note: Methods for protein complexes and metabolites are still in development.

```
In [6]: # SET FOLDERS AND DATA HERE
        import tempfile
        ROOT_DIR = tempfile.gettempdir()

        PROJECT = 'genes_GP'
        LIST_OF_GENES = ['b0761', 'b0889', 'b0995', 'b1013', 'b1014', 'b1040', 'b1130', 'b1187', 'b12
        PDB_FILE_TYPE = 'mmtf'
```

```
In [7]: # Create the GEM-PRO project
        my_gempro = GEMPRO(gem_name=PROJECT, root_dir=ROOT_DIR, genes_list=LIST_OF_GENES, pdb_file_ty
```

```
[2017-11-21 19:21] [ssbio.pipeline.gempro] INFO: Creating GEM-PRO project directory in folder /tmp
[2017-11-21 19:21] [ssbio.pipeline.gempro] INFO: /tmp/genes_GP: GEM-PRO project location
[2017-11-21 19:21] [ssbio.pipeline.gempro] INFO: 10: number of genes
```

Mapping gene ID –> sequence

First, we need to map these IDs to their protein sequences. There are 2 ID mapping services provided to do this - through **KEGG** or **UniProt**. The end goal is to map a UniProt ID to each ID, since there is a comprehensive mapping (and some useful APIs) between UniProt and the PDB.

Note: You only need to map gene IDs using one service. However you can run both if some genes don't map in one service and do map in another!

Methods

.. automethod:: ssbio.pipeline.gempro.GEMPRO.kegg$_{m}apping_{a}nd_{m}etadata$: $noindex$:

```
In [8]: # KEGG mapping of gene ids
        my_gempro.kegg_mapping_and_metadata(kegg_organism_code='eco')
        print('Missing KEGG mapping: ', my_gempro.missing_kegg_mapping)
        my_gempro.df_kegg_metadata.head()
```

```
HBox(children=(IntProgress(value=0, max=10), HTML(value='')))
```

```
[2017-11-21 19:21] [ssbio.pipeline.gempro] INFO: 10/10: number of genes mapped to KEGG
[2017-11-21 19:21] [ssbio.pipeline.gempro] INFO: Completed ID mapping --> KEGG. See the "df_kegg_meta
```

```
Missing KEGG mapping:  []
```

```
Out[8]: kegg     refseq uniprot  \
        gene
        b0761  eco:b0761  NP_415282  P0A9G8
        b0889  eco:b0889  NP_415409  P0ACJ0
        b0995  eco:b0995  NP_415515  P38684
        b1013  eco:b1013  NP_415533  P0ACU2
        b1014  eco:b1014  NP_415534  P09546

                                                      pdbs  sequence_file  \
        gene
        b0761                          1B9M;1H9S;1B9N;1O7L;1H9R  eco-b0761.faa
        b0889                                       2GQQ;2L4A  eco-b0889.faa
```

```
        b0995                                                 1ZGZ  eco-b0995.faa
        b1013                                  4JYK;4XK4;4X1E;3LOC  eco-b1013.faa
        b1014  3E2Q;4JNZ;3E2R;4JNY;2GPE;4O8A;3E2S;2FZN;1TJ1;1...  eco-b1014.faa

                metadata_file
        gene
        b0761  eco-b0761.kegg
        b0889  eco-b0889.kegg
        b0995  eco-b0995.kegg
        b1013  eco-b1013.kegg
        b1014  eco-b1014.kegg
```

.. automethod:: ssbio.pipeline.gempro.GEMPRO.uniprot$_m apping_a nd_m etadata$: $noindex$:

```
In [9]: # UniProt mapping
        my_gempro.uniprot_mapping_and_metadata(model_gene_source='ENSEMBLGENOME_ID')
        print('Missing UniProt mapping: ', my_gempro.missing_uniprot_mapping)
        my_gempro.df_uniprot_metadata.head()
```

```
[2017-11-21 19:21] [root] INFO: getUserAgent: Begin
[2017-11-21 19:21] [root] INFO: getUserAgent: user_agent: EBI-Sample-Client/ (services.py; Python 3.6
[2017-11-21 19:21] [root] INFO: getUserAgent: End
```

```
HBox(children=(IntProgress(value=0, max=10), HTML(value='')))
```

```
[2017-11-21 19:21] [ssbio.pipeline.gempro] INFO: 10/10: number of genes mapped to UniProt
[2017-11-21 19:21] [ssbio.pipeline.gempro] INFO: Completed ID mapping --> UniProt. See the "df_unipro
```

```
Missing UniProt mapping:  []
```

```
Out[9]: uniprot  reviewed gene_name                 kegg  \
        gene
        b0761  P0A9G8     False      modE  ecj:JW0744;eco:b0761
        b0889  P0ACJ0     False       lrp  ecj:JW0872;eco:b0889
        b0995  P38684     False      torR  ecj:JW0980;eco:b0995
        b1013  P0ACU2     False      rutR  ecj:JW0998;eco:b1013
        b1014  P09546     False      putA  ecj:JW0999;eco:b1014

                                refseq  \
        gene
        b0761  NP_415282.1;WP_001147439.1
        b0889  NP_415409.1;WP_000228473.1
        b0995  NP_415515.1;WP_001120125.1
        b1013  NP_415533.1;WP_000191701.1
        b1014  NP_415534.1;WP_001326840.1

                                                      pdbs  \
        gene
        b0761                           1B9M;1B9N;1H9R;1H9S;1O7L
        b0889                                       2GQQ;2L4A
        b0995                                            1ZGZ
        b1013                                3LOC;4JYK;4X1E;4XK4
        b1014  1TIW;1TJ0;1TJ1;1TJ2;2AY0;2FZM;2FZN;2GPE;2RBF;3...

                               pfam  \
        gene
        b0761          PF00126;PF03459
        b0889                  PF01037
        b0995          PF00072;PF00486
        b1013          PF00440;PF08362
```

```
b1014  PF00171;PF01619;PF14850

                                                 description  entry_date  \
gene
b0761                      Transcriptional regulator ModE  2017-10-25
b0889               Leucine-responsive regulatory protein  2017-10-25
b0995  TorCAD operon transcriptional regulatory prote...  2017-10-25
b1013             HTH-type transcriptional regulator RutR  2017-10-25
b1014                           Bifunctional protein PutA  2017-10-25

       entry_version    seq_date  seq_version sequence_file metadata_file
gene
b0761            105  2005-07-19            1  P0A9G8.fasta    P0A9G8.xml
b0889            106  2007-01-23            2  P0ACJ0.fasta    P0ACJ0.xml
b0995            148  1997-11-01            2  P38684.fasta    P38684.xml
b1013             99  2005-11-22            1  P0ACU2.fasta    P0ACU2.xml
b1014            179  1997-11-01            3  P09546.fasta    P09546.xml
```

.. automethod:: ssbio.pipeline.gempro.GEMPRO.set$_{r}epresentative_{s}equence$: $noindex$:

```
In [10]: # Set representative sequences
         my_gempro.set_representative_sequence()
         print('Missing a representative sequence: ', my_gempro.missing_representative_sequence)
         my_gempro.df_representative_sequences.head()
```

```
HBox(children=(IntProgress(value=0, max=10), HTML(value='')))


[2017-11-21 19:21] [ssbio.pipeline.gempro] INFO: 10/10: number of genes with a representative sequenc
[2017-11-21 19:21] [ssbio.pipeline.gempro] INFO: See the "df_representative_sequences" attribute for

Missing a representative sequence:  []
```

```
Out[10]: uniprot                 kegg  \
         gene
         b0761  P0A9G8  ecj:JW0744;eco:b0761
         b0889  P0ACJ0  ecj:JW0872;eco:b0889
         b0995  P38684  ecj:JW0980;eco:b0995
         b1013  P0ACU2  ecj:JW0998;eco:b1013
         b1014  P09546  ecj:JW0999;eco:b1014

                                                       pdbs sequence_file  \
         gene
         b0761                            1B9M;1B9N;1H9R;1H9S;1O7L  P0A9G8.fasta
         b0889                                        2GQQ;2L4A  P0ACJ0.fasta
         b0995                                             1ZGZ  P38684.fasta
         b1013                                3LOC;4JYK;4X1E;4XK4  P0ACU2.fasta
         b1014  1TIW;1TJ0;1TJ1;1TJ2;2AY0;2FZM;2FZN;2GPE;2RBF;3...  P09546.fasta

               metadata_file
         gene
         b0761    P0A9G8.xml
         b0889    P0ACJ0.xml
         b0995    P38684.xml
         b1013    P0ACU2.xml
         b1014    P09546.xml
```

Mapping representative sequence –> structure

These are the ways to map sequence to structure:

1. Use the UniProt ID and their automatic mappings to the PDB
2. BLAST the sequence to the PDB
3. Make homology models or
4. Map to existing homology models

You can only utilize option #1 to map to PDBs if there is a mapped UniProt ID set in the representative sequence. If not, you'll have to BLAST your sequence to the PDB or make a homology model. You can also run both for maximum coverage.

Methods

.. automethod:: ssbio.pipeline.gempro.GEMPRO.map$_u niprot_t o_p db$: $noindex$:

```
In [11]: # Mapping using the PDBe best_structures service
         my_gempro.map_uniprot_to_pdb(seq_ident_cutoff=.3)
         my_gempro.df_pdb_ranking.head()
```

```
[2017-11-21 19:21] [ssbio.pipeline.gempro] INFO: Mapping UniProt IDs --> PDB IDs...
[2017-11-21 19:21] [root] INFO: getUserAgent: Begin
[2017-11-21 19:21] [root] INFO: getUserAgent: user_agent: EBI-Sample-Client/ (services.py; Python 3.6
[2017-11-21 19:21] [root] INFO: getUserAgent: End
```

```
HBox(children=(IntProgress(value=0, max=10), HTML(value='')))
```

```
[2017-11-21 19:22] [ssbio.pipeline.gempro] INFO: 8/10: number of genes with at least one experimental
[2017-11-21 19:22] [ssbio.pipeline.gempro] INFO: Completed UniProt --> best PDB mapping. See the "df_
```

```
Out[11]: pdb_id pdb_chain_id uniprot experimental_method  resolution  coverage  \
         gene
         b0761   1b9n            A  P0A9G8   X-ray diffraction        2.09     1.000
         b0761   1b9m            A  P0A9G8   X-ray diffraction        1.75     1.000
         b0761   1b9m            B  P0A9G8   X-ray diffraction        1.75     1.000
         b0761   1b9n            B  P0A9G8   X-ray diffraction        2.09     1.000
         b0761   1h9r            A  P0A9G8   X-ray diffraction        1.90     0.534

                start  end  unp_start  unp_end  rank
         gene
         b0761      4  265          1      262     3
         b0761      4  265          1      262     1
         b0761      4  265          1      262     2
         b0761      4  265          1      262     4
         b0761      1  140        123      262    11
```

.. automethod:: ssbio.pipeline.gempro.GEMPRO.blast$_s eqs_t o_p db$: $noindex$:

```
In [12]: # Mapping using BLAST
         my_gempro.blast_seqs_to_pdb(all_genes=True, seq_ident_cutoff=.9, evalue=0.00001)
         my_gempro.df_pdb_blast.head(2)
```

```
HBox(children=(IntProgress(value=0, max=10), HTML(value='')))
```

```
[2017-11-21 19:22] [ssbio.pipeline.gempro] INFO: Completed sequence --> PDB BLAST. See the "df_pdb_bl
[2017-11-21 19:22] [ssbio.pipeline.gempro] INFO: 1: number of genes with additional structures added
```

Out[12]:

gene	pdb_id	pdb_chain_id	hit_score	hit_evalue	hit_percent_similar	hit_percent_ident	hit_num_ident	hit_num_similar
b1013	4x1e	A	966.0	1.382400e-104	0.910377	0.910377	193	193
b1013	4x1e	B	966.0	1.382400e-104	0.910377	0.910377	193	193

Downloading and ranking structures

Methods

.. automethod:: ssbio.pipeline.gempro.GEMPRO.pdb$_{d}ownloader_{a}nd_{m}etadata$: $noindex$:

Warning: Downloading all PDBs takes a while, since they are also parsed for metadata. You can skip this step and just set representative structures below if you want to minimize the number of PDBs downloaded.

```
In [13]: # Download all mapped PDBs and gather the metadata
         my_gempro.pdb_downloader_and_metadata()
         my_gempro.df_pdb_metadata.head(2)
```

```
HBox(children=(IntProgress(value=0, max=10), HTML(value='')))
```

```
[2017-11-21 19:22] [ssbio.pipeline.gempro] INFO: Updated PDB metadata dataframe. See the "df_pdb_meta
[2017-11-21 19:22] [ssbio.pipeline.gempro] INFO: Saved 40 structures total
```

Out[13]:

gene	chemicals	description	experimental_method	mapped_chains	pdb_id	pdb_title	resolution	structure_file	taxonomy_name
b0761	MOO	MOLYBDENUM TRANSPORT PROTEIN MODE	X-ray diffraction	A;B	1h9s	Molybdate bound complex of Dimop domain of Mod...	1.82	1h9s.mmtf	ESCHERICHIA COLI;ESCHERICHIA COLI
b0761	NI;WO4	MOLYBDENUM TRANSPORT PROTEIN MODE	X-ray diffraction	A;B	1h9r	Tungstate bound complex Dimop domain of ModE f...	1.90	1h9r.mmtf	ESCHERICHIA COLI

.. automethod:: ssbio.pipeline.gempro.GEMPRO.set$_{r}epresentative_{s}tructure$: $noindex$:

```
In [14]: # Set representative structures
         my_gempro.set_representative_structure()
         my_gempro.df_representative_structures.head()
```

```
HBox(children=(IntProgress(value=0, max=10), HTML(value='')))
```

```
[2017-11-21 19:22] [ssbio.core.protein] WARNING: b1130: no structures meet quality checks
[2017-11-21 19:22] [ssbio.core.protein] WARNING: b1014: no structures meet quality checks
[2017-11-21 19:22] [ssbio.core.protein] WARNING: b0995: no structures meet quality checks
```

```
[2017-11-21 19:22] [ssbio.pipeline.gempro] INFO: 5/10: number of genes with a representative structur
[2017-11-21 19:22] [ssbio.pipeline.gempro] INFO: See the "df_representative_structures" attribute for
```

Out[14]:

gene	id	is_experimental	file_type	structure_file
b0761	REP-1b9m	True	pdb	1b9m-A_clean.pdb
b0889	REP-2gqq	True	pdb	2gqq-A_clean.pdb
b1013	REP-4jyk	True	pdb	4jyk-A_clean.pdb
b1187	REP-1hw1	True	pdb	1hw1-A_clean.pdb
b1221	REP-1a04	True	pdb	1a04-A_clean.pdb

In [ ]:
```
# Looking at the information saved within a gene
my_gempro.genes.get_by_id('b1187').protein.representative_structure
my_gempro.genes.get_by_id('b1187').protein.representative_structure.get_dict()
```

```
<StructProp REP-1hw1 at 0x7fc7d120da20>
```

```
{'_structure_dir': '/tmp/genes_GP/genes/b1187/b1187_protein/structures',
 'chains': [<ChainProp A at 0x7fc7d1016e48>],
 'date': None,
 'description': 'FATTY ACID METABOLISM REGULATOR PROTEIN',
 'file_type': 'pdb',
 'id': 'REP-1hw1',
 'is_experimental': True,
 'mapped_chains': ['A'],
 'notes': {},
 'original_structure_id': '1hw1',
 'resolution': 1.5,
 'structure_file': '1hw1-A_clean.pdb',
 'taxonomy_name': 'Escherichia coli'}
```

Saving your GEM-PRO

Warning: Saving is still experimental. For a full GEM-PRO with sequences & structures, depending on the number of genes, saving can take >5 minutes.

.. automethod:: ssbio.pipeline.gempro.GEMPRO.save$_json$: $noindex$:

In [ ]:
```
import os.path as op
my_gempro.save_json(op.join(my_gempro.model_dir, '{}.json'.format(my_gempro.id)), compressio
```

```
[2017-11-21 19:22] [root] WARNING: json-tricks: numpy scalar serialization is experimental and may wo
```

GEM-PRO - SBML Model (iNJ661)

This notebook gives an example of how to run the GEM-PRO pipeline with a **SBML model**, in this case *i*NJ661, the metabolic model of *M. tuberculosis*.

Input: GEM (in SBML, JSON, or MAT formats)

Output: GEM-PRO model

Imports

```
In [1]: import sys
        import logging
```

```
In [2]: # Import the GEM-PRO class
        from ssbio.pipeline.gempro import GEMPRO
```

```
In [3]: # Printing multiple outputs per cell
        from IPython.core.interactiveshell import InteractiveShell
        InteractiveShell.ast_node_interactivity = "all"
```

Logging

Set the logging level in `logger.setLevel(logging.<LEVEL_HERE>)` to specify how verbose you want the pipeline to be. Debug is most verbose.

- `CRITICAL`
  - Only really important messages shown
- `ERROR`
  - Major errors
- `WARNING`
  - Warnings that don't affect running of the pipeline
- `INFO` (default)
  - Info such as the number of structures mapped per gene
- `DEBUG`
  - Really detailed information that will print out a lot of stuff

Warning: `DEBUG` mode prints out a large amount of information, especially if you have a lot of genes. This may stall your notebook!

```
In [4]: # Create logger
        logger = logging.getLogger()
        logger.setLevel(logging.INFO)  # SET YOUR LOGGING LEVEL HERE #
```

```
In [5]: # Other logger stuff for Jupyter notebooks
        handler = logging.StreamHandler(sys.stderr)
        formatter = logging.Formatter('[%(asctime)s] [%(name)s] %(levelname)s: %(message)s', datefmt=
        handler.setFormatter(formatter)
        logger.handlers = [handler]
```

Initialization of the project

Set these three things:

- `ROOT_DIR`
  - The directory where a folder named after your `PROJECT` will be created
- `PROJECT`

  - Your project name

- `LIST_OF_GENES`
  - Your list of gene IDs

A directory will be created in `ROOT_DIR` with your `PROJECT` name. The folders are organized like so:

```
ROOT_DIR
└── PROJECT
    ├── data  # General storage for pipeline outputs
    ├── model  # SBML and GEM-PRO models are stored here
    ├── genes  # Per gene information
    │   ├── <gene_id1>  # Specific gene directory
    │   │   └── protein
    │   │       ├── sequences  # Protein sequence files, alignments, etc.
    │   │       └── structures  # Protein structure files, calculations, etc.
    │   └── <gene_id2>
    │       └── protein
    │           ├── sequences
    │           └── structures
    ├── reactions  # Per reaction information
    │   └── <reaction_id1>  # Specific reaction directory
    │       └── complex
    │           └── structures  # Protein complex files
    └── metabolites  # Per metabolite information
        └── <metabolite_id1>  # Specific metabolite directory
            └── chemical
                └── structures  # Metabolite 2D and 3D structure files
```

Note: Methods for protein complexes and metabolites are still in development.

```
In [11]: # SET FOLDERS AND DATA HERE
         import tempfile
         ROOT_DIR = tempfile.gettempdir()

         PROJECT = 'mtuberculosis_gp_atlas'
         GEM_FILE = '/home/nathan/projects_unsynced/mtuberculosis_gp/model/iNJ661.json'
         GEM_FILE_TYPE = 'json'
         PDB_FILE_TYPE = 'mmtf'

In [12]: # Create the GEM-PRO project
         my_gempro = GEMPRO(gem_name=PROJECT, root_dir=ROOT_DIR, gem_file_path=GEM_FILE, gem_file_typ

[2017-11-21 19:23] [ssbio.pipeline.gempro] INFO: Creating GEM-PRO project directory in folder /tmp
[2017-11-21 19:23] [ssbio.pipeline.gempro] INFO: /tmp/mtuberculosis_gp_atlas: GEM-PRO project locatio
[2017-11-21 19:23] [ssbio.pipeline.gempro] INFO: iNJ661: loaded model
[2017-11-21 19:23] [ssbio.pipeline.gempro] INFO: 1025: number of reactions
[2017-11-21 19:23] [ssbio.pipeline.gempro] INFO: 720: number of reactions linked to a gene
[2017-11-21 19:23] [ssbio.pipeline.gempro] INFO: 661: number of genes (excluding spontaneous)
[2017-11-21 19:23] [ssbio.pipeline.gempro] INFO: 826: number of metabolites
[2017-11-21 19:23] [ssbio.pipeline.gempro] WARNING: IMPORTANT: All Gene objects have been transformed
[2017-11-21 19:23] [ssbio.pipeline.gempro] INFO: 661: number of genes
```

Mapping gene ID –> sequence

First, we need to map these IDs to their protein sequences. There are 2 ID mapping services provided to do this - through **KEGG** or **UniProt**. The end goal is to map a UniProt ID to each ID, since there is a comprehensive mapping

(and some useful APIs) between UniProt and the PDB.

Note: You only need to map gene IDs using one service. However you can run both if some genes don't map in one service and do map in another!

However, you don't need to map using these services if you already have the amino acid sequences for each protein. You can just manually load in the sequences as shown using the method `manual_seq_mapping`. Or, if you already have the UniProt IDs, you can load those in using the method `manual_uniprot_mapping`.

Methods

.. automethod:: ssbio.pipeline.gempro.GEMPRO.manual$_{s}eq_{m}apping$: $noindex$:

```
In [13]: gene_to_seq_dict = {'Rv1295': 'MTVPPTATHQPWPGVIAAYRDRLPVGDDWTPVTLLEGGTPLIAATNLSKQTGCTIHLKVEC
                             'Rv2233': 'VSSPRERRPASQAPRLSRRPPAHQTSRSSPDTTAPTGSGLSNRFVNDNGIVTDTTASGTNC
         my_gempro.manual_seq_mapping(gene_to_seq_dict)
```

```
[2017-11-21 19:23] [ssbio.pipeline.gempro] INFO: Loaded in 2 sequences
```

.. automethod:: ssbio.pipeline.gempro.GEMPRO.manual$_{u}niprot_{m}apping$: $noindex$:

```
In [14]: manual_uniprot_dict = {'Rv1755c': 'P9WIA9', 'Rv2321c': 'P71891', 'Rv0619': 'Q79FY3', 'Rv0618
         my_gempro.manual_uniprot_mapping(manual_uniprot_dict)
         my_gempro.df_uniprot_metadata.tail(4)
```

```
HBox(children=(IntProgress(value=0, max=5), HTML(value='')))
```

```
[2017-11-21 19:23] [ssbio.pipeline.gempro] INFO: Completed manual ID mapping --> UniProt. See the "di
```

```
Out[14]: uniprot  reviewed gene_name            kegg          refseq     pfam  \
         gene
         Rv0619   Q79FY3     False     galTb             NaN             NaN  PF02744
         Rv1755c  P9WIA9     False      plcD             NaN             NaN  PF04185
         Rv2321c  P71891     False     rocD2  mtv:RVBD_2321c  WP_003411956.1  PF00202
         Rv2322c  P71890     False     rocD1  mtv:RVBD_2322c  WP_003411957.1  PF00202

                                                  description  entry_date  \
         gene
         Rv0619   Probable galactose-1-phosphate uridylyltransfe...  2017-10-25
         Rv1755c                                 Phospholipase C 4  2017-07-05
         Rv2321c  Probable ornithine aminotransferase (C-terminu...  2017-10-25
         Rv2322c  Probable ornithine aminotransferase (N-terminu...  2017-10-25

                  entry_version    seq_date  seq_version sequence_file metadata_file
         gene
         Rv0619              79  2004-07-05            1  Q79FY3.fasta    Q79FY3.xml
         Rv1755c             18  2014-04-16            1  P9WIA9.fasta    P9WIA9.xml
         Rv2321c            117  1997-02-01            1  P71891.fasta    P71891.xml
         Rv2322c            118  1997-02-01            1  P71890.fasta    P71890.xml
```

.. automethod:: ssbio.pipeline.gempro.GEMPRO.kegg$_{m}apping_{a}nd_{m}etadata$: $noindex$:

```
In [15]: # KEGG mapping of gene ids
         my_gempro.kegg_mapping_and_metadata(kegg_organism_code='mtu')
         print('Missing KEGG mapping: ', my_gempro.missing_kegg_mapping)
         my_gempro.df_kegg_metadata.head()
```

```
HBox(children=(IntProgress(value=0, max=661), HTML(value='')))
```

```
[2017-11-21 19:24] [root] WARNING: status is not ok with Not Found
[2017-11-21 19:24] [ssbio.databases.kegg] WARNING: mtu:Rv1755c: no sequence file available
[2017-11-21 19:24] [root] WARNING: status is not ok with Not Found
[2017-11-21 19:24] [ssbio.databases.kegg] WARNING: mtu:Rv1755c: no metadata file available
[2017-11-21 19:24] [root] WARNING: status is not ok with Not Found
[2017-11-21 19:24] [ssbio.databases.kegg] WARNING: mtu:Rv2233: no sequence file available
[2017-11-21 19:24] [root] WARNING: status is not ok with Not Found
[2017-11-21 19:24] [ssbio.databases.kegg] WARNING: mtu:Rv2233: no metadata file available
[2017-11-21 19:24] [ssbio.core.protein] WARNING: Rv2233: representative sequence does not match mappe
[2017-11-21 19:26] [root] WARNING: status is not ok with Not Found
[2017-11-21 19:26] [ssbio.databases.kegg] WARNING: mtu:Rv0619: no sequence file available
[2017-11-21 19:26] [root] WARNING: status is not ok with Not Found
[2017-11-21 19:26] [ssbio.databases.kegg] WARNING: mtu:Rv0619: no metadata file available
[2017-11-21 19:26] [root] WARNING: status is not ok with Not Found
[2017-11-21 19:26] [ssbio.databases.kegg] WARNING: mtu:Rv0618: no sequence file available
[2017-11-21 19:26] [root] WARNING: status is not ok with Not Found
[2017-11-21 19:26] [ssbio.databases.kegg] WARNING: mtu:Rv2321c: no sequence file available
[2017-11-21 19:26] [root] WARNING: status is not ok with Not Found
[2017-11-21 19:26] [ssbio.databases.kegg] WARNING: mtu:Rv2321c: no metadata file available
[2017-11-21 19:26] [root] WARNING: status is not ok with Not Found
[2017-11-21 19:26] [ssbio.databases.kegg] WARNING: mtu:Rv2322c: no sequence file available
[2017-11-21 19:26] [root] WARNING: status is not ok with Not Found
[2017-11-21 19:26] [ssbio.databases.kegg] WARNING: mtu:Rv2322c: no metadata file available


[2017-11-21 19:27] [ssbio.pipeline.gempro] INFO: 655/661: number of genes mapped to KEGG
[2017-11-21 19:27] [ssbio.pipeline.gempro] INFO: Completed ID mapping --> KEGG. See the "df_kegg_meta

Missing KEGG mapping:  ['Rv0618', 'Rv1755c', 'Rv0619', 'Rv2322c', 'Rv2233', 'Rv2321c']
```

Out[15]:

gene	kegg	refseq	uniprot	pdbs	sequence_file	metadata_file
Rv0013	mtu:Rv0013	YP_177615	P9WN35	NaN	mtu-Rv0013.faa	mtu-Rv0013.kegg
Rv0032	mtu:Rv0032	NP_214546	P9WQ85	NaN	mtu-Rv0032.faa	mtu-Rv0032.kegg
Rv0046c	mtu:Rv0046c	NP_214560	P9WKI1	1GR0	mtu-Rv0046c.faa	mtu-Rv0046c.kegg
Rv0066c	mtu:Rv0066c	NP_214580	O53611	5KVU	mtu-Rv0066c.faa	mtu-Rv0066c.kegg
Rv0069c	mtu:Rv0069c	NP_214583	P9WGT5	NaN	mtu-Rv0069c.faa	mtu-Rv0069c.kegg

.. automethod:: ssbio.pipeline.gempro.GEMPRO.uniprot$_m apping_a nd_m etadata$: $noindex$:

```
In [16]: # UniProt mapping
         my_gempro.uniprot_mapping_and_metadata(model_gene_source='TUBERCULIST_ID')
         print('Missing UniProt mapping: ', my_gempro.missing_uniprot_mapping)
         my_gempro.df_uniprot_metadata.head()

[2017-11-21 19:27] [root] INFO: getUserAgent: Begin
[2017-11-21 19:27] [root] INFO: getUserAgent: user_agent: EBI-Sample-Client/ (services.py; Python 3.6
[2017-11-21 19:27] [root] INFO: getUserAgent: End

HBox(children=(IntProgress(value=0, max=661), HTML(value='')))
```

```
[2017-11-21 19:34] [ssbio.pipeline.gempro] INFO: 589/661: number of genes mapped to UniProt
[2017-11-21 19:34] [ssbio.pipeline.gempro] INFO: Completed ID mapping --> UniProt. See the "df_unipro

Missing UniProt mapping:  ['Rv3565', 'Rv0812', 'Rv0649', 'Rv1164', 'Rv2458', 'Rv1512', 'Rv0511', 'Rv2
```

Out[16]:

gene	uniprot	reviewed	gene_name	kegg	refseq	pdbs	pfam	description	entry_date	entry_version	seq_date	seq_version	sequence_file	metadata_file
Rv0013	P9WN35	False	trpG	mtu:Rv0013	WP_003899773.1;YP_177615.1	NaN	PF00117	Anthranilate synthase component 2	2017-06-07	22	2014-04-16	1	P9WN35.fasta	P9WN35.xml
Rv0032	P9WQ85	False	bioF2	mtu:Rv0032	NP_214546.1;WP_003905217.1	NaN	PF00155	Putative 8-amino-7-oxononanoate synthase 2	2017-05-10	20	2014-04-16	1	P9WQ85.fasta	P9WQ85.xml
Rv0046c	P9WKI1	False	ino1	mtu:Rv0046c	NP_214560.1;WP_003902822.1	1GR0	PF01658	Inositol-3-phosphate synthase	2017-10-25	26	2014-04-16	1	P9WKI1.fasta	P9WKI1.xml
Rv0066c	O53611	False	icd2	mtu:Rv0066c;mtv:RVBD_0066c	NP_214580.1;WP_003899797.1	5KVU	PF03971	Probable isocitrate dehydrogenase [NADP] Icd2 ...	2017-10-25	128	1998-06-01	1	O53611.fasta	O53611.xml
Rv0069c	P9WGT5	False	sdaA	mtu:Rv0069c	NP_214583.1;WP_003400600.1	NaN	PF03313;PF03315	L-serine dehydratase	2017-06-07	20	2014-04-16	1	P9WGT5.fasta	P9WGT5.xml

.. automethod:: ssbio.pipeline.gempro.GEMPRO.set$_{r}epresentative_{s}equence$: $noindex$:

If you have mapped with both KEGG and UniProt mappers, then you can set a representative sequence for the gene using this function. If you used just one, this will just set that ID as representative.

- If any sequences or IDs were provided manually, these will be set as representative first.
- UniProt mappings override KEGG mappings except when KEGG mappings have PDBs associated with them and UniProt doesn't.

```
In [17]: # Set representative sequences
         my_gempro.set_representative_sequence()
         print('Missing a representative sequence: ', my_gempro.missing_representative_sequence)
         my_gempro.df_representative_sequences.head()
```

```
HBox(children=(IntProgress(value=0, max=661), HTML(value='')))


[2017-11-21 19:34] [ssbio.pipeline.gempro] INFO: 661/661: number of genes with a representative seque
[2017-11-21 19:34] [ssbio.pipeline.gempro] INFO: See the "df_representative_sequences" attribute for

Missing a representative sequence:  []
```

Out[17]:

gene	uniprot	kegg	pdbs	sequence_file	metadata_file
Rv0013	P9WN35	mtu:Rv0013	NaN	P9WN35.fasta	P9WN35.xml

```
Rv0032   P9WQ85                  mtu:Rv0032   NaN  P9WQ85.fasta    P9WQ85.xml
Rv0046c  P9WKI1                 mtu:Rv0046c  1GR0  P9WKI1.fasta    P9WKI1.xml
Rv0066c  O53611  mtu:Rv0066c;mtv:RVBD_0066c  5KVU  O53611.fasta    O53611.xml
Rv0069c  P9WGT5                 mtu:Rv0069c   NaN  P9WGT5.fasta    P9WGT5.xml
```

Mapping representative sequence –> structure

These are the ways to map sequence to structure:

1. Use the UniProt ID and their automatic mappings to the PDB
2. BLAST the sequence to the PDB
3. Make homology models or
4. Map to existing homology models

You can only utilize option #1 to map to PDBs if there is a mapped UniProt ID set in the representative sequence. If not, you'll have to BLAST your sequence to the PDB or make a homology model. You can also run both for maximum coverage.

Methods

.. automethod:: ssbio.pipeline.gempro.GEMPRO.map$_u niprot_t o_p db$: $noindex$:

```
In [18]: # Mapping using the PDBe best_structures service
         my_gempro.map_uniprot_to_pdb(seq_ident_cutoff=.3)
         my_gempro.df_pdb_ranking.head()
```

```
[2017-11-21 19:34] [ssbio.pipeline.gempro] INFO: Mapping UniProt IDs --> PDB IDs...
[2017-11-21 19:34] [root] INFO: getUserAgent: Begin
[2017-11-21 19:34] [root] INFO: getUserAgent: user_agent: EBI-Sample-Client/ (services.py; Python 3.6
[2017-11-21 19:34] [root] INFO: getUserAgent: End
```

```
HBox(children=(IntProgress(value=0, max=661), HTML(value='')))
```

```
[2017-11-21 19:37] [ssbio.pipeline.gempro] INFO: 184/661: number of genes with at least one experimer
[2017-11-21 19:37] [ssbio.pipeline.gempro] INFO: Completed UniProt --> best PDB mapping. See the "df_
```

```
Out[18]: pdb_id pdb_chain_id uniprot experimental_method  resolution  coverage  \
         gene
         Rv0046c   1gr0            A  P9WKI1   X-ray diffraction        1.95       1.0
         Rv0066c   5kvu            D  O53611   X-ray diffraction        2.66       1.0
         Rv0066c   5kvu            B  O53611   X-ray diffraction        2.66       1.0
         Rv0066c   5kvu            A  O53611   X-ray diffraction        2.66       1.0
         Rv0066c   5kvu            C  O53611   X-ray diffraction        2.66       1.0

                  start  end  unp_start  unp_end  rank
         gene
         Rv0046c      1  367          1      367     1
         Rv0066c      1  745          1      745     4
         Rv0066c      1  745          1      745     2
         Rv0066c      1  745          1      745     1
         Rv0066c      1  745          1      745     3
```

.. automethod:: ssbio.pipeline.gempro.GEMPRO.blast$_s eqs_t o_p db$: $noindex$:

```
In [19]: # Mapping using BLAST
         my_gempro.blast_seqs_to_pdb(all_genes=True, seq_ident_cutoff=.9, evalue=0.00001)
         my_gempro.df_pdb_blast.head(2)
```

```
HBox(children=(IntProgress(value=0, max=661), HTML(value='')))
```

```
[2017-11-21 19:55] [ssbio.pipeline.gempro] INFO: Completed sequence --> PDB BLAST. See the "df_pdb_bl
[2017-11-21 19:55] [ssbio.pipeline.gempro] INFO: 34: number of genes with additional structures added
```

```
Out[19]: pdb_id pdb_chain_id  hit_score  hit_evalue  hit_percent_similar  \
         gene
         Rv0126   4lxf            A     3218.0         0.0                  1.0
         Rv0126   4lxf            B     3218.0         0.0                  1.0

                 hit_percent_ident  hit_num_ident  hit_num_similar
         gene
         Rv0126                1.0            601              601
         Rv0126                1.0            601              601
```

.. automethod:: ssbio.pipeline.gempro.GEMPRO.get$_i tasser_m odels$: $noindex$:

```
In [20]: tb_homology_dir = '/home/nathan/projects_archive/homology_models/MTUBERCULOSIS/'

         ##### EXAMPLE SPECIFIC CODE #####
         # Needed to map to older IDs used in this example
         import pandas as pd
         import os.path as op
         old_gene_to_homology = pd.read_csv(op.join(tb_homology_dir, 'data/161031-old_gene_to_uniprot
         gene_to_uniprot = old_gene_to_homology.set_index('m_gene').to_dict()['u_uniprot_acc']
         my_gempro.get_itasser_models(homology_raw_dir=op.join(tb_homology_dir, 'raw'), custom_itasse
         ### END EXAMPLE SPECIFIC CODE ###

         # Organizing I-TASSER homology models
         my_gempro.get_itasser_models(homology_raw_dir=op.join(tb_homology_dir, 'raw'))
         my_gempro.df_homology_models.head()
```

```
HBox(children=(IntProgress(value=0, max=661), HTML(value='')))
```

```
[2017-11-21 19:56] [ssbio.pipeline.gempro] INFO: Completed copying of 435 I-TASSER models to GEM-PRO
```

```
HBox(children=(IntProgress(value=0, max=661), HTML(value='')))
```

```
[2017-11-21 19:56] [ssbio.pipeline.gempro] INFO: Completed copying of 9 I-TASSER models to GEM-PRO di
```

```
Out[20]: id     structure_file  model_date difficulty top_template_pdb  \
         gene
         Rv0013   P9WN35  P9WN35_model1.pdb  2017-11-22       easy             1i7s
         Rv0032   P9WQ85  P9WQ85_model1.pdb  2017-11-22       easy             3a2b
         Rv0066c  O53611  O53611_model1.pdb  2017-11-22       easy             1itw
         Rv0069c  P9WGT5  P9WGT5_model1.pdb  2017-11-22       easy             4rqo
         Rv0070c  P9WGI7  P9WGI7_model1.pdb  2017-11-22       easy             3h7f

                 top_template_chain  c_score  tm_score  tm_score_err  rmsd  rmsd_err
         gene
         Rv0013                   B    -0.53      0.65          0.13   6.8       4.0
         Rv0032                   A    -2.89      0.39          0.13  15.7       3.3
         Rv0066c                  A     1.91      0.99          0.04   4.1       2.8
         Rv0069c                  A     1.18      0.88          0.07   4.6       3.0
         Rv0070c                  B     1.80      0.97          0.05   3.3       2.3
```

.. automethod:: ssbio.pipeline.gempro.GEMPRO.get$_{m}anual_{h}omology_{m}odels$: $noindex$:

```
In [21]: homology_model_dict = {}
         my_gempro.get_manual_homology_models(homology_model_dict)
```

```
HBox(children=(IntProgress(value=0, max=661), HTML(value='')))
```

```
[2017-11-21 19:56] [ssbio.pipeline.gempro] INFO: Updated homology model information for 0 genes.
```

Downloading and ranking structures

Methods

.. automethod:: ssbio.pipeline.gempro.GEMPRO.pdb$_{d}ownloader_{a}nd_{m}etadata$: $noindex$:

Warning: Downloading all PDBs takes a while, since they are also parsed for metadata. You can skip this step and just set representative structures below if you want to minimize the number of PDBs downloaded.

```
In [22]: # Download all mapped PDBs and gather the metadata
         my_gempro.pdb_downloader_and_metadata()
         my_gempro.df_pdb_metadata.head(2)
```

```
HBox(children=(IntProgress(value=0, max=661), HTML(value='')))
```

```
[2017-11-21 20:02] [ssbio.pipeline.gempro] INFO: Updated PDB metadata dataframe. See the "df_pdb_meta
[2017-11-21 20:02] [ssbio.pipeline.gempro] INFO: Saved 991 structures total
```

```
Out[22]: chemicals                                  description  \
         gene
         Rv0046c               CAC;NAD;ZN  INOSITOL-3-PHOSPHATE SYNTHASE (E.C.5.5.1.4)
         Rv0066c  EDO;GOL;MLA;MLT;NAP;SIN      Isocitrate dehydrogenase (E.C.1.1.1.42)

                 experimental_method mapped_chains pdb_id  \
         gene
         Rv0046c   X-ray diffraction             A   1gr0
         Rv0066c   X-ray diffraction       A;B;C;D   5kvu

                                                      pdb_title  resolution  \
         gene
         Rv0046c  myo-inositol 1-phosphate synthase from Mycobac...        1.95
         Rv0066c  Crystal structure of isocitrate dehydrogenase-...        2.66

                 structure_file                                    taxonomy_name
         gene
         Rv0046c      1gr0.mmtf                          MYCOBACTERIUM TUBERCULOSIS
         Rv0066c      5kvu.mmtf  Mycobacterium tuberculosis (strain ATCC 25618 ...
```

.. automethod:: ssbio.pipeline.gempro.GEMPRO.set$_{r}epresentative_{s}tructure$: $noindex$:

```
In [23]: # Set representative structures
         my_gempro.set_representative_structure()
         my_gempro.df_representative_structures.head()
```

```
HBox(children=(IntProgress(value=0, max=661), HTML(value='')))
```

```
[2017-11-21 20:02] [ssbio.core.protein] WARNING: Rv0432: no structures meet quality checks
[2017-11-21 20:02] [ssbio.core.protein] WARNING: Rv1286: no structures meet quality checks
[2017-11-21 20:02] [ssbio.core.protein] WARNING: Rv2933: no structures meet quality checks
[2017-11-21 20:02] [ssbio.core.protein] WARNING: Rv2945c: no structures meet quality checks
[2017-11-21 20:02] [ssbio.core.protein] WARNING: Rv2941: no structures meet quality checks
[2017-11-21 20:02] [ssbio.core.protein] WARNING: Rv2495c: no structures meet quality checks
[2017-11-21 20:03] [ssbio.core.protein] WARNING: Rv2987c: no structures meet quality checks
[2017-11-21 20:04] [ssbio.core.protein] WARNING: Rv1653: no structures meet quality checks
[2017-11-21 20:04] [ssbio.core.protein] WARNING: Rv3800c: no structures meet quality checks
[2017-11-21 20:04] [ssbio.core.protein] WARNING: Rv2498c: no structures meet quality checks
[2017-11-21 20:04] [ssbio.core.protein] WARNING: Rv1885c: no structures meet quality checks
[2017-11-21 20:04] [ssbio.core.protein] WARNING: Rv3601c: no structures meet quality checks
[2017-11-21 20:05] [ssbio.core.protein] WARNING: Rv3330: no structures meet quality checks
[2017-11-21 20:05] [ssbio.core.protein] WARNING: Rv3793: no structures meet quality checks
[2017-11-21 20:05] [ssbio.core.protein] WARNING: Rv1625c: no structures meet quality checks


[2017-11-21 20:05] [ssbio.pipeline.gempro] INFO: 590/661: number of genes with a representative struc
[2017-11-21 20:05] [ssbio.pipeline.gempro] INFO: See the "df_representative_structures" attribute for
```

Out[23]:

gene	id	is_experimental	file_type	structure_file
Rv0013	REP-P9WN35	False	pdb	P9WN35_model1-X_clean.pdb
Rv0032	REP-P9WQ85	False	pdb	P9WQ85_model1-X_clean.pdb
Rv0046c	REP-1gr0	True	pdb	1gr0-A_clean.pdb
Rv0066c	REP-5kvu	True	pdb	5kvu-A_clean.pdb
Rv0069c	REP-P9WGT5	False	pdb	P9WGT5_model1-X_clean.pdb

In [24]:
```
# Looking at the information saved within a gene
my_gempro.genes.get_by_id('Rv1295').protein.representative_structure
my_gempro.genes.get_by_id('Rv1295').protein.representative_structure.get_dict()
```

Out[24]:
```
<StructProp REP-2d1f at 0x7f72a72dcf98>
```

Out[24]:
```
{'_structure_dir': '/tmp/mtuberculosis_gp_atlas/genes/Rv1295/Rv1295_protein/structures',
 'chains': [<ChainProp A at 0x7f72a72dd828>],
 'date': None,
 'description': 'Threonine synthase (E.C.4.2.3.1)',
 'file_type': 'pdb',
 'id': 'REP-2d1f',
 'is_experimental': True,
 'mapped_chains': ['A'],
 'notes': {},
 'original_structure_id': '2d1f',
 'resolution': 2.5,
 'structure_file': '2d1f-A_clean.pdb',
 'taxonomy_name': 'Mycobacterium tuberculosis'}
```

Creating homology models

For those proteins with no representative structure, we can create homology models for them. `ssbio` contains some built in functions for easily running I-TASSER locally or on machines with `SLURM` (ie. on NERSC) or `Torque` job scheduling.

You can load in I-TASSER models once they complete using the `get_itasser_models` later.

Info: Homology modeling can take a long time - about 24-72 hours per protein (highly dependent on the sequence

length, as well as if there are available templates).

Methods

.. automethod:: ssbio.pipeline.gempro.GEMPRO.prep$_{i}tasser_{m}odels$: $noindex$:

```
In [25]: # Prep I-TASSER model folders
         my_gempro.prep_itasser_modeling('~/software/I-TASSER4.4', '~/software/ITLIB/', runtype='loca
```

```
[2017-11-21 20:05] [ssbio.protein.structure.homology.itasser.itasserprep] WARNING: Rv2934: I-TASSER r
[2017-11-21 20:05] [ssbio.protein.structure.homology.itasser.itasserprep] WARNING: Rv2932: I-TASSER r
[2017-11-21 20:05] [ssbio.protein.structure.homology.itasser.itasserprep] WARNING: Rv2933: I-TASSER r
[2017-11-21 20:05] [ssbio.protein.structure.homology.itasser.itasserprep] WARNING: Rv2931: I-TASSER r
[2017-11-21 20:05] [ssbio.protein.structure.homology.itasser.itasserprep] WARNING: Rv2380c: I-TASSER
[2017-11-21 20:05] [ssbio.protein.structure.homology.itasser.itasserprep] WARNING: Rv3859c: I-TASSER
[2017-11-21 20:05] [ssbio.protein.structure.homology.itasser.itasserprep] WARNING: Rv2476c: I-TASSER
[2017-11-21 20:05] [ssbio.protein.structure.homology.itasser.itasserprep] WARNING: Rv3800c: I-TASSER
[2017-11-21 20:05] [ssbio.protein.structure.homology.itasser.itasserprep] WARNING: Rv0107c: I-TASSER
[2017-11-21 20:05] [ssbio.protein.structure.homology.itasser.itasserprep] WARNING: Rv2940c: I-TASSER
[2017-11-21 20:05] [ssbio.protein.structure.homology.itasser.itasserprep] WARNING: Rv1662: I-TASSER r
[2017-11-21 20:05] [ssbio.protein.structure.homology.itasser.itasserprep] WARNING: Rv2524c: I-TASSER
[2017-11-21 20:05] [ssbio.pipeline.gempro] INFO: Prepared I-TASSER modeling folders for 71 genes in
```

Saving your GEM-PRO

Finally, you can save your GEM-PRO as a `JSON` or `pickle` file, so you don't have to run the pipeline again.

For most functions, if you rerun them, they will check for existing results saved as files. The only function that would take a long time is setting the representative structure, as they are each rechecked and cleaned. This is where saving helps!

Warning: Saving in JSON format is still experimental. For a full GEM-PRO with sequences & structures, depending on the number of genes, saving can take >5 minutes.

.. automethod:: ssbio.pipeline.gempro.GEMPRO.save$_{p}ickle$: $noindex$:

```
In [26]: import os.path as op
         my_gempro.save_pickle(op.join(my_gempro.model_dir, '{}.pckl'.format(my_gempro.id)))
```

.. automethod:: ssbio.pipeline.gempro.GEMPRO.save$_{j}son$: $noindex$:

```
In [27]: import os.path as op
         my_gempro.save_json(op.join(my_gempro.model_dir, '{}.json'.format(my_gempro.id)), compressio
```

```
[2017-11-21 20:05] [root] WARNING: json-tricks: numpy scalar serialization is experimental and may wo
[2017-11-21 20:05] [ssbio.core.io] INFO: Saved <class 'ssbio.pipeline.gempro.GEMPRO'> (id: mtubercul
```

Loading a saved GEM-PRO

```
In [28]: # Loading a pickle file
         import pickle
         with open('/tmp/mtuberculosis_gp_atlas/model/mtuberculosis_gp_atlas.pckl', 'rb') as f:
             my_saved_gempro = pickle.load(f)
```

```
In [29]: my_saved_gempro.genes[0].__json_encode__()
```

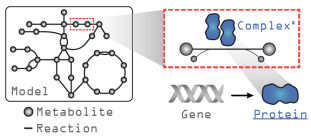
```
Out[29]: {'_root_dir': '/tmp/mtuberculosis_gp_atlas/genes',
          'id': 'Rv0417',
          'name': 'thiG',
          'notes': OrderedDict([('original_bigg_ids', ['Rv0417'])]),
          'protein': <Protein Rv0417 at 0x7f72ac1c8eb8>}

In [30]: my_saved_gempro.genes[0].protein.__json_encode__()

Out[30]: {'_root_dir': '/tmp/mtuberculosis_gp_atlas/genes/Rv0417',
          'description': None,
          'id': 'Rv0417',
          'notes': {},
          'pdb_file_type': 'mmtf',
          'representative_chain': 'X',
          'representative_chain_seq_coverage': 100.0,
          'representative_sequence': <UniProtProp P9WG73 at 0x7f72ad9a6080>,
          'representative_structure': <StructProp REP-P9WG73 at 0x7f7297766198>,
          'sequence_alignments': [<<class 'Bio.Align.MultipleSeqAlignment'> instance (2 records of le
          'sequences': [<KEGGProp mtu:Rv0417 at 0x7f72a6c55a58>,
           <UniProtProp P9WG73 at 0x7f72ad9a6080>],
          'structure_alignments': [],
          'structures': [<ITASSERProp P9WG73 at 0x7f7297766da0>,
           <StructProp REP-P9WG73 at 0x7f7297766198>]}

In [31]: # Loading a JSON file
         import ssbio.core.io
         my_saved_gempro = ssbio.core.io.load_json('/tmp/mtuberculosis_gp_atlas/model/mtuberculosis_g

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-31-1a7ccd1e60af> in <module>()
      1 # Loading a JSON file
      2 import ssbio.core.io
----> 3 my_saved_gempro = ssbio.core.io.load_json('/tmp/mtuberculosis_gp_atlas/model/mtuberculosis_gp

/mnt/projects/ssbio/ssbio/core/io.py in load_json(file, new_root_dir, decompression)
     23     else:
     24         with open(file, 'r') as f:
---> 25             my_object = load(f, decompression=decompression)
     26     if new_root_dir:
     27         my_object.root_dir = new_root_dir

~/anaconda3/lib/python3.6/site-packages/json_tricks-3.8.0-py3.6.egg/json_tricks/np.py in load(fp, pre
    125         return nonp.load(fp, preserve_order=preserve_order, ignore_comments=ignore_comments, c
    126                 obj_pairs_hooks=obj_pairs_hooks, extra_obj_pairs_hooks=extra_obj_pairs_hooks,
--> 127             allow_duplicates=allow_duplicates, **jsonkwargs)    128
    129

~/anaconda3/lib/python3.6/site-packages/json_tricks-3.8.0-py3.6.egg/json_tricks/nonp.py in load(fp, p
    191         return loads(string, preserve_order=preserve_order, ignore_comments=ignore_comments, c
    192                 obj_pairs_hooks=obj_pairs_hooks, extra_obj_pairs_hooks=extra_obj_pairs_hooks,
--> 193             allow_duplicates=allow_duplicates, **jsonkwargs)    194
    195

~/anaconda3/lib/python3.6/site-packages/json_tricks-3.8.0-py3.6.egg/json_tricks/nonp.py in loads(stri
    166         hooks = tuple(extra_obj_pairs_hooks) + obj_pairs_hooks
    167         hook = TricksPairHook(ordered=preserve_order, obj_pairs_hooks=hooks, allow_duplicates
--> 168         return json_loads(string, object_pairs_hook=hook, **jsonkwargs)
    169
    170
```

```
~/anaconda3/lib/python3.6/json/__init__.py in loads(s, encoding, cls, object_hook, parse_float, parse
    365     if parse_constant is not None:
    366         kw['parse_constant'] = parse_constant
--> 367     return cls(**kw).decode(s)

~/anaconda3/lib/python3.6/json/decoder.py in decode(self, s, _w)
    337
    338         """
--> 339         obj, end = self.raw_decode(s, idx=_w(s, 0).end())
    340         end = _w(s, end).end()
    341         if end != len(s):

~/anaconda3/lib/python3.6/json/decoder.py in raw_decode(self, s, idx)
    353         """
    354         try:
--> 355             obj, end = self.scan_once(s, idx)
    356         except StopIteration as err:
    357             raise JSONDecodeError("Expecting value", s, err.value) from None

~/anaconda3/lib/python3.6/site-packages/json_tricks-3.8.0-py3.6.egg/json_tricks/decoders.py in __cal
     39                 map = self.map_type(pairs)
     40                 for hook in self.obj_pairs_hooks:
---> 41                         map = hook(map)
     42                 return map
     43

~/anaconda3/lib/python3.6/site-packages/json_tricks-3.8.0-py3.6.egg/json_tricks/decoders.py in __cal
    149                                 obj.__json_decode__(**attrs)
    150                         else:
--> 151                                 obj.__dict__ = dict(attrs)
    152                         return  obj
    153                 return dct

TypeError: 'NoneType' object is not iterable
```

5.2.3 Features

- Automated mapping of sequence IDs
- Consolidating sequence IDs and setting a representative sequence
- Mapping of representative sequence –> structures
- Preparation of files for homology modeling (currently for I-TASSER)
- Running QC/QA on structures and setting a representative structure
- Automation of protein sequence and structure property calculation
- Creation of Pandas DataFrame summaries directly from downloaded metadata

5.2.4 COBRApy model additions



Let's take a look at a GEM loaded with *ssbio* and what additions exist compared to a GEM loaded with COBRApy. In the figure above, the text in grey indicates objects that exist in a `COBRApy Model` object, and in blue, the attributes added when loading with *ssbio*. Please note that the `Complex` object is still under development and currently non-functional.

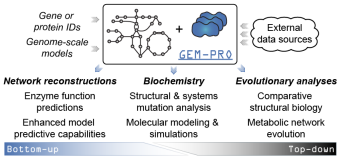
COBRApy

Under construction...

ssbio

Under construction...

5.2.5 Use cases



When would you create or use a GEM-PRO? The added context of manually curated network interactions to protein structures enables different scales of analyses. For instance...

From the "top-down":

- Global non-variant properties of protein structures such as the distribution of fold types can be compared within or between organisms[1],[2],[3], elucidating adaptations that are reflected in the structural proteome.
- Multi-strain modelling techniques ([10],[11],[12]) would allow strain-specific changes to be investigated at the molecular level, potentially explaining phenotypic differences or strain adaptations to certain environments.

[1] Zhang Y, Thiele I, Weekes D, Li Z, Jaroszewski L, Ginalski K, et al. Three-dimensional structural view of the central metabolic network of Thermotoga maritima. Science. 2009 Sep 18;325(5947):1544–9. Available from: http://dx.doi.org/10.1126/science.1174671

[2] Brunk E, Mih N, Monk J, Zhang Z, O'Brien EJ, Bliven SE, et al. Systems biology of the structural proteome. BMC Syst Biol. 2016;10: 26. doi:10.1186/s12918-016-0271-6

[3] Monk JM, Lloyd CJ, Brunk E, Mih N, Sastry A, King Z, et al. iML1515, a knowledgebase that computes Escherichia coli traits. Nat Biotechnol. 2017;35: 904–908. doi:10.1038/nbt.3956

[10] Bosi, E, Monk, JM, Aziz, RK, Fondi, M, Nizet, V, & Palsson, BO. (2016). Comparative genome-scale modelling of Staphylococcus aureus strains identifies strain-specific metabolic capabilities linked to pathogenicity. Proceedings of the National Academy of Sciences of the United States of America, 113/26: E3801–9. DOI: 10.1073/pnas.1523199113

[11] Monk, JM, Koza, A, Campodonico, MA, Machado, D, Seoane, JM, Palsson, BO, Herrgård, MJ, et al. (2016). Multi-omics Quantification of Species Variation of Escherichia coli Links Molecular Features with Strain Phenotypes. Cell systems, 3/3: 238–51.e12. DOI: 10.1016/j.cels.2016.08.013

[12] Ong, WK, Vu, TT, Lovendahl, KN, Llull, JM, Serres, MH, Romine, MF, & Reed, JL. (2014). Comparisons of Shewanella strains based on genome annotations, modeling, and experiments. BMC systems biology, 8: 31. DOI: 10.1186/1752-0509-8-31

From the "bottom-up"

- Structural properties predicted from sequence or calculated from structure can be utilized to enhance model predictive capabilities[4],[5],[6],[7],[8],[9]

5.2.6 File organization

Files such as sequences, structures, alignment files, and property calculation outputs can optionally be cached on a user's disk to minimize calls to web services, limit recalculations, and provide direct inputs to common sequence and structure algorithms which often require local copies of the data. For a GEM-PRO project, files are organized in the following fashion once a root directory and project name are set:

```
<ROOT_DIR>
└── <PROJECT_NAME>
        ├── data  # General directory for pipeline outputs
        ├── model  # SBML and GEM-PRO models are stored in this directory
        └── genes  # Per gene information
            └── <gene_id1>  # Specific gene directory
                └── <protein_id1>  # Protein directory
                    ├── sequences  # Protein sequence files, alignments, etc.
                    └── structures  # Protein structure files, calculations, etc.
```

5.2.7 Further reading

For examples in which structures have been integrated into a GEM and utilized on a genome-scale, please see the following:

5.2.8 References

5.3 The Protein Class

5.3.1 Introduction

This section will give an overview of the methods that can be executed for the Protein class, which is a basic representation of a protein by a collection of amino acid sequences and 3D structures.

[4] Chang RL, Xie L, Xie L, Bourne PE, Palsson BØ. Drug off-target effects predicted using structural analysis in the context of a metabolic network model. PLoS Comput Biol. 2010 Sep 23;6(9):e1000938. Available from: http://dx.doi.org/10.1371/journal.pcbi.1000938

[5] Chang RL, Andrews K, Kim D, Li Z, Godzik A, Palsson BO. Structural systems biology evaluation of metabolic thermotolerance in Escherichia coli. Science. 2013 Jun 7;340(6137):1220–3. Available from: http://dx.doi.org/10.1126/science.1234012

[6] Chang RL, Xie L, Bourne PE, Palsson BO. Antibacterial mechanisms identified through structural systems pharmacology. BMC Syst Biol. 2013 Oct 10;7:102. Available from: http://dx.doi.org/10.1186/1752-0509-7-102

[7] Mih N, Brunk E, Bordbar A, Palsson BO. A Multi-scale Computational Platform to Mechanistically Assess the Effect of Genetic Variation on Drug Responses in Human Erythrocyte Metabolism. PLoS Comput Biol. 2016;12: e1005039. doi:10.1371/journal.pcbi.1005039

[8] Chen K, Gao Y, Mih N, O'Brien EJ, Yang L, Palsson BO. Thermosensitivity of growth is determined by chaperone-mediated proteome reallocation. Proceedings of the National Academy of Sciences. 2017;114: 11548–11553. doi:10.1073/pnas.1705524114

[9] Yang L, Mih N, Yurkovich JT, Park JH, Seo S, Kim D, et al. Multi-scale model of the proteomic and metabolic consequences of reactive oxygen species. bioRxiv. 2017. p. 227892. doi:10.1101/227892

5.3.2 Tutorials

Protein - Structure Mapping, Alignments, and Visualization

This notebook gives an example of how to **map a single protein sequence to its structure**, along with conducting sequence alignments and visualizing the mutations.

Input: Protein ID + amino acid sequence + mutated sequence(s)

Output: Representative protein structure, sequence alignments, and visualization of mutations

---

Imports

```
In [1]: import sys
        import logging
```

```
In [2]: # Import the Protein class
        from ssbio.core.protein import Protein
```

```
In [3]: # Printing multiple outputs per cell
        from IPython.core.interactiveshell import InteractiveShell
        InteractiveShell.ast_node_interactivity = "all"
```

Logging

Set the logging level in `logger.setLevel(logging.<LEVEL_HERE>)` to specify how verbose you want the pipeline to be. Debug is most verbose.

- `CRITICAL`
  - Only really important messages shown
- `ERROR`
  - Major errors
- `WARNING`
  - Warnings that don't affect running of the pipeline
- `INFO` (default)
  - Info such as the number of structures mapped per gene
- `DEBUG`
  - Really detailed information that will print out a lot of stuff

Warning: `DEBUG` mode prints out a large amount of information, especially if you have a lot of genes. This may stall your notebook!

```
In [4]: # Create logger
        logger = logging.getLogger()
        logger.setLevel(logging.INFO)  # SET YOUR LOGGING LEVEL HERE #
```

```
In [5]: # Other logger stuff for Jupyter notebooks
        handler = logging.StreamHandler(sys.stderr)
        formatter = logging.Formatter('[%(asctime)s] [%(name)s] %(levelname)s: %(message)s', datefmt=
        handler.setFormatter(formatter)
        logger.handlers = [handler]
```

Initialization of the project

Set these three things:

- ROOT_DIR
  - The directory where a folder named after your PROTEIN_ID will be created
- PROTEIN_ID
  - Your protein ID
- PROTEIN_SEQ
  - Your protein sequence

A directory will be created in ROOT_DIR with your PROTEIN_ID name. The folders are organized like so:

```
ROOT_DIR
└── PROTEIN_ID
    ├── sequences  # Protein sequence files, alignments, etc.
    └── structures  # Protein structure files, calculations, etc.
```

```
In [6]: # SET FOLDERS AND DATA HERE
        import tempfile
        ROOT_DIR = tempfile.gettempdir()

        PROTEIN_ID = 'SRR1753782_00918'
        PROTEIN_SEQ = 'MSKQQIGVVGMAVMGRNLALNIESRGYTVSVFNRSREKTEEVIAENPGKKLVPYYTVKEFVESLETPRRILLMVKAG
```

class ssbio.core.protein.**Protein**(*ident*, *description=None*, *root_dir=None*, *pdb_file_type='mmtf'*)

Store information on a protein that represents the translated unit of a gene.

The main utilities of this class are to:

1. Load, parse, and store the same (ie. from different database sources) or similar (ie. from different strains) protein sequences as SeqProp objects in the sequences attribute
2. Load, parse, and store multiple experimental or predicted protein structures as StructProp objects in the structures attribute
3. Set a single representative sequence and structure
4. Calculate, store, and access pairwise sequence alignments to the representative sequence or structure
5. Provide summaries of alignments and mutations seen
6. Map between residue numbers of sequences and structures

Parameters

- **ident** (*str*) – Unique identifier for this protein
- **description** (*str*) – Optional description for this protein

- **root_dir** (*str*) – Path to where the folder named by this protein's ID will be created. Default is current working directory.
- **pdb_file_type** (*str*) – pdb, pdb.gz, mmcif, cif, cif.gz, xml.gz, mmtf, mmtf.gz - choose a file type for files downloaded from the PDB

Todo:

- Implement structural alignment objects

```
In [7]: # Create the Protein object
        my_protein = Protein(ident=PROTEIN_ID, root_dir=ROOT_DIR, pdb_file_type='mmtf')
```

```
In [8]: # Load the protein sequence
        # This sets the loaded sequence as the representative one
        my_protein.load_manual_sequence(seq=PROTEIN_SEQ, ident='WT', write_fasta_file=True, set_as_re
```

```
Out[8]: <SeqProp WT at 0x7ff5fa89fac8>
```

Mapping sequence –> structure

Since the sequence has been provided, we just need to BLAST it to the PDB.

Note: These methods do not download any 3D structure files.

Methods

Protein.**blast_representative_sequence_to_pdb**(*seq_ident_cutoff=0*, *evalue=0.0001*, *display_link=False*, *outdir=None*, *force_rerun=False*)

BLAST the representative protein sequence to the PDB. Saves a raw BLAST result file (XML file).

Parameters

- **seq_ident_cutoff** (*float, optional*) – Cutoff results based on percent coverage (in decimal form)
- **evalue** (*float, optional*) – Cutoff for the E-value - filters for significant hits. 0.001 is liberal, 0.0001 is stringent (default).
- **display_link** (*bool, optional*) – Set to True if links to the HTML results should be displayed
- **outdir** (*str*) – Path to output directory of downloaded XML files, must be set if protein directory was not initialized
- **force_rerun** (*bool, optional*) – If existing BLAST results should not be used, set to True. Default is False.

Returns List of new PDBProp objects added to the structures attribute

Return type list

```
In [9]: # Mapping using BLAST
        my_protein.blast_representative_sequence_to_pdb(seq_ident_cutoff=0.9, evalue=0.00001)
        my_protein.df_pdb_blast.head()
```

```
Out[9]: ['2zyd', '2zya', '3fwn', '2zyg']

Out[9]: pdb_chain_id  hit_score  hit_evalue  hit_percent_similar  \
        pdb_id
        2zya              A     2319.0         0.0             0.987179
        2zya              B     2319.0         0.0             0.987179
        2zyd              A     2319.0         0.0             0.987179
        2zyd              B     2319.0         0.0             0.987179
        2zyg              A     2284.0         0.0             0.982906

                hit_percent_ident  hit_num_ident  hit_num_similar
        pdb_id
        2zya             0.963675            451              462
        2zya             0.963675            451              462
        2zyd             0.963675            451              462
        2zyd             0.963675            451              462
        2zyg             0.950855            445              460
```

Downloading and ranking structures

Methods

Protein.**pdb_downloader_and_metadata**(*outdir=None*, *pdb_file_type=None*, *force_rerun=False*)

Download ALL mapped experimental structures to the protein structures directory.

Parameters

- **outdir** (*str*) – Path to output directory, if protein structures directory not set or other output directory is desired
- **pdb_file_type** (*str*) – Type of PDB file to download, if not already set or other format is desired
- **force_rerun** (*bool*) – If files should be re-downloaded if they already exist

Returns List of PDB IDs that were downloaded

Return type list

Todo:

- Parse mmtf or PDB file for header information, rather than always getting the cif file for header info

---

Warning: Downloading all PDBs takes a while, since they are also parsed for metadata. You can skip this step and just set representative structures below if you want to minimize the number of PDBs downloaded.

```
In [10]: # Download all mapped PDBs and gather the metadata
         my_protein.pdb_downloader_and_metadata()
         my_protein.df_pdb_metadata.head(2)

Out[10]: ['2zyd', '2zya', '3fwn', '2zyg']

Out[10]: pdb_title  \
         pdb_id
         2zya    Dimeric 6-phosphogluconate dehydrogenase compl...
         2zyd    Dimeric 6-phosphogluconate dehydrogenase compl...
```

```
                                                    description experimental_method  \
pdb_id
2zya    6-phosphogluconate dehydrogenase, decarboxylat...   X-RAY DIFFRACTION
2zyd    6-phosphogluconate dehydrogenase, decarboxylat...   X-RAY DIFFRACTION

       mapped_chains  resolution chemicals     taxonomy_name structure_file
pdb_id
2zya             A;B         1.6       6PG  Escherichia coli      2zya.mmtf
2zyd             A;B         1.5       GLO  Escherichia coli      2zyd.mmtf
```

Protein.**set_representative_structure**(*seq_outdir=None*, *struct_outdir=None*, *pdb_file_type=None*, *engine='needle'*, *always_use_homology=False*, *rez_cutoff=0.0*, *seq_ident_cutoff=0.5*, *allow_missing_on_termini=0.2*, *allow_mutants=True*, *allow_deletions=False*, *allow_insertions=False*, *allow_unresolved=True*, *clean=True*, *keep_chemicals=None*, *force_rerun=False*)

Set a representative structure from a structure in the structures attribute.

Each gene can have a combination of the following, which will be analyzed to set a representative structure.

- Homology model(s)
- Ranked PDBs
- BLASTed PDBs

If the `always_use_homology` flag is true, homology models are always set as representative when they exist. If there are multiple homology models, we rank by the percent sequence coverage.

Parameters

- **seq_outdir** (*str*) – Path to output directory of sequence alignment files, must be set if Protein directory was not created initially
- **struct_outdir** (*str*) – Path to output directory of structure files, must be set if Protein directory was not created initially
- **pdb_file_type** (*str*) – `pdb`, `pdb.gz`, `mmcif`, `cif`, `cif.gz`, `xml.gz`, `mmtf`, `mmtf.gz` - choose a file type for files downloaded from the PDB
- **engine** (*str*) – `biopython` or `needle` - which pairwise alignment program to use. `needle` is the standard EMBOSS tool to run pairwise alignments. `biopython` is Biopython's implementation of needle. Results can differ!
- **always_use_homology** (*bool*) – If homology models should always be set as the representative structure
- **rez_cutoff** (*float*) – Resolution cutoff, in Angstroms (only if experimental structure)
- **seq_ident_cutoff** (*float*) – Percent sequence identity cutoff, in decimal form
- **allow_missing_on_termini** (*float*) – Percentage of the total length of the reference sequence which will be ignored when checking for modifications. Example: if 0.1, and reference sequence is 100 AA, then only residues 5 to 95 will be checked for modifications.
- **allow_mutants** (*bool*) – If mutations should be allowed or checked for
- **allow_deletions** (*bool*) – If deletions should be allowed or checked for
- **allow_insertions** (*bool*) – If insertions should be allowed or checked for

- **allow_unresolved** (*bool*) – If unresolved residues should be allowed or checked for
- **clean** (*bool*) – If structure should be cleaned
- **keep_chemicals** (*str, list*) – Keep specified chemical names if structure is to be cleaned
- **force_rerun** (*bool*) – If sequence to structure alignment should be rerun

Returns Representative structure from the list of structures. This is a not a map to the original structure, it is copied and optionally cleaned from the original one.

Return type *StructProp*

```
In [11]: # Set representative structures
         my_protein.set_representative_structure()
```

```
Out[11]: <StructProp REP-2zyd at 0x7ff5f9ac6828>
```

Loading and aligning new sequences

You can load additional sequences into this protein object and align them to the representative sequence.

Methods

Protein.**load_manual_sequence**(*seq*, *ident=None*, *write_fasta_file=False*, *outdir=None*, *set_as_representative=False*, *force_rewrite=False*)

Load a manual sequence given as a string and optionally set it as the representative sequence. Also store it in the sequences attribute.

Parameters

- **seq** (*str, Seq, SeqRecord*) – Sequence string, Biopython Seq or SeqRecord object
- **ident** (*str*) – Optional identifier for the sequence, required if seq is a string. Also will override existing IDs in Seq or SeqRecord objects if set.
- **write_fasta_file** (*bool*) – If this sequence should be written out to a FASTA file
- **outdir** (*str*) – Path to output directory
- **set_as_representative** (*bool*) – If this sequence should be set as the representative one
- **force_rewrite** (*bool*) – If the FASTA file should be overwritten if it already exists

Returns Sequence that was loaded into the `sequences` attribute

Return type *SeqProp*

```
In [12]: # Input your mutated sequence and load it
         mutated_protein1_id = 'N17P_SNP'
         mutated_protein1_seq = 'MSKQQIGVVGMAVMGRPLALNIESRGYTVSVFNRSREKTEEVIAENPGKKLVPYYTVKEFVESLETPR

         my_protein.load_manual_sequence(ident=mutated_protein1_id, seq=mutated_protein1_seq)
```

```
Out[12]: <SeqProp N17P_SNP at 0x7ff5f90be2e8>
```

```
In [13]: # Input another mutated sequence and load it
         mutated_protein2_id = 'Q4S_N17P_SNP'
         mutated_protein2_seq = 'MSKSQIGVVGMAVMGRPLALNIESRGYTVSVFNRSREKTEEVIAENPGKKLVPYYTVKEFVESLETPR

         my_protein.load_manual_sequence(ident=mutated_protein2_id, seq=mutated_protein2_seq)
```

```
Out[13]: <SeqProp Q4S_N17P_SNP at 0x7ff5f8c9fb38>
```

Protein.**pairwise_align_sequences_to_representative**(*gapopen=10*, *gapextend=0.5*, *outdir=None*, *engine='needle'*, *parse=True*, *force_rerun=False*)

Pairwise all sequences in the sequences attribute to the representative sequence. Stores the alignments in the `sequence_alignments` DictList attribute.

Parameters

- **gapopen** (*int*) – Only for `engine='needle'` - Gap open penalty is the score taken away when a gap is created
- **gapextend** (*float*) – Only for `engine='needle'` - Gap extension penalty is added to the standard gap penalty for each base or residue in the gap
- **outdir** (*str*) – Only for `engine='needle'` - Path to output directory. Default is the protein sequence directory.
- **engine** (*str*) – `biopython` or `needle` - which pairwise alignment program to use. `needle` is the standard EMBOSS tool to run pairwise alignments. `biopython` is Biopython's implementation of needle. Results can differ!
- **parse** (*bool*) – Store locations of mutations, insertions, and deletions in the alignment object (as an annotation)
- **force_rerun** (*bool*) – Only for `engine='needle'` - Default False, set to True if you want to rerun the alignment if outfile exists.

```
In [14]: # Conduct pairwise sequence alignments
         my_protein.pairwise_align_sequences_to_representative()
```

```
In [15]: # View IDs of all sequence alignments
         [x.id for x in my_protein.sequence_alignments]

         # View the stored information for one of the alignments
         my_alignment = my_protein.sequence_alignments.get_by_id('SRR1753782_00918_N17P_SNP')
         my_alignment.annotations
         str(my_alignment[0].seq)
         str(my_alignment[1].seq)
```

```
Out[15]: ['WT_2zyd-A',
          'WT_2zyd-B',
          'SRR1753782_00918_N17P_SNP',
          'SRR1753782_00918_Q4S_N17P_SNP']
```

```
Out[15]: {'a_seq': 'WT',
          'b_seq': 'N17P_SNP',
          'deletions': [],
          'gaps': 0,
          'identity': 467,
          'insertions': [],
          'mutations': [('N', 17, 'P')],
          'percent_gaps': 0.0,
          'percent_identity': 99.8,
          'percent_similarity': 99.8,
          'score': 2381.0,
          'similarity': 467,
          'ssbio_type': 'seqalign'}
```

```
Out[15]: 'MSKQQIGVVGMAVMGRNLALNIESRGYTVSVFNRSREKTEEVIAENPGKKLVPYYTVKEFVESLETPRRILLMVKAGAGTDAAIDSLKPYI
```

```
Out[15]: 'MSKQQIGVVGMAVMGRPLALNIESRGYTVSVFNRSREKTEEVIAENPGKKLVPYYTVKEFVESLETPRRILLMVKAGAGTDAAIDSLKPYI
```

Protein.**sequence_mutation_summary**(*alignment_ids=None*, *alignment_type=None*)

Summarize all mutations found in the sequence_alignments attribute.

Returns 2 dictionaries, single_counter and fingerprint_counter.

single_counter: Dictionary of `{point mutation: list of genes/strains}` Example:

```
{
    ('A', 24, 'V'): ['Strain1', 'Strain2', 'Strain4'],
    ('R', 33, 'T'): ['Strain2']
}
```

Here, we report which genes/strains have the single point mutation.

fingerprint_counter: Dictionary of `{mutation group: list of genes/strains}` Example:

```
{
    (('A', 24, 'V'), ('R', 33, 'T')): ['Strain2'],
    (('A', 24, 'V')): ['Strain1', 'Strain4']
}
```

Here, we report which genes/strains have the specific combinations (or "fingerprints") of point mutations

Parameters

- **alignment_ids** (*str, list*) – Specified alignment ID or IDs to use
- **alignment_type** (*str*) – Specified alignment type contained in the `annotation` field of an alignment object, `seqalign` or `structalign` are the current types.

Returns single_counter, fingerprint_counter

Return type dict, dict

```
In [16]: # Summarize all the mutations in all sequence alignments
         s,f = my_protein.sequence_mutation_summary(alignment_type='seqalign')
         print('Single mutations:')
         s
         print('----------------------')
         print('Mutation fingerprints')
         f
```

```
Single mutations:
```

```
Out[16]: {('N', 17, 'P'): ['N17P_SNP', 'Q4S_N17P_SNP'], ('Q', 4, 'S'): ['Q4S_N17P_SNP']}
```

```
----------------------
Mutation fingerprints
```

```
Out[16]: {(('N', 17, 'P'),): ['N17P_SNP'],
          (('Q', 4, 'S'), ('N', 17, 'P')): ['Q4S_N17P_SNP']}
```

Some additional methods

Getting binding site/other information from UniProt

```
In [17]: import ssbio.databases.uniprot
```

```
In [18]: this_examples_uniprot = 'P14062'
         sites = ssbio.databases.uniprot.uniprot_sites(this_examples_uniprot)
         my_protein.representative_sequence.features = sites
         my_protein.representative_sequence.features
```

```
Out[18]: [SeqFeature(FeatureLocation(ExactPosition(9), ExactPosition(15)), type='Nucleotide binding'),
          SeqFeature(FeatureLocation(ExactPosition(32), ExactPosition(35)), type='Nucleotide binding'),
          SeqFeature(FeatureLocation(ExactPosition(73), ExactPosition(76)), type='Nucleotide binding'),
          SeqFeature(FeatureLocation(ExactPosition(127), ExactPosition(130)), type='Region'),
          SeqFeature(FeatureLocation(ExactPosition(185), ExactPosition(187)), type='Region'),
          SeqFeature(FeatureLocation(ExactPosition(182), ExactPosition(183)), type='Active site'),
          SeqFeature(FeatureLocation(ExactPosition(189), ExactPosition(190)), type='Active site'),
          SeqFeature(FeatureLocation(ExactPosition(101), ExactPosition(102)), type='Binding site'),
          SeqFeature(FeatureLocation(ExactPosition(101), ExactPosition(102)), type='Binding site'),
          SeqFeature(FeatureLocation(ExactPosition(190), ExactPosition(191)), type='Binding site'),
          SeqFeature(FeatureLocation(ExactPosition(259), ExactPosition(260)), type='Binding site'),
          SeqFeature(FeatureLocation(ExactPosition(286), ExactPosition(287)), type='Binding site'),
          SeqFeature(FeatureLocation(ExactPosition(444), ExactPosition(445)), type='Binding site'),
          SeqFeature(FeatureLocation(ExactPosition(450), ExactPosition(451)), type='Binding site'),
          SeqFeature(FeatureLocation(ExactPosition(0), ExactPosition(468)), type='Chain', id='PRO_000
```

Mapping sequence residue numbers to structure residue numbers

Methods

Protein.**map_seqprop_resnums_to_structprop_resnums**(*resnums*, *seqprop=None*, *structprop=None*, *chain_id=None*, *use_representatives=False*)

Map a residue number in any SeqProp to the structure's residue number for a specified chain.

Parameters

- **resnums** (*int, list*) – Residue numbers in the sequence
- **seqprop** (SeqProp) – SeqProp object
- **structprop** (StructProp) – StructProp object
- **chain_id** (*str*) – Chain ID to map to
- **use_representatives** (*bool*) – If the representative sequence and structure should be used. If True, seqprop, structprop, and chain_id do not need to be defined.

Returns Mapping of sequence residue numbers to structure residue numbers

Return type dict

```
In [19]: # Returns a dictionary mapping sequence residue numbers to structure residue identifiers
         # Will warn you if residues are not present in the structure
         structure_sites = my_protein.map_seqprop_resnums_to_structprop_resnums(resnums=[1,3,45],
                                                                                use_representatives=T
         structure_sites
```

```
[2017-11-21 19:21] [ssbio.core.protein] WARNING: REP-2zyd-A, 1: structure file does not contain coord
```

```
Out[19]: {3: 3, 45: 45}
```

Viewing structures

The awesome package nglview is utilized as a backend for viewing structures within a Jupyter notebook. `ssbio` view functions will either return a `NGLWidget` object, which is the same as using `nglview` like the below example, or act upon the widget object itself.

```
# This is how NGLview usually works - it will load a structure file and return a␣
↪NGLWidget "view" object.
import nglview
view = nglview.show_structure_file(my_protein.representative_structure.structure_path)
view
```

Methods

`StructProp.`**`view_structure`**(*only_chains=None*, *opacity=1.0*, *recolor=False*, *gui=False*)

Use NGLviewer to display a structure in a Jupyter notebook

Parameters

- **only_chains** (*str, list*) – Chain ID or IDs to display
- **opacity** (*float*) – Opacity of the structure
- **recolor** (*bool*) – If structure should be cleaned and recolored to silver
- **gui** (*bool*) – If the NGLview GUI should show up

Returns NGLviewer object

```
In [20]: # View just the structure
         view = my_protein.representative_structure.view_structure()
         view
```

```
NGLWidget()
```

`Protein.`**`add_mutations_to_nglview`**(*view*, *alignment_type='seqalign'*, *alignment_ids=None*, *seqprop=None*, *structprop=None*, *chain_id=None*, *use_representatives=False*, *grouped=False*, *color='red'*, *unique_colors=True*, *opacity_range=(0.8, 1)*, *scale_range=(1, 5)*)

Add representations to an NGLWidget view object for residues that are mutated in the `sequence_alignments` attribute.

Parameters

- **view** (*NGLWidget*) – NGLWidget view object
- **alignment_type** (*str*) – Specified alignment type contained in the `annotation` field of an alignment object, `seqalign` or `structalign` are the current types.
- **alignment_ids** (*str, list*) – Specified alignment ID or IDs to use
- **seqprop** (`SeqProp`) – SeqProp object
- **structprop** (`StructProp`) – StructProp object
- **chain_id** (*str*) – ID of the structure's chain to get annotation from
- **use_representatives** (*bool*) – If the representative sequence/structure/chain IDs should be used
- **grouped** (*bool*) – If groups of mutations should be colored and sized together
- **color** (*str*) – Color of the mutations (overridden if unique_colors=True)
- **unique_colors** (*bool*) – If each mutation/mutation group should be colored uniquely
- **opacity_range** (*tuple*) – Min/max opacity values (mutations that show up more will be opaque)

- **scale_range** (*tuple*) – Min/max size values (mutations that show up more will be bigger)

```
In [21]: # Map the mutations on the visualization (scale increased) - will show up on the above view
         my_protein.add_mutations_to_nglview(view=view, alignment_type='seqalign', scale_range=(4,7),
                                             use_representatives=True)
```

```
[2017-11-21 19:21] [ssbio.protein.structure.structprop] INFO: Selection: ( :A ) and not hydrogen and
[2017-11-21 19:21] [ssbio.protein.structure.structprop] INFO: Selection: ( :A ) and not hydrogen and
```

Protein.**add_features_to_nglview**(*view*, *seqprop=None*, *structprop=None*, *chain_id=None*, *use_representatives=False*)

Add select features from the selected SeqProp object to an NGLWidget view object.

Currently parsing for:

- Single residue features (ie. metal binding sites)
- Disulfide bonds

Parameters

- **view** (*NGLWidget*) – NGLWidget view object
- **seqprop** (SeqProp) – SeqProp object
- **structprop** (StructProp) – StructProp object
- **chain_id** (*str*) – ID of the structure's chain to get annotation from
- **use_representatives** (*bool*) – If the representative sequence/structure/chain IDs should be used

```
In [22]: # Add sites as shown above in the table to the view
         my_protein.add_features_to_nglview(view=view, use_representatives=True)
```

```
[2017-11-21 19:21] [ssbio.core.protein] INFO: Active site at sequence residue 183, structure residue
[2017-11-21 19:21] [ssbio.core.protein] INFO: Active site at sequence residue 190, structure residue
[2017-11-21 19:21] [ssbio.core.protein] INFO: Binding site at sequence residue 102, structure residue
[2017-11-21 19:21] [ssbio.core.protein] INFO: Binding site at sequence residue 102, structure residue
[2017-11-21 19:21] [ssbio.core.protein] INFO: Binding site at sequence residue 191, structure residue
[2017-11-21 19:21] [ssbio.core.protein] INFO: Binding site at sequence residue 260, structure residue
[2017-11-21 19:21] [ssbio.core.protein] INFO: Binding site at sequence residue 287, structure residue
[2017-11-21 19:21] [ssbio.core.protein] INFO: Binding site at sequence residue 445, structure residue
[2017-11-21 19:21] [ssbio.core.protein] INFO: Binding site at sequence residue 451, structure residue
```

Saving

Protein.**save_json**(*outfile*, *compression=False*)

Save the object as a JSON file using json_tricks

```
In [23]: import os.path as op
         my_protein.save_json(op.join(my_protein.protein_dir, '{}.json'.format(my_protein.id)))
```

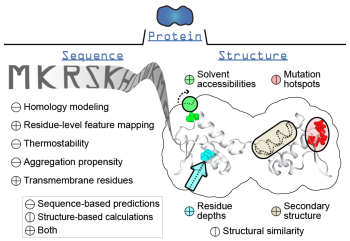
```
[2017-11-21 19:21] [root] WARNING: json-tricks: numpy scalar serialization is experimental and may wo
[2017-11-21 19:21] [ssbio.core.io] INFO: Saved <class 'ssbio.core.protein.Protein'> (id: SRR1753782_
```

5.3.3 Features

- Load, parse, and store the same (ie. from different database sources) or similar (ie. from different strains) protein sequences as SeqProp objects in the sequences attribute

- Load, parse, and store multiple experimental or predicted protein structures as `StructProp` objects in the `structures` attribute
- Set a single representative sequence and structure
- Calculate, store, and access pairwise sequence alignments to the representative sequence or structure
- Provide summaries of alignments and mutations seen
- Map between residue numbers of sequences and structures

5.3.4 Object attributes



5.3.5 Further reading

For examples in which tools from the Protein class have been used for analysis, please see the following:

5.4 The StructProp Class

5.4.1 Introduction

This section will give an overview of the methods that can be executed for a single protein structure.

5.4.2 Tutorials

PDBProp - Working With a Single PDB Structure

This notebook gives a tutorial of the **PDBProp object**, specifically how chains are handled and how to map a sequence to it.

Input: PDB ID

Output: PDBProp object

Imports

```
In [1]: from ssbio.databases.pdb import PDBProp
        from ssbio.databases.uniprot import UniProtProp
```

```
In [2]: import sys
        import logging

In [3]: # Create logger
        logger = logging.getLogger()
        logger.setLevel(logging.DEBUG)  # SET YOUR LOGGING LEVEL HERE #

In [4]: # Other logger stuff for Jupyter notebooks
        handler = logging.StreamHandler(sys.stderr)
        formatter = logging.Formatter('[%(asctime)s] [%(name)s] %(levelname)s: %(message)s', datefmt=
        handler.setFormatter(formatter)
        logger.handlers = [handler]
```

Basic methods

```
In [5]: my_structure = PDBProp(ident='5T4Q', description='E. coli ATP synthase')
```

Download the structure

Downloading will: - Download the file type of choice to the specific output directory - Parse the PDB header file to fill out the metadata fields

```
In [6]: import tempfile
        my_structure.download_structure_file(outdir=tempfile.gettempdir(), file_type='mmtf')
```

```
[2017-11-21 19:21] [ssbio.utils] DEBUG: /tmp/5t4q.mmtf: file already unzipped
[2017-11-21 19:21] [ssbio.databases.pdb] DEBUG: /tmp/5t4q.mmtf: saved structure file
[2017-11-21 19:21] [ssbio.databases.pdb] DEBUG: 5T4Q: downloaded mmtf file
```

View all attributes

```
In [7]: my_structure.get_dict()
```

```
Out[7]: {'_structure_dir': '/tmp',
         'chains': [],
         'date': None,
         'description': 'E. coli ATP synthase',
         'experimental_method': None,
         'file_type': 'mmtf',
         'id': '5T4Q',
         'is_experimental': True,
         'mapped_chains': [],
         'notes': {},
         'resolution': None,
         'structure_file': '5t4q.mmtf',
         'taxonomy_name': None}
```

Set chains that we are interested in (if any)

The `mapped_chains` attribute allows us to limit sequence analyses to specified chains (see the later section where we *align a sequence to this structure*). For this example, the ATP synthase is a complex of a number of protein chains, and if we are interested in a specific gene transcript, we can set those.

```
In [8]: # Chains A, B, and C make up ATP synthase subunit alpha - from the gene b3734 (UniProt ID P0
        my_structure.add_mapped_chain_ids(['A', 'B', 'C'])
```

```
[2017-11-21 19:21] [ssbio.protein.structure.structprop] DEBUG: A: added to list of mapped chains
[2017-11-21 19:21] [ssbio.protein.structure.structprop] DEBUG: B: added to list of mapped chains
[2017-11-21 19:21] [ssbio.protein.structure.structprop] DEBUG: C: added to list of mapped chains
```

Parse the structure to work with the Biopython Structure object

Parsing the structure will parse the sequences of each chain, and store those in the `chains` attribute. It will also return a Biopython Structure object which opens up all methods available for structures in Biopython.

```
In [9]: parsed_structure = my_structure.parse_structure()
        print(type(parsed_structure.structure))
        print(type(parsed_structure.first_model))
```

```
[2017-11-21 19:21] [ssbio.protein.structure.utils.structureio] DEBUG: 5t4q.mmtf: parsed 3D coordinate
[2017-11-21 19:21] [ssbio.protein.structure.structprop] DEBUG: A: added to chains list
[2017-11-21 19:21] [ssbio.protein.structure.structprop] DEBUG: B: added to chains list
[2017-11-21 19:21] [ssbio.protein.structure.structprop] DEBUG: C: added to chains list
[2017-11-21 19:21] [ssbio.protein.structure.structprop] DEBUG: D: added to chains list
[2017-11-21 19:21] [ssbio.protein.structure.structprop] DEBUG: E: added to chains list
[2017-11-21 19:21] [ssbio.protein.structure.structprop] DEBUG: F: added to chains list
[2017-11-21 19:21] [ssbio.protein.structure.structprop] DEBUG: G: added to chains list
[2017-11-21 19:21] [ssbio.protein.structure.structprop] DEBUG: H: added to chains list
[2017-11-21 19:21] [ssbio.protein.structure.structprop] DEBUG: I: added to chains list
[2017-11-21 19:21] [ssbio.protein.structure.structprop] DEBUG: J: added to chains list
[2017-11-21 19:21] [ssbio.protein.structure.structprop] DEBUG: K: added to chains list
[2017-11-21 19:21] [ssbio.protein.structure.structprop] DEBUG: L: added to chains list
[2017-11-21 19:21] [ssbio.protein.structure.structprop] DEBUG: M: added to chains list
[2017-11-21 19:21] [ssbio.protein.structure.structprop] DEBUG: N: added to chains list
[2017-11-21 19:21] [ssbio.protein.structure.structprop] DEBUG: O: added to chains list
[2017-11-21 19:21] [ssbio.protein.structure.structprop] DEBUG: P: added to chains list
[2017-11-21 19:21] [ssbio.protein.structure.structprop] DEBUG: Q: added to chains list
[2017-11-21 19:21] [ssbio.protein.structure.structprop] DEBUG: R: added to chains list
[2017-11-21 19:21] [ssbio.protein.structure.structprop] DEBUG: S: added to chains list
[2017-11-21 19:21] [ssbio.protein.structure.structprop] DEBUG: T: added to chains list
[2017-11-21 19:21] [ssbio.protein.structure.structprop] DEBUG: U: added to chains list
[2017-11-21 19:21] [ssbio.protein.structure.structprop] DEBUG: V: added to chains list
[2017-11-21 19:21] [ssbio.protein.structure.structprop] DEBUG: 5T4Q: gathered chain sequences
[2017-11-21 19:21] [ssbio.protein.structure.structprop] DEBUG: A: adding chain sequence to ChainProp
[2017-11-21 19:21] [ssbio.protein.structure.structprop] DEBUG: B: adding chain sequence to ChainProp
[2017-11-21 19:21] [ssbio.protein.structure.structprop] DEBUG: C: adding chain sequence to ChainProp
[2017-11-21 19:21] [ssbio.protein.structure.structprop] DEBUG: D: adding chain sequence to ChainProp
[2017-11-21 19:21] [ssbio.protein.structure.structprop] DEBUG: E: adding chain sequence to ChainProp
[2017-11-21 19:21] [ssbio.protein.structure.structprop] DEBUG: F: adding chain sequence to ChainProp
[2017-11-21 19:21] [ssbio.protein.structure.structprop] DEBUG: G: adding chain sequence to ChainProp
[2017-11-21 19:21] [ssbio.protein.structure.structprop] DEBUG: H: adding chain sequence to ChainProp
[2017-11-21 19:21] [ssbio.protein.structure.structprop] DEBUG: I: adding chain sequence to ChainProp
[2017-11-21 19:21] [ssbio.protein.structure.structprop] DEBUG: J: adding chain sequence to ChainProp
[2017-11-21 19:21] [ssbio.protein.structure.structprop] DEBUG: K: adding chain sequence to ChainProp
[2017-11-21 19:21] [ssbio.protein.structure.structprop] DEBUG: L: adding chain sequence to ChainProp
[2017-11-21 19:21] [ssbio.protein.structure.structprop] DEBUG: M: adding chain sequence to ChainProp
[2017-11-21 19:21] [ssbio.protein.structure.structprop] DEBUG: N: adding chain sequence to ChainProp
[2017-11-21 19:21] [ssbio.protein.structure.structprop] DEBUG: O: adding chain sequence to ChainProp
[2017-11-21 19:21] [ssbio.protein.structure.structprop] DEBUG: P: adding chain sequence to ChainProp
[2017-11-21 19:21] [ssbio.protein.structure.structprop] DEBUG: Q: adding chain sequence to ChainProp
[2017-11-21 19:21] [ssbio.protein.structure.structprop] DEBUG: R: adding chain sequence to ChainProp
[2017-11-21 19:21] [ssbio.protein.structure.structprop] DEBUG: S: adding chain sequence to ChainProp
[2017-11-21 19:21] [ssbio.protein.structure.structprop] DEBUG: T: adding chain sequence to ChainProp
[2017-11-21 19:21] [ssbio.protein.structure.structprop] DEBUG: U: adding chain sequence to ChainProp
```

```
[2017-11-21 19:21] [ssbio.protein.structure.structprop] DEBUG: V: adding chain sequence to ChainProp
```

```
<class 'Bio.PDB.Structure.Structure'>
<class 'Bio.PDB.Model.Model'>
```

Clean the structure and save the structure

Cleaning a structure does the following: - Add missing chain identifiers to a PDB file - Select a single chain if noted - Remove alternate atom locations - Add atom occupancies - Add B (temperature) factors (default Biopython behavior)

In the example below, we will clean the structure so it only includes our mapped chains.

```
In [10]: cleaned_structure = my_structure.clean_structure(outdir='/tmp', keep_chains=my_structure.map
         cleaned_structure
```

```
[2017-11-21 19:21] [ssbio.protein.structure.utils.structureio] DEBUG: 5t4q.mmtf: parsed 3D coordinate
```

```
Out[10]: '/tmp/5t4q_clean.pdb'
```

Viewing the structure

```
In [13]: # The original structure
         my_structure.view_structure(recolor=False)
```

```
NGLWidget()
```

```
In [14]: # The cleaned structure
         import nglview
         nglview.show_structure_file(cleaned_structure)
```

```
NGLWidget()
```

FATCAT - Structure Similarity

This notebook shows how to run and parse FATCAT, a structural similarity calculator.

```
In [1]: import ssbio.protein.structure.properties.fatcat as fatcat
```

```
In [2]: import os
        import os.path as op
        import tempfile

        ROOT_DIR = tempfile.gettempdir()
        OUT_DIR = op.join(ROOT_DIR, 'fatcat_testing')
        if not op.exists(OUT_DIR):
            os.mkdir(OUT_DIR)
        FATCAT_SH = '/home/nathan/software/fatcat/runFATCAT.sh'
```

Pairwise

```
In [6]: fatcat_outfile = fatcat.run_fatcat(structure_path_1='../../ssbio/test/test_files/structures/1
                                           structure_path_2='../../ssbio/test/test_files/structures/1
                                           outdir=OUT_DIR,
                                           fatcat_sh=FATCAT_SH, print_cmd=True, force_rerun=True)
        print('Output file:', fatcat_outfile)
```

```
/home/nathan/software/fatcat/runFATCAT.sh -file1 ../../ssbio/test/test_files/structures/12as-A_clean
Protein Comparison Tool 4.1.1-SNAPSHOT 20151103-1640
file from local /mnt/projects/ssbio/docs/notebooks/../../ssbio/test/test_files/structures/12as-A_clea
file from local /mnt/projects/ssbio/docs/notebooks/../../ssbio/test/test_files/structures/1a9x-A_clea
Output file: /tmp/fatcat_testing/12as-A_clean__1a9x-A_clean.xml
```

```
In [7]: fatcat.parse_fatcat(fatcat_outfile)
```

```
Out[7]: {'tm_score': 0.27}
```

All-by-all

```
In [8]: structs = ['../../ssbio/test/test_files/structures/12as-A_clean.pdb',
                   '../../ssbio/test/test_files/structures/1af6-A_clean.pdb',
                   '../../ssbio/test/test_files/structures/1a9x-A_clean.pdb']
```

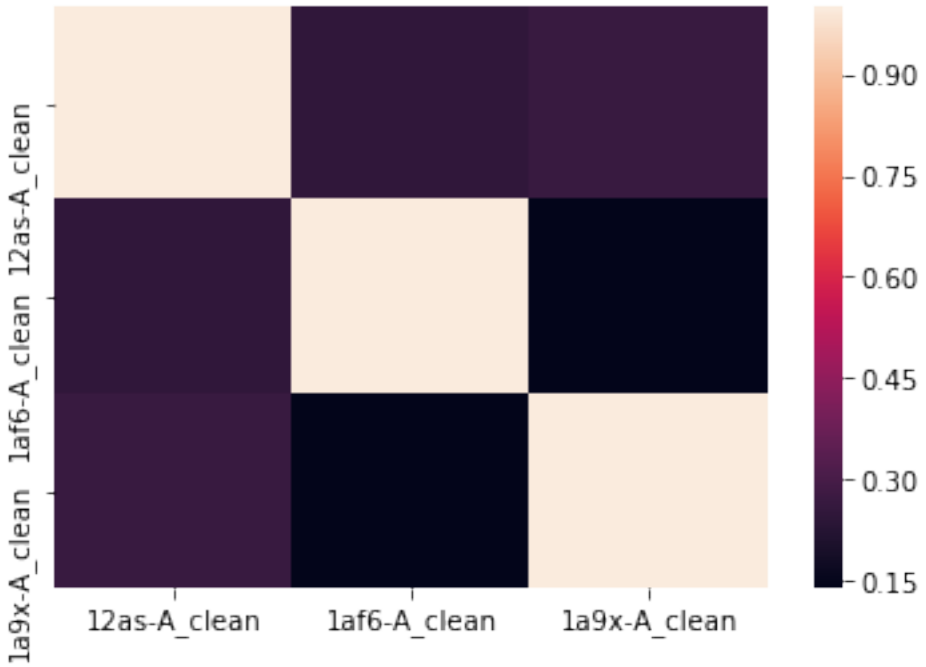
```
In [9]: tm_scores = fatcat.run_fatcat_all_by_all(structs, fatcat_sh=FATCAT_SH, outdir=OUT_DIR)
        tm_scores
```

```
3it [00:08,  2.95s/it]
```

```
Out[9]: 12as-A_clean  1af6-A_clean  1a9x-A_clean
        12as-A_clean          1.00          0.25          0.27
        1af6-A_clean          0.25          1.00          0.14
        1a9x-A_clean          0.27          0.14          1.00
```

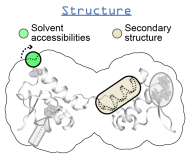
```
In [10]: %matplotlib inline
         import seaborn as sns
         sns.heatmap(tm_scores)
```

```
Out[10]: <matplotlib.axes._subplots.AxesSubplot at 0x7f4f060c7cf8>
```



5.4.3 External programs

DSSP



Description

- DSSP home page

DSSP (**D**efine **S**econdary **S**tructure of **P**roteins) is the standard method used to assign secondary structure annotations to a protein structure. DSSP utilizes the atomic coordinates of a structure to assign the structure codes, which are:

Code	Description
H	Alpha helix
B	Beta bridge
E	Strand
G	Helix-3
I	Helix-5
T	Turn
S	Bend

Furthermore, DSSP calculates geometric properties such as the phi and psi angles between residues and solvent accessibilities. *ssbio* provides wrappers around the Biopython DSSP module to execute and parse DSSP results, as well as converting the information into a Pandas DataFrame format with calculated relative solvent accessbilities (see *ssbio.protein.structure.properties.dssp* for details).

Instructions (Ubuntu)

Note: These instructions were created on an Ubuntu 17.04 system.

1. Install the DSSP package

```
sudo apt-get install dssp
```

2. The program installs itself as `mkdssp`, not `dssp`, and Biopython looks to execute `dssp`, so we need to symlink the name `dssp` to `mkdssp`

```
sudo ln -s /usr/bin/mkdssp /usr/bin/dssp
```

3. Then you should be able to run `dssp` in your terminal

Instructions (Mac OSX)

- Instructions for installing on Mac

- Instructions for installing on Mac (alternate)

FAQs

- How do I cite DSSP?
  - Kabsch W & Sander C (1983) DSSP: definition of secondary structure of proteins given a set of 3D coordinates. Biopolymers 22: 2577–2637
- I'm having issues running DSSP...
  - See the ssbio wiki for (hopefully) some solutions - or add yours in when you find the answer!

API

ssbio.protein.structure.properties.dssp.**all_dssp_props**(*filename*, *file_type*)
Returns a large dictionary of SASA, secondary structure composition, and surface/buried composition. Values are computed using DSSP. Input: PDB or MMCIF filename Output: Dictionary of values obtained from dssp

ssbio.protein.structure.properties.dssp.**calc_sasa**(*dssp_df*)
Calculation of SASA utilizing the DSSP program.

DSSP must be installed for biopython to properly call it. Install using apt-get on Ubuntu or from: http://swift.cmbi.ru.nl/gv/dssp/

Input: PDB or CIF structure file Output: SASA (integer) of structure

ssbio.protein.structure.properties.dssp.**calc_surface_buried**(*dssp_df*)
Calculates the percent of residues that are in the surface or buried, as well as if they are polar or nonpolar. Returns a dictionary of this.

ssbio.protein.structure.properties.dssp.**get_dssp_df**(*model*, *pdb_file*, *outfile=None*, *outdir=None*, *outext='_dssp.df'*, *force_rerun=False*)

Parameters

- **model** –
- **pdb_file** –
- **outfile** –
- **outdir** –
- **outext** –
- **force_rerun** –

Returns:

ssbio.protein.structure.properties.dssp.**get_dssp_df_on_file**(*pdb_file*, *outfile=None*, *outdir=None*, *outext='_dssp.df'*, *force_rerun=False*)
Run DSSP directly on a structure file with the Biopython method Bio.PDB.DSSP.dssp_dict_from_pdb_file

Avoids errors like: PDBException: Structure/DSSP mismatch at <Residue MSE het= resseq=19 icode= >
by not matching information to the structure file (DSSP fills in the ID "X" for unknown residues)

Parameters

- **pdb_file** – Path to PDB file
- **outfile** – Name of output file
- **outdir** – Path to output directory
- **outext** – Extension of output file
- **force_rerun** – If DSSP should be rerun if the outfile exists

Returns DSSP results, summarized

Return type Pandas DataFrame

ssbio.protein.structure.properties.dssp.**get_ss_class**(*pdb_file*, *dssp_file*, *chain*)
Define the secondary structure class of a PDB file at the specific chain

Parameters

- **pdb_file** –
- **dssp_file** –
- **chain** –

Returns:

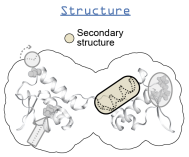
ssbio.protein.structure.properties.dssp.**secondary_structure_summary**(*dssp_df*)
Summarize the secondary structure content of the DSSP dataframe for each chain.

Parameters **dssp_df** – Pandas DataFrame of parsed DSSP results

Returns Chain to secondary structure summary dictionary

Return type dict

STRIDE



Description

- STRIDE home page
- STRIDE download page
- STRIDE documentation

STRIDE (**St**ructural **ide**ntification) is a program used to assign secondary structure annotations to a protein structure. STRIDE has slightly more complex criteria to assign codes compared to **dssp_**. STRIDE utilizes the atomic coordinates of a structure to assign the structure codes, which are:

Code	Description
H	Alpha helix
G	3-10 helix
I	PI-helix
E	Extended conformation
B or b	Isolated bridge
T	Turn
C	Coil (none of the above)

Instructions (Unix)

Note: These instructions were created on an Ubuntu 17.04 system.

1. Download the source from the STRIDE download page
2. Create a new folder named "stride" in a place where you store software and extract the source into it

```
mkdir /path/to/software/stride
cp /path/to/downloaded/stride.tar.gz /path/to/software/stride
cd /path/to/software/stride
tar -zxf stride.tar.gz
```

3. Build the program from source and copy its binary:

```
cd /path/to/software/stride
make
cp stride /usr/local/bin
```

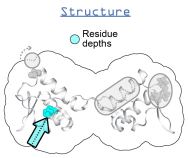
4. Then you should be able to run `stride` in your terminal

FAQs

- How do I cite STRIDE?
  - Frishman D & Argos P (1995) Knowledge-based protein secondary structure assignment. Proteins 23: 566–579 Available at: http://dx.doi.org/10.1002/prot.340230412
- I'm having issues running STRIDE...
  - See the ssbio wiki for (hopefully) some solutions - or add yours in when you find the answer!

API

MSMS



Description

- MSMS home page
- Download
- Manual
- Manuscript

MSMS computes solvent excluded surfaces on a protein structure. Generally, MSMS is used to calculate residue depths (in Angstroms) from the surface of a protein, using a PDB file as an input. *ssbio* provides wrappers through Biopython to run MSMS as well as store the depths in an associated `StructProp` object.

Instructions (Unix)

Note: These instructions were created on an Ubuntu 17.04 system.

1. Head to the Download page, and under the header "MSMS 2.6.X - Current Release" download the "Unix/Linux i86_64" version - if this doesn't work though you'll want to try the "Unix/Linux i86" version later.
2. Download it, unarchive it to your library path:

```
sudo mkdir /usr/local/lib/msms
cd /usr/local/lib/msms
sudo tar zxvf /path/to/your/downloaded/file/msms_i86Linux2_2.6.1.tar.gz
```

3. Symlink the binaries (or alternatively, add the two locations to your PATH):

```
sudo ln -s /usr/local/lib/msms/msms.i86Linux2.2.6.1 /usr/local/bin/msms
sudo ln -s /usr/local/lib/msms/pdb_to_xyzr* /usr/local/bin
```

4. Fix a bug in the pdb_to_xyzr file (see: http://mailman.open-bio.org/pipermail/biopython/2015-November/015787.html):

```
$ sudo gedit /usr/local/lib/msms/pdb_to_xyzr
```

   at line 34, change:

```
$ numfile = "./atmtypenumbers"
```

   to:

```
> numfile = "/usr/local/lib/msms/atmtypenumbers"
```

5. Repeat step 5 for the file `/usr/local/lib/msms/pdb_to_xyzrn`
6. Now try running `msms` in the terminal, it should say:

```
MSMS 2.6.1 started on structure
Copyright M.F. Sanner (1994)
Compilation flags -O2 -DVERBOSE -DTIMING
MSMS: No input stream specified
```

FAQs

- How do I cite MSMS?
  - Sanner MF, Olson AJ & Spehner J-C (1996) Reduced surface: an efficient way to compute molecular surfaces. Biopolymers 38: 305–320. Available at: http://mgl.scripps.edu/people/sanner/html/papers/msmsTextAndFigs.pdf
- How long does it take to run?
  - Depending on the size of the protein structure, the program can take up to a couple minutes to execute.
- I'm having issues running MSMS...
  - See the ssbio wiki for (hopefully) some solutions - or add yours in when you find the answer!

API

ssbio.protein.structure.properties.msms.**get_msms_df**(*model*, *pdb_file*, *outfile=None*, *outdir=None*, *outext='_msms.df'*, *force_rerun=False*)

Run MSMS (using Biopython) on a Biopython Structure Model and the path to the actual PDB file.

Returns a dictionary of:

{chain_id: {resnum1_id: (res_depth, ca_depth)}, {resnum2_id: (res_depth, ca_depth)} }

Depths are in units Angstroms. 1A = 10^-10 m = 1nm

Parameters

- **model** – Biopython Structure Model
- **pdb_file** – Path to PDB file

Returns ResidueDepth property_dict, reformatted

Return type Pandas DataFrame

ssbio.protein.structure.properties.msms.**get_msms_df_on_file**(*pdb_file*, *outfile=None*, *outdir=None*, *outext='_msms.df'*, *force_rerun=False*)

Run MSMS (using Biopython) on a PDB file.

Saves a CSV file of: chain: chain ID resnum: residue number (PDB numbering) icode: residue insertion code res_depth: average depth of all atoms in a residue ca_depth: depth of the alpha carbon atom

Depths are in units Angstroms. 1A = 10^-10 m = 1nm

Parameters

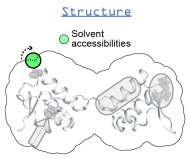
- **pdb_file** – Path to PDB file
- **outfile** – Optional name of output file (without extension)
- **outdir** – Optional output directory
- **outext** – Optional extension for the output file
- **outext** – Suffix appended to json results file

- **force_rerun** – Rerun MSMS even if results exist already

Returns ResidueDepth property_dict, reformatted

Return type Pandas DataFrame

FreeSASA



Description

- FreeSASA home page
- FreeSASA Github

FreeSASA is an open source library written in C for calculating solvent accessible surface areas of a protein. FreeSASA also contains Python bidings, and the plan is to include these bindings with *ssbio* in the future.

Instructions (Unix)

Note: These instructions were created on an Ubuntu 17.04 system with a Python installation through Anaconda3.

Note: FreeSASA Python bindings are slightly difficult to install with Python 3 - ssbio provides wrappers for the command line executable instead

1. Download the latest tarball (see FreeSASA home page), expand it and run

```
./configure --enable-python-bindings CFLAGS="-fPIC -O2"
make
```

2. If you have a user-specific Python executable (ie. through Anaconda), edit the freesasa-2.0/bindings/Makefile, lines 805, 809, 815 to change:

```
python setup.py [...]
```

   to (type *which python* to get the path to enter):

```
/path/to/your/anaconda/python setup.py [...]
```

3. Install with

```
sudo make install
```

FAQs

- How do I cite FreeSASA?
  - Mitternacht S (2016) FreeSASA: An open source C library for solvent accessible surface area calculations. F1000Res. 5: 189 Available at: http://dx.doi.org/10.12688/f1000research.7931.1
- I'm having issues running FreeSASA...
  - See the ssbio wiki for (hopefully) some solutions - or add yours in when you find the answer!

API

ssbio.protein.structure.properties.freesasa.**parse_rsa_data**(*rsa_outfile*, *ignore_hets=True*)

Process a NACCESS or freesasa RSA output file. Adapted from Biopython NACCESS modele.

Parameters

- **rsa_outfile** (*str*) – Path to RSA output file
- **ignore_hets** (*bool*) – If HETATMs should be excluded from the final dictionary. This is extremely important when loading this information into a ChainProp's SeqRecord, since this will throw off the sequence matching.

Returns Per-residue dictionary of RSA values

Return type dict

ssbio.protein.structure.properties.freesasa.**run_freesasa**(*infile*, *outfile*, *include_hetatms=True*, *outdir=None*, *force_rerun=False*)

Run freesasa on a PDB file, output using the NACCESS RSA format.

Parameters

- **infile** (*str*) – Path to PDB file (only PDB file format is accepted)
- **outfile** (*str*) – Path or filename of output file
- **include_hetatms** (*bool*) – If heteroatoms should be included in the SASA calculations
- **outdir** (*str*) – Path to output file if not specified in outfile
- **force_rerun** (*bool*) – If freesasa should be rerun even if outfile exists

Returns Path to output SASA file

Return type str

FATCAT

Description

- FATCAT home page
- jFATCAT Java version
- jFATCAT download page

FATCAT is a structural alignment tool that allows you to determine the similarity of a pair of protein structures.

Warning: Parsing FATCAT results is currently incomplete and will only return TM-scores as of now - but TM-scores only show up in development versions of jFATCAT

Instructions

Note: These instructions were created on an Ubuntu 17.04 system.

1. Download the Java port of FATCAT from the jFATCAT download page, under the section "Older file downloads" with the filename "protein-comparison-tool_<DATE>.tar.gz"
2. Extract it to a place where you store software
3. Run `ssbio.protein.structure.properties.fatcat.run_fatcat()` on two structures, pointing to the path of the *runFATCAT.sh* script

FAQs

- How do I cite FATCAT?
  - Ye Y & Godzik A (2003) Flexible structure alignment by chaining aligned fragment pairs allowing twists. Bioinformatics 19 Suppl 2: ii246–55 Available at: https://www.ncbi.nlm.nih.gov/pubmed/14534198
- I'm having issues running FATCAT...
  - See the ssbio wiki for (hopefully) some solutions - or add yours in when you find the answer!

API

ssbio.protein.structure.properties.fatcat.**parse_fatcat**(*fatcat_xml*)

Parse a FATCAT XML result file.

Parameters **fatcat_xml** (*str*) – Path to FATCAT XML result file

Returns Parsed information from the output

Return type dict

Todo:

- Only returning TM-score at the moment

ssbio.protein.structure.properties.fatcat.**run_fatcat**(*structure_path_1*, *structure_path_2*, *fatcat_sh*, *outdir=''*, *silent=False*, *print_cmd=False*, *force_rerun=False*)

Run FATCAT on two PDB files, and return the path of the XML result file.

Parameters

- **structure_path_1** (*str*) – Path to PDB file

- **structure_path_2** (*str*) – Path to PDB file
- **fatcat_sh** (*str*) – Path to "runFATCAT.sh" executable script
- **outdir** (*str*) – Path to where FATCAT XML output files will be saved
- **silent** (*bool*) – If stdout should be silenced from showing up in Python console output
- **print_cmd** (*bool*) – If command to run FATCAT should be printed to stdout
- **force_rerun** (*bool*) – If FATCAT should be run even if XML output files already exist

Returns Path to XML output file

Return type str

ssbio.protein.structure.properties.fatcat.**run_fatcat_all_by_all**(*list_of_structure_paths*, *fatcat_sh*, *outdir=''*, *silent=True*, *force_rerun=False*)

Run FATCAT on all pairs of structures given a list of structures.

Parameters

- **list_of_structure_paths** (*list*) – List of PDB file paths
- **fatcat_sh** (*str*) – Path to "runFATCAT.sh" executable script
- **outdir** (*str*) – Path to where FATCAT XML output files will be saved
- **silent** (*bool*) – If command to run FATCAT should be printed to stdout
- **force_rerun** (*bool*) – If FATCAT should be run even if XML output files already exist

Returns TM-scores (similarity) between all structures

Return type Pandas DataFrame

OPM

Description

- OPM home page
- OPM web server
- OPM web server instructions and description of results

OPM is a program to predict the location of transmembrane planes in protein structures, utilizing the atomic coordinates. *ssbio* provides a wrapper to submit PDB files to the web server, cache, and parse the results

Instructions

1. Use the function *ssbio.protein.structure.properties.opm.run_ppm_server()* to upload a PDB file to the PPM server.

FAQs

- How can I install OPM?
  - OPM is only available as a web server. *ssbio* provides a wrapper for the web server and allows you to submit protein structures to it along with caching the output files.
- How do I cite OPM?
  - Lomize MA, Pogozheva ID, Joo H, Mosberg HI & Lomize AL (2012) OPM database and PPM web server: resources for positioning of proteins in membranes. Nucleic Acids Res. 40: D370–6 Available at: http://dx.doi.org/10.1093/nar/gkr703
- I'm having issues running OPM...
  - See the ssbio wiki for (hopefully) some solutions - or add yours in when you find the answer!

API

ssbio.protein.structure.properties.opm.**run_ppm_server**(*pdb_file*, *outfile*, *force_rerun=False*)

Run the PPM server from OPM to predict transmembrane residues.

Parameters

- **pdb_file** (*str*) – Path to PDB file
- **outfile** (*str*) – Path to output HTML results file
- **force_rerun** (*bool*) – Flag to rerun PPM if HTML results file already exists

Returns Dictionary of information from the PPM run, including a link to download the membrane protein file

Return type dict

5.5 The SeqProp Class

5.5.1 Introduction

This section will give an overview of the methods that can be executed for a single protein sequence.

5.5.2 Tutorials

SeqProp - Protein Sequence Properties

This notebook gives an overview the available **calculations for properties of a single protein sequence**.

Input: Amino acid sequence

Output: Amino acid sequence properties

Note: See *ssbio.protein.sequence.seqprop.SeqProp* for a description of all the available attributes and functions.

Imports

```
In [1]: import sys
        import logging
        import os.path as op
```

```
In [2]: # Import the SeqProp class
        from ssbio.protein.sequence.seqprop import SeqProp
```

```
In [3]: # Printing multiple outputs per cell
        from IPython.core.interactiveshell import InteractiveShell
        InteractiveShell.ast_node_interactivity = "all"
```

Logging

Set the logging level in `logger.setLevel(logging.<LEVEL_HERE>)` to specify how verbose you want the pipeline to be. Debug is most verbose.

- `CRITICAL`
  - Only really important messages shown
- `ERROR`
  - Major errors
- `WARNING`
  - Warnings that don't affect running of the pipeline
- `INFO` (default)
  - Info such as the number of structures mapped per gene
- `DEBUG`
  - Really detailed information that will print out a lot of stuff

```
In [4]: # Create logger
        logger = logging.getLogger()
        logger.setLevel(logging.INFO)  # SET YOUR LOGGING LEVEL HERE #
```

```
In [5]: # Other logger stuff for Jupyter notebooks
        handler = logging.StreamHandler(sys.stderr)
        formatter = logging.Formatter('[%(asctime)s] [%(name)s] %(levelname)s: %(message)s', datefmt=
        handler.setFormatter(formatter)
        logger.handlers = [handler]
```

Initialization of the project

Set these two things:

- `PROTEIN_ID`
  - Your protein ID

- PROTEIN_SEQ
  - Your protein sequence

```
In [6]: # SET IDS HERE
        PROTEIN_ID = 'YIAJ_ECOLI'
        PROTEIN_SEQ = 'MGKEVMGKKENEMAQEKERPAGSQSLFRGLMLIEILSNYPNGCPLAHLSELAGLNKSTVHRLLQGLQSCGYVTTAPA
```

```
In [7]: # Create the SeqProp object
        my_seq = SeqProp(id=PROTEIN_ID, seq=PROTEIN_SEQ)
```

SeqProp.**write_fasta_file**(*outfile*, *force_rerun=False*)
Write a FASTA file for the protein sequence, seq will now load directly from this file.

Parameters

- **outfile** (*str*) – Path to new FASTA file to be written to
- **force_rerun** (*bool*) – If an existing file should be overwritten

```
In [8]: # Write temporary FASTA file for property calculations that require FASTA file as input
        import tempfile
        ROOT_DIR = tempfile.gettempdir()

        my_seq.write_fasta_file(outfile=op.join(ROOT_DIR, 'tmp.fasta'), force_rerun=True)
        my_seq.sequence_path
```

```
Out[8]: '/tmp/tmp.fasta'
```

Computing and storing protein properties

A SeqProp object is simply an extension of the Biopython SeqRecord object. Global properties which describe or summarize the entire protein sequence are stored in the annotations attribute, while local residue-specific properties are stored in the letter_annotations attribute.

Basic global properties

SeqProp.**get_biopython_pepstats**()
Run Biopython's built in ProteinAnalysis module and store statistics in the annotations attribute.

```
In [9]: # Global properties using the Biopython ProteinAnalysis module
        my_seq.get_biopython_pepstats()
        {k:v for k,v in my_seq.annotations.items() if k.endswith('-biop')}
```

```
Out[9]: {'amino_acids_percent-biop': {'A': 0.09219858156028368,
          'C': 0.014184397163120567,
          'D': 0.028368794326241134,
          'E': 0.07801418439716312,
          'F': 0.024822695035460994,
          'G': 0.07446808510638298,
          'H': 0.03900709219858156,
          'I': 0.07092198581560284,
          'K': 0.04609929078014184,
          'L': 0.1099290780141844,
          'M': 0.03546099290780142,
          'N': 0.03900709219858156,
          'P': 0.04609929078014184,
          'Q': 0.03546099290780142,
          'R': 0.04964539007092199,
          'S': 0.07446808510638298,
```

```
 'T': 0.06028368794326241,
 'V': 0.0425531914893617,
 'W': 0.0035460992907801418,
 'Y': 0.03546099290780142},
'aromaticity-biop': 0.06382978723404256,
'instability_index-biop': 46.34609929078015,
'isoelectric_point-biop': 6.41558837890625,
'molecular_weight-biop': 31066.304700000015,
'monoisotopic-biop': False,
'percent_helix_naive-biop': 0.2872340425531915,
'percent_strand_naive-biop': 0.31560283687943264,
'percent_turn_naive-biop': 0.23404255319148937}
```

SeqProp.**get_emboss_pepstats**()

Run the EMBOSS pepstats program on the protein sequence.

Stores statistics in the `annotations` attribute. Saves a `.pepstats` file of the results where the sequence file is located.

```
In [10]: # Global properties from the EMBOSS pepstats program
         my_seq.get_emboss_pepstats()
         {k:v for k,v in my_seq.annotations.items() if k.endswith('-pepstats')}
```

```
Out[10]: {'percent_acidic-pepstats': 0.10638,
          'percent_aliphatic-pepstats': 0.3156,
          'percent_aromatic-pepstats': 0.10284,
          'percent_basic-pepstats': 0.13475,
          'percent_charged-pepstats': 0.24112999999999998,
          'percent_non-polar-pepstats': 0.5496500000000001,
          'percent_polar-pepstats': 0.45035,
          'percent_small-pepstats': 0.47163,
          'percent_tiny-pepstats': 0.3156}
```

SeqProp.**get_aggregation_propensity**(*email*, *password*, *cutoff_v=5*, *cutoff_n=5*, *run_amylmuts=False*, *outdir=None*)

Run the AMYLPRED2 web server to calculate the aggregation propensity of this protein sequence, which is the number of aggregation-prone segments on the unfolded protein sequence.

Stores statistics in the `annotations` attribute, under the key *aggprop-amylpred*.

See *ssbio.protein.sequence.properties.aggregation_propensity* for instructions and details.

```
In [11]: # Aggregation propensity - the predicted number of aggregation-prone segments on an unfolde
         my_seq.get_aggregation_propensity(outdir=ROOT_DIR, email='nmih@ucsd.edu', password='ssbiotes
         {k:v for k,v in my_seq.annotations.items() if k.endswith('-amylpred')}
```

```
Out[11]: {'aggprop-amylpred': 7}
```

SeqProp.**get_kinetic_folding_rate**(*secstruct*, *at_temp=None*)

Run the FOLD-RATE web server to calculate the kinetic folding rate given an amino acid sequence and its structural classficiation (alpha/beta/mixed)

Stores statistics in the `annotations` attribute, under the key *kinetic_folding_rate_<TEMP>-foldrate*.

See `ssbio.protein.sequence.properties.kinetic_folding_rate.get_foldrate()` for instructions and details.

```
In [12]: # Kinetic folding rate - the predicted rate of folding for this protein sequence
         secstruct_class = 'mixed'
         my_seq.get_kinetic_folding_rate(secstruct=secstruct_class)
         {k:v for k,v in my_seq.annotations.items() if k.endswith('-foldrate')}
```

```
Out[12]: {'kinetic_folding_rate_37.0_C-foldrate': '3.1'}
```

SeqProp.**get_thermostability**(*at_temp*)
Run the thermostability calculator using either the Dill or Oobatake methods.

Stores calculated (dG, Keq) tuple in the `annotations` attribute, under the key *thermostability_<TEMP>-<METHOD_USED>*.

See `ssbio.protein.sequence.properties.thermostability.get_dG_at_T()` for instructions and details.

```
In [13]: # Thermostability - prediction of free energy of unfolding dG from protein sequence
         # Stores (dG, Keq)
         my_seq.get_thermostability(at_temp=32.0)
         my_seq.get_thermostability(at_temp=37.0)
         my_seq.get_thermostability(at_temp=42.0)
         {k:v for k,v in my_seq.annotations.items() if k.startswith('thermostability_')}

Out[13]: {'thermostability_32.0_C-oobatake': (-485.4540664728014, 2.22678150661948),
          'thermostability_37.0_C-oobatake': (-2126.8775952298206, 31.527746910631482),
          'thermostability_42.0_C-oobatake': (-4205.694728563369, 825.0926295027567)}
```

5.5.3 External programs

EMBOSS

Description

- EMBOSS home page
- EMBOSS source code

EMBOSS is the European Molecular Biology Open Software Suite. EMBOSS contains a wide array of general purpose bioinformatics programs. For the GEM-PRO pipeline, we mainly need the *needle* pairwise alignment tool (although this can be replaced with Biopython's built-in pairwise alignment function), and the *pepstats* protein sequence statistics tool.

Instructions (Ubuntu)

Note: These instructions were created on an Ubuntu 17.04 system.

1. Install the EMBOSS package which contains many programs

   ```
   sudo apt-get install emboss
   ```

2. And then once that installs, try running the `needle` program:

   ```
   needle
   ```

Instructions (Mac OSX, other Unix)

1. Just install after downloading the EMBOSS source code

```
./configure
make
sudo make install
```

FAQs

- How do I cite EMBOSS?
  - Rice P, Longden I & Bleasby A (2000) EMBOSS: the European Molecular Biology Open Software Suite. Trends Genet. 16: 276–277 Available at: http://www.ncbi.nlm.nih.gov/pubmed/10827456
- I'm having issues running EMBOSS programs...
  - See the ssbio wiki for (hopefully) some solutions - or add yours in when you find the answer!

API

ssbio.protein.sequence.properties.residues.**biopython_protein_analysis**(*inseq*)

Utiize Biopython's ProteinAnalysis module to return general sequence properties of an amino acid string.

For full definitions see: http://biopython.org/DIST/docs/api/Bio.SeqUtils.ProtParam.ProteinAnalysis-class.html

Parameters **inseq** – Amino acid sequence

Returns Dictionary of sequence properties. Some definitions include: instability_index: Any value above 40 means the protein is unstable (has a short half life). secondary_structure_fraction: Percentage of protein in helix, turn or sheet

Return type dict

Todo: Finish definitions of dictionary

---

ssbio.protein.sequence.properties.residues.**emboss_pepstats_on_fasta**(*infile*, *outfile=''*, *outdir=''*, *outext='.pepstats'*, *force_rerun=False*)

Run EMBOSS pepstats on a FASTA file.

Parameters

- **infile** – Path to FASTA file
- **outfile** – Name of output file without extension
- **outdir** – Path to output directory
- **outext** – Extension of results file, default is ".pepstats"
- **force_rerun** – Flag to rerun pepstats

Returns Path to output file.

Return type str

ssbio.protein.sequence.properties.residues.**emboss_pepstats_parser**(*infile*)
Get dictionary of pepstats results.

Parameters **infile** – Path to pepstats outfile

Returns Parsed information from pepstats

Return type dict

Todo: Only currently parsing the bottom of the file for percentages of properties.

ssbio.protein.sequence.properties.residues.**flexibility_index**(*aa_one*)
From Smith DK, Radivoja P, ObradovicZ, et al. Improved amino acid flexibility parameters, Protein Sci.2003, 12:1060

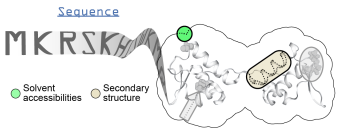
Author: Ke Chen

Parameters **aa_one** –

Returns:

ssbio.protein.sequence.properties.residues.**grantham_score**(*ref_aa*, *mut_aa*)
https://github.com/ashutoshkpandey/Annotation/blob/master/Grantham_score_calculator.py

SCRATCH



Description

- SCRATCH home page
- SCRATCH download page (for SSpro and ACCpro)

SCRATCH is a suite of tools to predict many types of structural properties directly from sequence. *ssbio* contains wrappers to execute and parse results from *SSpro*/*SSpro8* - predictors of secondary structure, and *ACCpro*/*ACCpro20* - predictors of solvent accessibility.

Instructions

Note: These instructions were created on an Ubuntu 17.04 system.

1. Download the source and install it using the perl script:

```
wget http://download.igb.uci.edu/SCRATCH-1D_1.1.tar.gz
tar -xvzf SCRATCH-1D_1.1.tar.gz
cd SCRATCH-1D_1.1
perl install.pl
```

2. To run it from the command line directly:

```
./run_SCRATCH-1D_predictors.sh  input_fasta  output_prefix  [num_threads]
```

3. *ssbio* also provides command line wrappers to run it and parse the results, see *ssbio.protein.sequence.properties.scratch* for details.

FAQs

- How do I cite SCRATCH?
  - Cheng J, Randall AZ, Sweredoski MJ & Baldi P (2005) SCRATCH: a protein structure and structural feature prediction server. Nucleic Acids Res. 33: W72–6 Available at: http://dx.doi.org/10.1093/nar/gki396
- I'm having issues running STRIDE...
  - See the ssbio wiki for (hopefully) some solutions - or add yours in when you find the answer!

API

class ssbio.protein.sequence.properties.scratch.**SCRATCH**(*project_name*, *seq_file=None*, *seq_str=None*)

Provide wrappers for running and parsing SCRATCH on a sequence file or sequence string.

To run from the command line: ./run_SCRATCH-1D_predictors.sh input_fasta output_prefix [num_threads]

SCRATCH predicts:

- **Secondary structure**
  - 3 classes (helix, strand, other) using SSpro
  - 8 classes (standard DSSP definitions) using SSpro8
- **Relative solvent accessibility (RSA, also known as relative accessible surface area)**
  - @ 25% exposed RSA cutoff (<25% RSA means it is buried)
  - @ all cutoffs in 5% increments from 0 to 100

accpro20_results()

Parse the ACCpro output file and return a dict of secondary structure compositions

accpro20_summary(*cutoff*)

Parse the ACCpro output file and return a summary of percent exposed/buried residues based on a cutoff.

Below the cutoff = buried Equal to or greater than cutoff = exposed The default cutoff used in accpro is 25%.

The output file is just a FASTA formatted file, so you can get residue level information by parsing it like a normal sequence file.

Parameters **cutoff** (*float*) – Cutoff for defining a buried or exposed residue.

Returns Percentage of buried and exposed residues

Return type dict

accpro_results()

Parse the ACCpro output file and return a dict of secondary structure compositions.

accpro_summary()
Parse the ACCpro output file and return a summary of percent exposed/buried residues.

The output file is just a FASTA formatted file, so you can get residue level information by parsing it like a normal sequence file.

Returns Percentage of buried and exposed residues

Return type dict

run_scratch(*path_to_scratch*, *num_cores=1*, *outname=None*, *outdir=None*, *force_rerun=False*)
Run SCRATCH on the sequence_file that was loaded into the class.

Parameters

- **path_to_scratch** – Path to the SCRATCH executable, run_SCRATCH-1D_predictors.sh
- **outname** – Prefix to name the output files
- **outdir** – Directory to store the output files
- **force_rerun** – Flag to force rerunning of SCRATCH even if the output files exist

Returns:

sspro8_results()
Parse the SSpro8 output file and return a dict of secondary structure compositions.

sspro8_summary()
Parse the SSpro8 output file and return a summary of secondary structure composition.

The output file is just a FASTA formatted file, so you can get residue level information by parsing it like a normal sequence file.

Returns

Percentage of: H: alpha-helix G: 310-helix I: pi-helix (extremely rare) E: extended strand B: beta-bridge T: turn S: bend C: the rest

Return type dict

sspro_results()
Parse the SSpro output file and return a dict of secondary structure compositions.

Returns

Keys are sequence IDs, values are the lists of secondary structure predictions. H: helix E: strand C: the rest

Return type dict

sspro_summary()
Parse the SSpro output file and return a summary of secondary structure composition.

The output file is just a FASTA formatted file, so you can get residue level information by parsing it like a normal sequence file.

Returns

Percentage of: H: helix E: strand C: the rest

Return type dict

ssbio.protein.sequence.properties.scratch.**read_accpro20**(*infile*)

Read the accpro20 output (.acc20) and return the parsed FASTA records.

Keeps the spaces between the accessibility numbers.

Parameters **infile** – Path to .acc20 file

Returns Dictionary of accessibilities with keys as the ID

Return type dict

FOLD-RATE

Description

- FOLD-RATE home page

This module provides a function to predict the **kinetic folding rate** (k_f) given an amino acid sequence and its structural classficiation (alpha/beta/mixed).

Instructions

1. Obtain your protein's sequence
2. Determine the main secondary structure composition of the protein (`all-alpha`, `all-beta`, `mixed`, or `unknown`)
3. Input the sequence and secondary structure composition into the function `ssbio.protein.sequence.properties.kinetic_folding_rate.get_foldrate()`

FAQs

- What is the main secondary structure composition of my protein?
  - `all-alpha` = dominated by α-helices; $\alpha > 40\%$ and $\beta < 5\%$
  - `all-beta` = dominated by β-strands; $\beta > 40\%$ and $\alpha < 5\%$
  - `mixed` = contain both α-helices and β-strands; $\alpha > 15\%$ and $\beta > 10\%$
- What is the kinetic folding rate?
  - Protein folding rate is a measure of slow/fast folding of proteins from the unfolded state to native three-dimensional structure.
- What units is it in?
  - Number of proteins folded per second
- How can I install FOLD-RATE?
  - FOLD-RATE is only available as a web server. *ssbio* provides a wrapper for the web server and allows you to submit protein sequences to it along with caching the output files.
- How do I cite FOLD-RATE?
  - Gromiha MM, Thangakani AM & Selvaraj S (2006) FOLD-RATE: prediction of protein folding rates from amino acid sequence. Nucleic Acids Res. 34: W70–4 Available at: http://dx.doi.org/10.1093/nar/gkl043
- How can this parameter be used on a genome-scale?

  - See: Chen K, Gao Y, Mih N, O'Brien EJ, Yang L & Palsson BO (2017) Thermosensitivity of growth is determined by chaperone-mediated proteome reallocation. Proceedings of the National Academy of Sciences 114: 11548–11553 Available at: http://www.pnas.org/content/114/43/11548.abstract

- I'm having issues running FOLD-RATE...
  - See the ssbio wiki for (hopefully) some solutions - or add yours in when you find the answer!

API

AMYLPRED2

Description

- AMYLPRED2 home page
- AMYLPRED2 registration link

This module provides a function to predict the **aggregation propensity** of proteins, specifically the number of aggregation-prone segments on an unfolded protein sequence. AMYLPRED2 is a consensus method of different methods. In order to obtain the best balance between sensitivity and specificity, we follow the author's guidelines to consider every 5 consecutive residues agreed among at least 5 methods contributing 1 to the aggregation propensity.

Instructions

1. Create an account on the webserver at the AMYLPRED2 registration link.
2. Create a new AMYLPRED object with your email and password initialized along with it.
3. Run `ssbio.protein.sequence.properties.aggregation_propensity.AMYLPRED.get_aggregation_propensity()` on a protein sequence.

FAQs

- What is aggregation propensity?
  - The number of aggregation-prone segments on an unfolded protein sequence.
- How can I install AMYLPRED2?
  - AMYLPRED2 is only available as a web server. *ssbio* provides a wrapper for the web server and allows you to submit protein sequences to it along with caching the output files.
- How do I cite AMYLPRED2?
  - Tsolis AC, Papandreou NC, Iconomidou VA & Hamodrakas SJ (2013) A consensus method for the prediction of 'aggregation-prone' peptides in globular proteins. PLoS One 8: e54175 Available at: http://dx.doi.org/10.1371/journal.pone.0054175
- How can this parameter be used on a genome-scale?
  - See: Chen K, Gao Y, Mih N, O'Brien EJ, Yang L & Palsson BO (2017) Thermosensitivity of growth is determined by chaperone-mediated proteome reallocation. Proceedings of the National Academy of Sciences 114: 11548–11553 Available at: http://www.pnas.org/content/114/43/11548.abstract
- I'm having issues running AMYLPRED2...
  - See the ssbio wiki for (hopefully) some solutions - or add yours in when you find the answer!

API

class ssbio.protein.sequence.properties.aggregation_propensity.**AMYLPRED**(*email*, *password*)

Class to submit sequences to AMYLPRED2.

Instructions:

1. Create an account on the webserver at the AMYLPRED2 registration link.
2. Create a new AMYLPRED object with your email and password initialized along with it.
3. Run `get_aggregation_propensity` on a protein sequence.

email
str – Account email

password
str – Account password

Todo:

- Properly implement force_rerun and caching functions

---

get_aggregation_propensity(*seq*, *outdir*, *cutoff_v=5*, *cutoff_n=5*, *run_amylmuts=False*)
Run the AMYLPRED2 web server for a protein sequence and get the consensus result for aggregation propensity.

Parameters

- **seq** (*str, Seq, SeqRecord*) – Amino acid sequence
- **outdir** (*str*) – Directory to where output files should be saved
- **cutoff_v** (*int*) – The minimal number of methods that agree on a residue being a aggregation-prone residue
- **cutoff_n** (*int*) – The minimal number of consecutive residues to be considered as a 'stretch' of aggregation-prone region
- **run_amylmuts** (*bool*) – If AMYLMUTS method should be run, default False. AMYLMUTS is optional as it is the most time consuming and generates a slightly different result every submission.

Returns Aggregation propensity - the number of aggregation-prone segments on an unfolded protein sequence

Return type int

parse_method_results(*results_file*, *met*)
Parse the output of a AMYLPRED2 result file.

run_amylpred2(*seq*, *outdir*, *run_amylmuts=False*)
Run all methods on the AMYLPRED2 web server for an amino acid sequence and gather results.

Result files are cached in `/path/to/outdir/AMYLPRED2_results`.

Parameters

- **seq** (*str*) – Amino acid sequence as a string
- **outdir** (*str*) – Directory to where output files should be saved

- **run_amylmuts** (*bool*) – If AMYLMUTS method should be run, default False

Returns Result for each method run

Return type dict

TMHMM

Description

- TMHMM home page
- TMHMM download page
- TMHMM installation instructions

TMHMM is a program to predict the location of transmembrane helices in proteins, directly from sequence. *ssbio* provides a wrapper to execute and parse the "long" output format of TMHMM.

Instructions (Unix)

Note: These instructions were created on an Ubuntu 17.04 system.

1. Register for the software (academic license only) at the TMHMM download page
2. Receive instructions to download the software at your email address
3. Download the file *tmhmm-2.0c.Linux.tar.gz*
4. Extract it to a place where you store software
5. Install it according to the TMHMM installation instructions, repeated and annotated below...
   (a) Insert the correct path for perl 5.x in the first line of the scripts bin/tmhmm and bin/tmhmmformat.pl (if not `/usr/local/bin/perl`). Use `which perl` and `perl -v` in the terminal to help find the correct path.
   (b) Make sure you have an executable version of *decodeanhmm* in the bin directory.
   (c) Include the directory containing tmhmm in your path (how do I add something to my dummiesunix-path?)
   (d) Read the TMHMM2.0.guide.html
   (e) Run the program by doing the following:

```
tmhmm my_sequences.fasta
```

FAQs

- How do I cite TMHMM?
  - Krogh A, Larsson B, von Heijne G & Sonnhammer EL (2001) Predicting transmembrane protein topology with a hidden Markov model: application to complete genomes. J. Mol. Biol. 305: 567–580 Available at: http://dx.doi.org/10.1006/jmbi.2000.4315
- I'm having issues running TMHMM...
  - See the ssbio wiki for (hopefully) some solutions - or add yours in when you find the answer!

API

ssbio.protein.sequence.properties.tmhmm.**label_TM_tmhmm_residue_numbers_and_leaflets**(*tmhmm_s*

Determine the residue numbers of the TM-helix residues that cross the membrane and label them by leaflet.

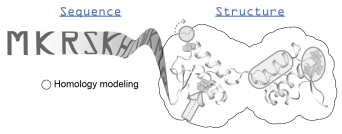
Parameters **tmhmm_seq** – g.protein.representative_sequence.seq_record.letter_annotations['TM-tmhmm']

Returns a dictionary with leaflet_variable : [residue list] where the variable is inside or outside TM_boundary dict: outputs a dictionar with : TM helix number : [TM helix residue start , TM helix residue end]

Return type leaflet_dict

Todo: untested method!

I-TASSER



Description

- Home page: I-TASSER
- Download link: I-TASSER Suite

I-TASSER (Iterative Threading ASSEmbly Refinement) is a program for protein homology modeling and functional prediction from a protein sequence. The I-TASSER suite provides numerous other tools such as for ligand-binding site predictions, model refinement, secondary structure predictions, B-factor estimations, and more. *ssbio* mainly provides tools to run and parse I-TASSER homology modeling results, as well as COACH consensus binding site predictions (optionally with EC number and GO term predictions). Also, scripts are provided to automate homology modeling on a large scale using TORQUE or Slurm job schedulers in a cluster computing environment.

Instructions

Note: These instructions were created on an Ubuntu 17.04 system.

---

1. **Read the README on the I-TASSER Suite page for the most up-to-date instructions**
2. Make sure you have Java installed and it can be run from the command line with `java`
3. Head to the I-TASSER download page and register for an license (academic only) to get a password emailed to you
4. Log in to the I-TASSER download page and download the archive
5. Unpack the software archive into a convenient directory - a library should also be downloaded to this directory
6. Run download_lib.pl to then download the library files - this will take some time:

```
/path/to/<I-TASSER_directory>/download_lib.pl -libdir ITLIB
```

7. Now, I-TASSER can be run according to the README under section 4
8. **To enable GO term predictions...**
    (a) under construction...
9. Tip: to update template libraries, create a new command in your crontab (first run `crontab -e`), and make sure to replace `<USERNAME>` with your username:

```
0 4 * * 1,5 <USERNAME> /path/to/I-TASSER4.4/download_lib.pl -libdir /path/
↪to/ITLIB
```

That will run the library update at 4 am every Monday and Friday.

FAQs

- What is a homology model?
  - A predicted 3D structure model of a protein sequence. Models can be template-based, when they are based on an existing experimental structure; or *ab initio*, generated without a template. Generally, *ab initio* models are much less reliable.
- Can I just run I-TASSER using their web server and parse those results with *ssbio*?
  - Not yet, but you can manually input the model1.pdb file as a new structure for now.
- How do I cite I-TASSER?
  - Roy A, Kucukural A & Zhang Y (2010) I-TASSER: a unified platform for automated protein structure and function prediction. Nat. Protoc. 5: 725–738 Available at: http://dx.doi.org/10.1038/nprot.2010.5
- **How do I run I-TASSER with TORQUE or Slurm job schedulers?**
  - under construction...
- I'm having issues running I-TASSER...
  - See the ssbio wiki for (hopefully) some solutions - or add yours in when you find the answer!

API

class ssbio.protein.structure.homology.itasser.itasserprep.**ITASSERPrep**(*ident*, *seq_str*, *root_dir*, *itasser_path*, *itlib_path*, *execute_dir=None*, *light=True*, *runtype='local'*, *print_exec=False*, *java_home=None*, *binding_site_pred=False*, *ec_pred=False*, *go_pred=False*, *additional_options=None*, *job_scheduler_header=None*)

Prepare a sequence for I-TASSER runs

prep_folder(*seq*)

Take in a sequence string and prepares the folder for the I-TASSER run

class ssbio.protein.structure.homology.itasser.itasserprop.**ITASSERProp**(*ident*, *original_results_path*, *coach_results_folder='model*, *model_to_use='model1'*)

Parse all available information for a local I-TASSER modeling run.

Initializes a class to collect I-TASSER modeling information and optionally copy results to a new directory. SEE: https://zhanglab.ccmb.med.umich.edu/papers/2015_1.pdf for detailed information.

Parameters

- **ident** (*str*) – ID of I-TASSER modeling run
- **original_results_path** (*str*) – Path to I-TASSER modeling folder
- **coach_results_folder** (*str*) – Path to original COACH results
- **model_to_use** (*str*) – Which I-TASSER model to use. Default is "model1"

copy_results(*copy_to_dir*, *rename_model_to=None*, *force_rerun=False*)

Copy the raw information from I-TASSER modeling to a new folder.

Copies all files in the list _attrs_to_copy.

Parameters

- **copy_to_dir** (*str*) – Directory to copy the minimal set of results per sequence.
- **rename_model_to** (*str*) – New file name (without extension)
- **force_rerun** (*bool*) – If existing models and results should be overwritten.

get_dict(*only_attributes=None*, *exclude_attributes=None*, *df_format=False*)
Summarize the I-TASSER run in a dictionary containing modeling results and top predictions from COACH

Parameters

- **only_attributes** (*str, list*) – Attributes that should be returned. If not provided, all are returned.
- **exclude_attributes** (*str, list*) – Attributes that should be excluded.
- **df_format** (*bool*) – If dictionary values should be formatted for a dataframe (everything possible is transformed into strings, int, or float - if something can't be transformed it is excluded)

Returns Dictionary of attributes

Return type dict

ssbio.protein.structure.homology.itasser.itasserprop.**parse_bfp_dat**(*infile*)
Parse the B-factor predictions in BFP.dat

Parameters **infile** (*str*) – Path to BFP.dat

Returns List of B-factor predictions for all residues

Return type list

ssbio.protein.structure.homology.itasser.itasserprop.**parse_coach_bsites_inf**(*infile*)
Parse the Bsites.inf output file of COACH and return a list of rank-ordered binding site predictions

Bsites.inf contains the summary of COACH clustering results after all other prediction algorithms have finished For each site (cluster), there are three lines: Line 1: site number, c-score of coach prediction, cluster size Line 2: algorithm, PDB ID, ligand ID, center of binding site (cartesian coordinates),

c-score of the algorithm's prediction, binding residues from single template

Line 3: Statistics of ligands in the cluster

C-score information "In our training data, a prediction with C-score>0.35 has average false positive and false negative rates below 0.16 and 0.13, respectively." (https://zhanglab.ccmb.med.umich.edu/COACH/COACH.pdf)

Parameters **infile** (*str*) – Path to Bsites.inf

Returns

Ranked list of dictionaries, keys defined below site_num: cluster which is the consensus binding site c_score: confidence score of the cluster prediction cluster_size: number of predictions within this cluster algorithm: main? algorithm used to make the prediction pdb_template_id: PDB ID of the template used to make the prediction pdb_template_chain: chain of the PDB which has the ligand pdb_ligand: predicted ligand to bind binding_location_coords: centroid of the predicted ligand position in the homology model c_score_method: confidence score for the main algorithm binding_residues: predicted residues to bind the ligand ligand_cluster_counts: number of predictions per ligand

Return type list

ssbio.protein.structure.homology.itasser.itasserprop.**parse_coach_ec**(*infile*)
Parse the EC.dat output file of COACH and return a list of rank-ordered EC number predictions

EC.dat contains the predicted EC number and active residues. The columns are: PDB_ID, TM-score, RMSD, Sequence identity, Coverage, Confidence score, EC number, and Active site residues

Parameters **infile** (*str*) – Path to EC.dat

Returns

Ranked list of dictionaries, keys defined below pdb_template_id: PDB ID of the template used to make the prediction pdb_template_chain: chain of the PDB which has the ligand tm_score: TM-score of the template to the model (similarity score) rmsd: RMSD of the template to the model (also a measure of similarity) seq_ident: percent sequence identity seq_coverage: percent sequence coverage c_score: confidence score of the EC prediction ec_number: predicted EC number binding_residues: predicted residues to bind the ligand

Return type list

ssbio.protein.structure.homology.itasser.itasserprop.**parse_coach_ec_df**(*infile*)
Parse the EC.dat output file of COACH and return a dataframe of results

EC.dat contains the predicted EC number and active residues. The columns are: PDB_ID, TM-score, RMSD, Sequence identity, Coverage, Confidence score, EC number, and Active site residues

Parameters **infile** (*str*) – Path to EC.dat

Returns Pandas DataFrame summarizing EC number predictions

Return type DataFrame

ssbio.protein.structure.homology.itasser.itasserprop.**parse_coach_go**(*infile*)
Parse a GO output file from COACH and return a rank-ordered list of GO term predictions

The columns in all files are: GO terms, Confidence score, Name of GO terms. GO_MF.dat - GO terms in 'molecular function' GO_BP.dat - GO terms in 'biological process' GO_CC.dat - GO terms in 'cellular component'

Parameters **infile** (*str*) – Path to any COACH GO prediction file

Returns

Organized dataframe of results, columns defined below go_id: GO term ID go_term: GO term text c_score: confidence score of the GO prediction

Return type Pandas DataFrame

ssbio.protein.structure.homology.itasser.itasserprop.**parse_cscore**(*infile*)
Parse the cscore file to return a dictionary of scores.

Parameters **infile** (*str*) – Path to cscore

Returns Dictionary of scores

Return type dict

ssbio.protein.structure.homology.itasser.itasserprop.**parse_exp_dat**(*infile*)
Parse the solvent accessibility predictions in exp.dat

Parameters **infile** (*str*) – Path to exp.dat

Returns List of solvent accessibility predictions for all residues

Return type list

ssbio.protein.structure.homology.itasser.itasserprop.**parse_init_dat**(*infile*)
Parse the main init.dat file which contains the modeling results

The first line of the file init.dat contains stuff like: “120 easy 40 8” The other lines look like this: ” 161 11.051 1 1guqA MUSTER” and getting the first 10 gives you the top 10 templates used in modeling

Parameters **infile** (*stt*) – Path to init.dat

Returns Dictionary of parsed information

Return type dict

ssbio.protein.structure.homology.itasser.itasserprop.**parse_seq_dat**(*infile*)

Parse the secondary structure predictions in seq.dat

Parameters **infile** (*str*) – Path to seq.dat

Returns List of secondary structure predictions for all residues

Return type list

5.6 Software

Analysis level	Function type	Name	Function	Internal Python functions	External software	Web ser
Network model or a set of proteins	Pipeline	GEM-PRO	Pipeline to automatically map gene IDs, protein sequences, or GEMs to available experimental structures. Enables streamlined analysis for all functions described below for individual proteins.	*The GEM-PRO Pipeline*		
Protein sequence	Sequence-based calculation	Various sequence properties	Basic properties of the sequence, such as percent of polar, non-polar, hydrophobic or hydrophilic residues.	Biopython ProteinAnalysis	*EMBOSS*	
		Sequence alignment	Basic functions to run pairwise or multiple sequence alignments	Biopython pairwise2	*EMBOSS*	
	Sequence-based prediction	Aggregation propensity	Consensus method to predict the aggregation propensity of proteins, specifically the number of aggregation-prone segments on an unfolded protein sequence			*AMYLPR*
		Secondary structure and solvent accessibilities	Predictions of secondary structure and relative solvent accessibilities per residue		*SCRATCH*	
		Thermostability	Free energy of unfolding (ΔG), adapted from Oobatake (Oobatake & Ooi 1993) and Dill (Dill et al. 2011)	ssbio custom functions		
		Transmembrane domains	Prediction of transmembrane domains from sequence		*TMHMM*	
Protein structure	Sequence-based prediction	Homology modeling	Preparation scripts and parsers for		*I-TASSER*	

5.7 Python API

Information on select functions, classes, or methods.

5.7.1 GEMPRO

class ssbio.pipeline.gempro.**GEMPRO**(*gem_name*, *pdb_file_type='mmtf'*, *root_dir=None*, *gem=None*, *gem_file_path=None*, *gem_file_type=None*, *genes_list=None*, *genes_and_sequences=None*, *genome_path=None*, *description=None*, *custom_spont_id=None*)

Generic class to represent all information for a GEM-PRO project.

Initialize the GEM-PRO project with a genome-scale model, a list of genes, or a dict of genes and sequences. Specify the name of your project, along with the root directory where a folder with that name will be created.

Main methods provided are:

1. Automated mapping of sequence IDs
   - With KEGG mapper
   - With UniProt mapper
   - Allowing manual gene ID –> protein sequence entry
   - Allowing manual gene ID –> UniProt ID
2. Consolidating sequence IDs and setting a representative sequence
   - Currently these are set based on available PDB IDs
3. Mapping of representative sequence –> structures
   - With UniProt –> ranking of PDB structures
   - BLAST representative sequence –> PDB database
4. Preparation of files for homology modeling (currently for I-TASSER)
   - Mapping to existing models
   - Preparation for running I-TASSER
   - Parsing I-TASSER runs
5. Running QC/QA on structures and setting a representative structure
   - Various cutoffs (mutations, insertions, deletions) can be set to filter structures
6. Automation of protein sequence and structure property calculation
7. Creation of Pandas DataFrame summaries directly from downloaded metadata

Parameters

- **gem_name** (*str*) – The name of your GEM or just your project in general. This will be the name of the main folder that is created in root_dir.
- **pdb_file_type** (*str*) – `pdb`, `pdb.gz`, `mmcif`, `cif`, `cif.gz`, `xml.gz`, `mmtf`, `mmtf.gz` - choose a file type for files downloaded from the PDB

- **root_dir** (*str*) – Path to where the folder named after `gem_name` will be created. If not provided, directories will not be created and output directories need to be specified for some steps
- **gem** (*Model*) – COBRApy Model object
- **gem_file_path** (*str*) – Path to GEM file
- **gem_file_type** (*str*) – GEM model type - `sbml` (or `xml`), `mat`, or `json` formats
- **genes_list** (*list*) – List of gene IDs that you want to map
- **genes_and_sequences** (*dict*) – Dictionary of gene IDs and their amino acid sequence strings
- **genome_path** (*str*) – Genome FASTA file of protein coding sequences
- **description** (*str*) – Optional string to describe your project
- **custom_spont_id** (*str*) – ID of spontaneous gene

add_genes_by_id(*genes_list*)
Add gene IDs manually into the GEM-PRO project.

Parameters **genes_list** (*list*) – List of gene IDs as strings.

base_dir
str – GEM-PRO project folder.

blast_seqs_to_pdb(*seq_ident_cutoff=0*, *evalue=0.0001*, *all_genes=False*, *display_link=False*, *outdir=None*, *force_rerun=False*)
BLAST each representative protein sequence to the PDB. Saves raw BLAST results (XML files).

Parameters

- **seq_ident_cutoff** (*float, optional*) – Cutoff results based on percent coverage (in decimal form)
- **evalue** (*float, optional*) – Cutoff for the E-value - filters for significant hits. 0.001 is liberal, 0.0001 is stringent (default).
- **all_genes** (*bool*) – If all genes should be BLASTed, or only those without any structures currently mapped
- **display_link** (*bool, optional*) – Set to True if links to the HTML results should be displayed
- **outdir** (*str*) – Path to output directory of downloaded files, must be set if GEM-PRO directories were not created initially
- **force_rerun** (*bool, optional*) – If existing BLAST results should not be used, set to True. Default is False

data_dir
str – Directory where all data are stored.

df_homology_models
DataFrame – Get a dataframe of I-TASSER homology model results

df_kegg_metadata
DataFrame – Pandas DataFrame of KEGG metadata per protein.

df_pdb_blast
DataFrame – Get a dataframe of PDB BLAST results

df_pdb_metadata
DataFrame – Get a dataframe of PDB metadata (PDBs have to be downloaded first).

df_pdb_ranking
DataFrame – Get a dataframe of UniProt -> best structure in PDB results

df_proteins
DataFrame – Get a summary dataframe of all proteins in the project.

df_representative_sequences
DataFrame – Pandas DataFrame of representative sequence information per protein.

df_representative_structures
DataFrame – Get a dataframe of representative protein structure information.

df_uniprot_metadata
DataFrame – Pandas DataFrame of UniProt metadata per protein.

find_disulfide_bridges(*representatives_only=True*)
Run Biopython's disulfide bridge finder and store found bridges.

Annotations are stored in the protein structure's chain sequence at: `<chain_prop>.seq_record.annotations['SSBOND-biopython']`

Parameters **representative_only** (*bool*) – If analysis should only be run on the representative structure

genes
DictList – All genes excluding spontaneous ones.

genes_dir
str – Directory where all gene specific information is stored.

genes_with_a_representative_sequence
DictList – All genes with a representative sequence.

genes_with_a_representative_structure
DictList – All genes with a representative protein structure.

genes_with_experimental_structures
DictList – All genes that have at least one experimental structure.

genes_with_homology_models
DictList – All genes that have at least one homology model.

genes_with_structures
DictList – All genes with any mapped protein structures.

get_dssp_annotations(*representatives_only=True*, *force_rerun=False*)
Run DSSP on structures and store calculations.

Annotations are stored in the protein structure's chain sequence at: `<chain_prop>.seq_record.letter_annotations['*-dssp']`

Parameters

- **representative_only** (*bool*) – If analysis should only be run on the representative structure
- **force_rerun** (*bool*) – If calculations should be rerun even if an output file exists

get_freesasa_annotations(*include_hetatms=False*, *representatives_only=True*, *force_rerun=False*)
Run freesasa on structures and store calculations.

Annotations are stored in the protein structure's chain sequence at: `<chain_prop>.seq_record.letter_annotations['*-freesasa']`

Parameters

- **include_hetatms** (*bool*) – If HETATMs should be included in calculations. Defaults to `False`.
- **representative_only** (*bool*) – If analysis should only be run on the representative structure
- **force_rerun** (*bool*) – If calculations should be rerun even if an output file exists

get_itasser_models(*homology_raw_dir*, *custom_itasser_name_mapping=None*, *outdir=None*, *force_rerun=False*)

Copy generated I-TASSER models from a directory to the GEM-PRO directory.

Parameters

- **homology_raw_dir** (*str*) – Root directory of I-TASSER folders.
- **custom_itasser_name_mapping** (*dict*) – Use this if your I-TASSER folder names differ from your model gene names. Input a dict of {model_gene: ITASSER_folder}.
- **outdir** (*str*) – Path to output directory of downloaded files, must be set if GEM-PRO directories were not created initially
- **force_rerun** (*bool*) – If homology files should be copied again even if they exist in the GEM-PRO directory

get_manual_homology_models(*input_dict*, *outdir=None*, *clean=True*, *force_rerun=False*)

Copy homology models to the GEM-PRO project.

Requires an input of a dictionary formatted like so:

```
{
    model_gene: {
                    homology_model_id1: {
                                            'model_file': '/path/to/homology/
↪model.pdb',
                                            'file_type': 'pdb'
                                            'additional_info': info_value
                                        },
                    homology_model_id2: {
                                            'model_file': '/path/to/homology/
↪model.pdb'
                                            'file_type': 'pdb'
                                        }
                }
}
```

Parameters

- **input_dict** (*dict*) – Dictionary of dictionaries of gene names to homology model IDs and other information
- **outdir** (*str*) – Path to output directory of downloaded files, must be set if GEM-PRO directories were not created initially
- **clean** (*bool*) – If homology files should be cleaned and saved as a new PDB file

- **force_rerun** (*bool*) – If homology files should be copied again even if they exist in the GEM-PRO directory

get_msms_annotations(*representatives_only=True*, *force_rerun=False*)
Run MSMS on structures and store calculations.

Annotations are stored in the protein structure's chain sequence at: `<chain_prop>.seq_record.letter_annotations['*-msms']`

Parameters

- **representative_only** (*bool*) – If analysis should only be run on the representative structure
- **force_rerun** (*bool*) – If calculations should be rerun even if an output file exists

get_scratch_predictions(*path_to_scratch*, *results_dir*, *scratch_basename='scratch'*, *num_cores=1*, *exposed_buried_cutoff=25*, *custom_gene_mapping=None*)
Run and parse `SCRATCH` results to predict secondary structure and solvent accessibility. Annotations are stored in the protein's representative sequence at:

- `.annotations`
- `.letter_annotations`

Parameters

- **path_to_scratch** (*str*) – Path to SCRATCH executable
- **results_dir** (*str*) – Path to SCRATCH results folder, which will have the files (scratch.ss, scratch.ss8, scratch.acc, scratch.acc20)
- **scratch_basename** (*str*) – Basename of the SCRATCH results ('scratch' is default)
- **num_cores** (*int*) – Number of cores to use to parallelize SCRATCH run
- **exposed_buried_cutoff** (*int*) – Cutoff of exposed/buried for the acc20 predictions
- **custom_gene_mapping** (*dict*) – Default parsing of SCRATCH output files is to look for the model gene IDs. If your output files contain IDs which differ from the model gene IDs, use this dictionary to map model gene IDs to result file IDs. Dictionary keys must match model genes.

get_sequence_properties(*representatives_only=True*)
Run Biopython ProteinAnalysis and EMBOSS pepstats to summarize basic statistics of all protein sequences. Results are stored in the protein's respective SeqProp objects at `.annotations`

Parameters **representative_only** (*bool*) – If analysis should only be run on the representative sequences

get_tmhmm_predictions(*tmhmm_results*, *custom_gene_mapping=None*)
Parse TMHMM results and store in the representative sequences.

This is a basic function to parse pre-run TMHMM results. Run TMHMM from the web service (http://www.cbs.dtu.dk/services/TMHMM/) by doing the following:

1. Write all representative sequences in the GEM-PRO using the function `write_representative_sequences_file`
2. Upload the file to http://www.cbs.dtu.dk/services/TMHMM/ and choose "Extensive, no graphics" as the output

3. Copy and paste the results (ignoring the top header and above "HELP with output formats") into a file and save it
4. Run this function on that file

Parameters

- **tmhmm_results** (*str*) – Path to TMHMM results (long format)
- **custom_gene_mapping** (*dict*) – Default parsing of TMHMM output is to look for the model gene IDs. If your output file contains IDs which differ from the model gene IDs, use this dictionary to map model gene IDs to result file IDs. Dictionary keys must match model genes.

kegg_mapping_and_metadata(*kegg_organism_code*, *custom_gene_mapping=None*, *outdir=None*, *set_as_representative=False*, *force_rerun=False*)

Map all genes in the model to KEGG IDs using the KEGG service.

Steps:

1. Download all metadata and sequence files in the sequences directory
2. Creates a KEGGProp object in the protein.sequences attribute
3. Returns a Pandas DataFrame of mapping results

Parameters

- **kegg_organism_code** (*str*) – The three letter KEGG code of your organism
- **custom_gene_mapping** (*dict*) – If your model genes differ from the gene IDs you want to map, custom_gene_mapping allows you to input a dictionary which maps model gene IDs to new ones. Dictionary keys must match model gene IDs.
- **outdir** (*str*) – Path to output directory of downloaded files, must be set if GEM-PRO directories were not created initially
- **set_as_representative** (*bool*) – If mapped KEGG IDs should be set as representative sequences
- **force_rerun** (*bool*) – If you want to overwrite any existing mappings and files

load_cobra_model(*model*)

Load a COBRApy Model object into the GEM-PRO project.

Parameters **model** (*Model*) – COBRApy `Model` object

manual_seq_mapping(*gene_to_seq_dict*, *outdir=None*, *set_as_representative=True*)

Read a manual input dictionary of model gene IDs –> protein sequences. By default sets them as representative.

Parameters

- **gene_to_seq_dict** (*dict*) – Mapping of gene IDs to their protein sequence strings
- **outdir** (*str*) – Path to output directory of downloaded files, must be set if GEM-PRO directories were not created initially
- **set_as_representative** (*bool*) – If mapped sequences should be set as representative

manual_uniprot_mapping(*gene_to_uniprot_dict*, *outdir=None*, *set_as_representative=True*)

Read a manual dictionary of model gene IDs –> UniProt IDs. By default sets them as representative.

This allows for mapping of the missing genes, or overriding of automatic mappings.

Input a dictionary of:

```
{
    <gene_id1>: <uniprot_id1>,
    <gene_id2>: <uniprot_id2>,
}
```

Parameters

- **gene_to_uniprot_dict** – Dictionary of mappings as shown above
- **outdir** (*str*) – Path to output directory of downloaded files, must be set if GEM-PRO directories were not created initially
- **set_as_representative** (*bool*) – If mapped UniProt IDs should be set as representative sequences

map_uniprot_to_pdb(*seq_ident_cutoff=0.0*, *outdir=None*, *force_rerun=False*)

Map all representative sequences' UniProt ID to PDB IDs using the PDBe "Best Structures" API. Will save a JSON file of the results to each protein's `sequences` folder.

The "Best structures" API is available at https://www.ebi.ac.uk/pdbe/api/doc/sifts.html The list of PDB structures mapping to a UniProt accession sorted by coverage of the protein and, if the same, resolution.

Parameters

- **seq_ident_cutoff** (*float*) – Sequence identity cutoff in decimal form
- **outdir** (*str*) – Output directory to cache JSON results of search
- **force_rerun** (*bool*) – Force re-downloading of JSON results if they already exist

Returns A rank-ordered list of PDBProp objects that map to the UniProt ID

Return type list

missing_homology_models

list – List of genes with no mapping to any homology models.

missing_kegg_mapping

list – List of genes with no mapping to KEGG.

missing_pdb_structures

list – List of genes with no mapping to any experimental PDB structure.

missing_representative_sequence

list – List of genes with no mapping to a representative sequence.

missing_representative_structure

list – List of genes with no mapping to a representative structure.

missing_uniprot_mapping

list – List of genes with no mapping to UniProt.

model_dir

str – Directory where original GEMs and GEM-related files are stored.

pdb_downloader_and_metadata(*outdir=None*, *pdb_file_type=None*, *force_rerun=False*)

Download ALL mapped experimental structures to each protein's structures directory.

Parameters

- **outdir** (*str*) – Path to output directory, if GEM-PRO directories were not set or other output directory is desired
- **pdb_file_type** (*str*) – Type of PDB file to download, if not already set or other format is desired
- **force_rerun** (*bool*) – If files should be re-downloaded if they already exist

prep_itasser_modeling(*itasser_installation*, *itlib_folder*, *runtype*, *create_in_dir=None*, *execute_from_dir=None*, *all_genes=False*, *print_exec=False*, ***kwargs*)

Prepare to run I-TASSER homology modeling for genes without structures, or all genes.

Parameters

- **itasser_installation** (*str*) – Path to I-TASSER folder, i.e. `~/software/I-TASSER4.4`
- **itlib_folder** (*str*) – Path to ITLIB folder, i.e. `~/software/ITLIB`
- **runtype** – How you will be running I-TASSER - local, slurm, or torque
- **create_in_dir** (*str*) – Local directory where folders will be created
- **execute_from_dir** (*str*) – Optional path to execution directory - use this if you are copying the homology models to another location such as a supercomputer for running
- **all_genes** (*bool*) – If all genes should be prepped, or only those without any mapped structures
- **print_exec** (*bool*) – If the execution statement should be printed to run modelling

---

Todo:

- Document kwargs - extra options for I-TASSER, SLURM or Torque execution
- Allow modeling of any sequence in sequences attribute, select by ID or provide SeqProp?

---

root_dir

str – Directory where GEM-PRO project folder named after the attribute `base_dir` is located.

set_representative_sequence(*force_rerun=False*)

Automatically consolidate loaded sequences (manual, UniProt, or KEGG) and set a single representative sequence.

Manually set representative sequences override all existing mappings. UniProt mappings override KEGG mappings except when KEGG mappings have PDBs associated with them and UniProt doesn't.

Parameters **force_rerun** (*bool*) – Set to True to recheck stored sequences

set_representative_structure(*seq_outdir=None*, *struct_outdir=None*, *pdb_file_type=None*, *engine='needle'*, *always_use_homology=False*, *rez_cutoff=0.0*, *seq_ident_cutoff=0.5*, *allow_missing_on_termini=0.2*, *allow_mutants=True*, *allow_deletions=False*, *allow_insertions=False*, *allow_unresolved=True*, *clean=True*, *force_rerun=False*)

Set all representative structure for proteins from a structure in the structures attribute.

Each gene can have a combination of the following, which will be analyzed to set a representative structure.

- Homology model(s)
- Ranked PDBs
- BLASTed PDBs

If the `always_use_homology` flag is true, homology models are always set as representative when they exist. If there are multiple homology models, we rank by the percent sequence coverage.

Parameters

- **seq_outdir** (*str*) – Path to output directory of sequence alignment files, must be set if GEM-PRO directories were not created initially
- **struct_outdir** (*str*) – Path to output directory of structure files, must be set if GEM-PRO directories were not created initially
- **pdb_file_type** (*str*) – `pdb`, `pdb.gz`, `mmcif`, `cif`, `cif.gz`, `xml.gz`, `mmtf`, `mmtf.gz` - choose a file type for files downloaded from the PDB
- **engine** (*str*) – `biopython` or `needle` - which pairwise alignment program to use. `needle` is the standard EMBOSS tool to run pairwise alignments. `biopython` is Biopython's implementation of needle. Results can differ!
- **always_use_homology** (*bool*) – If homology models should always be set as the representative structure
- **rez_cutoff** (*float*) – Resolution cutoff, in Angstroms (only if experimental structure)
- **seq_ident_cutoff** (*float*) – Percent sequence identity cutoff, in decimal form
- **allow_missing_on_termini** (*float*) – Percentage of the total length of the reference sequence which will be ignored when checking for modifications. Example: if 0.1, and reference sequence is 100 AA, then only residues 5 to 95 will be checked for modifications.
- **allow_mutants** (*bool*) – If mutations should be allowed or checked for
- **allow_deletions** (*bool*) – If deletions should be allowed or checked for
- **allow_insertions** (*bool*) – If insertions should be allowed or checked for
- **allow_unresolved** (*bool*) – If unresolved residues should be allowed or checked for
- **clean** (*bool*) – If structures should be cleaned
- **force_rerun** (*bool*) – If sequence to structure alignment should be rerun

uniprot_mapping_and_metadata(*model_gene_source*, *custom_gene_mapping=None*, *outdir=None*, *set_as_representative=False*, *force_rerun=False*)

Map all genes in the model to UniProt IDs using the UniProt mapping service. Also download all metadata and sequences.

Parameters

- **model_gene_source** (*str*) – the database source of your model gene IDs. See: http://www.uniprot.org/help/api_idmapping Common model gene sources are:
  - Ensembl Genomes - `ENSEMBLGENOME_ID` (i.e. E. coli b-numbers)
  - Entrez Gene (GeneID) - `P_ENTREZGENEID`
  - RefSeq Protein - `P_REFSEQ_AC`
- **custom_gene_mapping** (*dict*) – If your model genes differ from the gene IDs you want to map, custom_gene_mapping allows you to input a dictionary which maps model gene IDs to new ones. Dictionary keys must match model genes.
- **outdir** (*str*) – Path to output directory of downloaded files, must be set if GEM-PRO directories were not created initially

- **set_as_representative** (*bool*) – If mapped UniProt IDs should be set as representative sequences
- **force_rerun** (*bool*) – If you want to overwrite any existing mappings and files

write_representative_sequences_file(*outname*, *outdir=None*, *set_ids_from_model=True*)

Write all the model's sequences as a single FASTA file. By default, sets IDs to model gene IDs.

Parameters

- **outname** (*str*) – Name of the output FASTA file without the extension
- **outdir** (*str*) – Path to output directory of downloaded files, must be set if GEM-PRO directories were not created initially
- **set_ids_from_model** (*bool*) – If the gene ID source should be the model gene IDs, not the original sequence ID

5.7.2 Protein

class ssbio.core.protein.**Protein**(*ident*, *description=None*, *root_dir=None*, *pdb_file_type='mmtf'*)

Store information on a protein that represents the translated unit of a gene.

The main utilities of this class are to:

1. Load, parse, and store the same (ie. from different database sources) or similar (ie. from different strains) protein sequences as `SeqProp` objects in the `sequences` attribute
2. Load, parse, and store multiple experimental or predicted protein structures as `StructProp` objects in the `structures` attribute
3. Set a single representative sequence and structure
4. Calculate, store, and access pairwise sequence alignments to the representative sequence or structure
5. Provide summaries of alignments and mutations seen
6. Map between residue numbers of sequences and structures

Parameters

- **ident** (*str*) – Unique identifier for this protein
- **description** (*str*) – Optional description for this protein
- **root_dir** (*str*) – Path to where the folder named by this protein's ID will be created. Default is current working directory.
- **pdb_file_type** (*str*) – `pdb`, `pdb.gz`, `mmcif`, `cif`, `cif.gz`, `xml.gz`, `mmtf`, `mmtf.gz` - choose a file type for files downloaded from the PDB

Todo:

- Implement structural alignment objects

---

add_features_to_nglview(*view*, *seqprop=None*, *structprop=None*, *chain_id=None*, *use_representatives=False*)

Add select features from the selected SeqProp object to an NGLWidget view object.

Currently parsing for:

- Single residue features (ie. metal binding sites)
- Disulfide bonds

Parameters

- **view** (*NGLWidget*) – NGLWidget view object
- **seqprop** (SeqProp) – SeqProp object
- **structprop** (StructProp) – StructProp object
- **chain_id** (*str*) – ID of the structure's chain to get annotation from
- **use_representatives** (*bool*) – If the representative sequence/structure/chain IDs should be used

add_fingerprint_to_nglview(*view*, *fingerprint*, *seqprop=None*, *structprop=None*, *chain_id=None*, *use_representatives=False*, *color='red'*, *opacity_range=(0.8, 1)*, *scale_range=(1, 5)*)

Add representations to an NGLWidget view object for residues that are mutated in the `sequence_alignments` attribute.

Parameters

- **view** (*NGLWidget*) – NGLWidget view object
- **fingerprint** (*dict*) – Single mutation group from the `sequence_mutation_summary` function
- **seqprop** (SeqProp) – SeqProp object
- **structprop** (StructProp) – StructProp object
- **chain_id** (*str*) – ID of the structure's chain to get annotation from
- **use_representatives** (*bool*) – If the representative sequence/structure/chain IDs should be used
- **color** (*str*) – Color of the mutations (overridden if unique_colors=True)
- **opacity_range** (*tuple*) – Min/max opacity values (mutations that show up more will be opaque)
- **scale_range** (*tuple*) – Min/max size values (mutations that show up more will be bigger)

add_mutations_to_nglview(*view*, *alignment_type='seqalign'*, *alignment_ids=None*, *seqprop=None*, *structprop=None*, *chain_id=None*, *use_representatives=False*, *grouped=False*, *color='red'*, *unique_colors=True*, *opacity_range=(0.8, 1)*, *scale_range=(1, 5)*)

Add representations to an NGLWidget view object for residues that are mutated in the `sequence_alignments` attribute.

Parameters

- **view** (*NGLWidget*) – NGLWidget view object
- **alignment_type** (*str*) – Specified alignment type contained in the `annotation` field of an alignment object, `seqalign` or `structalign` are the current types.
- **alignment_ids** (*str, list*) – Specified alignment ID or IDs to use
- **seqprop** (SeqProp) – SeqProp object

- **structprop** (StructProp) – StructProp object
- **chain_id** (*str*) – ID of the structure's chain to get annotation from
- **use_representatives** (*bool*) – If the representative sequence/structure/chain IDs should be used
- **grouped** (*bool*) – If groups of mutations should be colored and sized together
- **color** (*str*) – Color of the mutations (overridden if unique_colors=True)
- **unique_colors** (*bool*) – If each mutation/mutation group should be colored uniquely
- **opacity_range** (*tuple*) – Min/max opacity values (mutations that show up more will be opaque)
- **scale_range** (*tuple*) – Min/max size values (mutations that show up more will be bigger)

align_seqprop_to_structprop(*seqprop*, *structprop*, *chains=None*, *outdir=None*, *engine='needle'*, *parse=True*, *force_rerun=False*, ***kwargs*)

Run and store alignments of a SeqProp to chains in the `mapped_chains` attribute of a StructProp.

Alignments are stored in the sequence_alignments attribute, with the IDs formatted as `<SeqProp_ID>_<StructProp_ID>-<Chain_ID>`. Although it is more intuitive to align to individual ChainProps, StructProps should be loaded as little as possible to reduce run times so the alignment is done to the entire structure.

Parameters

- **seqprop** (SeqProp) – SeqProp object with a loaded sequence
- **structprop** (StructProp) – StructProp object with a loaded structure
- **chains** (*str, list*) – Chain ID or IDs to map to. If not specified, `mapped_chains` attribute is inspected for chains. If no chains there, all chains will be aligned to.
- **outdir** (*str*) – Directory to output sequence alignment files (only if running with needle)
- **engine** (*str*) – `biopython` or `needle` - which pairwise alignment program to use. `needle` is the standard EMBOSS tool to run pairwise alignments. `biopython` is Biopython's implementation of needle. Results can differ!
- **parse** (*bool*) – Store locations of mutations, insertions, and deletions in the alignment object (as an annotation)
- **force_rerun** (*bool*) – If alignments should be rerun
- ****kwargs** – Other alignment options

Todo:

- Document **kwargs for alignment options

---

blast_representative_sequence_to_pdb(*seq_ident_cutoff=0*, *evalue=0.0001*, *display_link=False*, *outdir=None*, *force_rerun=False*)

BLAST the representative protein sequence to the PDB. Saves a raw BLAST result file (XML file).

Parameters

- **seq_ident_cutoff** (*float, optional*) – Cutoff results based on percent coverage (in decimal form)

- **evalue** (*float, optional*) – Cutoff for the E-value - filters for significant hits. 0.001 is liberal, 0.0001 is stringent (default).
- **display_link** (*bool, optional*) – Set to True if links to the HTML results should be displayed
- **outdir** (*str*) – Path to output directory of downloaded XML files, must be set if protein directory was not initialized
- **force_rerun** (*bool, optional*) – If existing BLAST results should not be used, set to True. Default is False.

Returns List of new `PDBProp` objects added to the `structures` attribute

Return type list

df_homology_models
DataFrame – Get a dataframe of I-TASSER homology model results

df_pdb_blast
DataFrame – Get a dataframe of PDB BLAST results

df_pdb_metadata
DataFrame – Get a dataframe of PDB metadata (PDBs have to be downloaded first)

df_pdb_ranking
DataFrame – Get a dataframe of UniProt -> best structure in PDB results

filter_sequences(*seq_type*)
Return a DictList of only specified types in the sequences attribute.

Parameters **seq_type** (`SeqProp`) – Object type

Returns A filtered DictList of specified object type only

Return type DictList

find_disulfide_bridges(*representative_only=True*)
Run Biopython's disulfide bridge finder and store found bridges.

Annotations are stored in the protein structure's chain sequence at: `<chain_prop>.seq_record.annotations['SSBOND-biopython']`

Parameters **representative_only** (*bool*) – If analysis should only be run on the representative structure

find_representative_chain(*seqprop*, *structprop*, *chains_to_check=None*, *seq_ident_cutoff=0.5*, *allow_missing_on_termini=0.2*, *allow_mutants=True*, *allow_deletions=False*, *allow_insertions=False*, *allow_unresolved=True*)
Set and return the representative chain based on sequence quality checks to a reference sequence.

Parameters

- **seqprop** (`SeqProp`) – SeqProp object to compare to chain sequences
- **structprop** (`StructProp`) – StructProp object with chains to compare to in the `mapped_chains` attribute. If there are none present, `chains_to_check` can be specified, otherwise all chains are checked.
- **chains_to_check** (*str, list*) – Chain ID or IDs to check for sequence coverage quality
- **seq_ident_cutoff** (*float*) – Percent sequence identity cutoff, in decimal form

- **allow_missing_on_termini** (*float*) – Percentage of the total length of the reference sequence which will be ignored when checking for modifications. Example: if 0.1, and reference sequence is 100 AA, then only residues 5 to 95 will be checked for modifications.
- **allow_mutants** (*bool*) – If mutations should be allowed or checked for
- **allow_deletions** (*bool*) – If deletions should be allowed or checked for
- **allow_insertions** (*bool*) – If insertions should be allowed or checked for
- **allow_unresolved** (*bool*) – If unresolved residues should be allowed or checked for

Returns the best chain ID, if any

Return type str

get_dssp_annotations(*representative_only=True*, *force_rerun=False*)
Run DSSP on structures and store calculations.

Annotations are stored in the protein structure's chain sequence at: `<chain_prop>.seq_record.letter_annotations['*-dssp']`

Parameters

- **representative_only** (*bool*) – If analysis should only be run on the representative structure
- **force_rerun** (*bool*) – If calculations should be rerun even if an output file exists

---

Todo:

- Some errors arise from storing annotations for nonstandard amino acids, need to run DSSP separately for those

---

get_experimental_structures()
DictList: Return a DictList of all experimental structures in self.structures

get_freesasa_annotations(*include_hetatms=False*, *representative_only=True*, *force_rerun=False*)
Run freesasa on structures and store calculations.

Annotations are stored in the protein structure's chain sequence at: `<chain_prop>.seq_record.letter_annotations['*-freesasa']`

Parameters

- **include_hetatms** (*bool*) – If HETATMs should be included in calculations. Defaults to `False`.
- **representative_only** (*bool*) – If analysis should only be run on the representative structure
- **force_rerun** (*bool*) – If calculations should be rerun even if an output file exists

get_homology_models()
DictList: Return a DictList of all homology models in self.structures

get_msms_annotations(*representative_only=True*, *force_rerun=False*)
Run MSMS on structures and store calculations.

Annotations are stored in the protein structure's chain sequence at: `<chain_prop>.seq_record.letter_annotations['*-msms']`

Parameters

- **representative_only** (*bool*) – If analysis should only be run on the representative structure
- **force_rerun** (*bool*) – If calculations should be rerun even if an output file exists

get_residue_annotations(*seq_resnum*, *seqprop=None*, *structprop=None*, *chain_id=None*, *use_representatives=False*)

Get all residue-level annotations stored in the SeqProp `letter_annotations` field for a given residue number.

Uses the representative sequence, structure, and chain ID stored by default. If other properties from other structures are desired, input the proper IDs. An alignment for the given sequence to the structure must be present in the sequence_alignments list.

Parameters

- **seq_resnum** (*int*) – Residue number in the sequence
- **seqprop** (SeqProp) – SeqProp object
- **structprop** (StructProp) – StructProp object
- **chain_id** (*str*) – ID of the structure's chain to get annotation from
- **use_representatives** (*bool*) – If the representative sequence/structure/chain IDs should be used

Returns All available letter_annotations for this residue number

Return type dict

get_sequence_properties(*representative_only=True*)

Run Biopython ProteinAnalysis and EMBOSS pepstats to summarize basic statistics of the protein sequences. Results are stored in the protein's respective SeqProp objects at `.annotations`

Parameters **representative_only** (*bool*) – If analysis should only be run on the representative sequence

load_itasser_folder(*ident*, *itasser_folder*, *organize=False*, *outdir=None*, *organize_name=None*, *set_as_representative=False*, *representative_chain='X'*, *force_rerun=False*)

Load the results folder from an I-TASSER run (local, not from the website) and copy relevant files over to the protein structures directory.

Parameters

- **ident** (*str*) – I-TASSER ID
- **itasser_folder** (*str*) – Path to results folder
- **organize** (*bool*) – If select files from modeling should be copied to the Protein directory
- **outdir** (*str*) – Path to directory where files will be copied and organized to
- **organize_name** (*str*) – Basename of files to rename results to. If not provided, will use id attribute.
- **set_as_representative** – If this structure should be set as the representative structure
- **representative_chain** (*str*) – If `set_as_representative` is `True`, provide the representative chain ID

- **force_rerun** (*bool*) – If the PDB should be reloaded if it is already in the list of structures

Returns The object that is now contained in the structures attribute

Return type *ITASSERProp*

load_kegg(*kegg_id*, *kegg_organism_code=None*, *kegg_seq_file=None*, *kegg_metadata_file=None*, *set_as_representative=False*, *download=False*, *outdir=None*, *force_rerun=False*)

Load a KEGG ID, sequence, and metadata files into the sequences attribute.

Parameters

- **kegg_id** (*str*) – KEGG ID
- **kegg_organism_code** (*str*) – KEGG organism code to prepend to the kegg_id if not part of it already. Example: `eco:b1244`, `eco` is the organism code
- **kegg_seq_file** (*str*) – Path to KEGG FASTA file
- **kegg_metadata_file** (*str*) – Path to KEGG metadata file (raw KEGG format)
- **set_as_representative** (*bool*) – If this KEGG ID should be set as the representative sequence
- **download** (*bool*) – If the KEGG sequence and metadata files should be downloaded if not provided
- **outdir** (*str*) – Where the sequence and metadata files should be downloaded to
- **force_rerun** (*bool*) – If ID should be reloaded and files redownloaded

Returns object contained in the sequences attribute

Return type KEGGProp

load_manual_sequence(*seq*, *ident=None*, *write_fasta_file=False*, *outdir=None*, *set_as_representative=False*, *force_rewrite=False*)

Load a manual sequence given as a string and optionally set it as the representative sequence. Also store it in the sequences attribute.

Parameters

- **seq** (*str, Seq, SeqRecord*) – Sequence string, Biopython Seq or SeqRecord object
- **ident** (*str*) – Optional identifier for the sequence, required if seq is a string. Also will override existing IDs in Seq or SeqRecord objects if set.
- **write_fasta_file** (*bool*) – If this sequence should be written out to a FASTA file
- **outdir** (*str*) – Path to output directory
- **set_as_representative** (*bool*) – If this sequence should be set as the representative one
- **force_rewrite** (*bool*) – If the FASTA file should be overwritten if it already exists

Returns Sequence that was loaded into the `sequences` attribute

Return type *SeqProp*

load_manual_sequence_file(*ident*, *seq_file*, *copy_file=False*, *outdir=None*, *set_as_representative=False*)

Load a manual sequence, given as a FASTA file and optionally set it as the representative sequence. Also store it in the sequences attribute.

Parameters

- **ident** (*str*) – Sequence ID
- **seq_file** (*str*) – Path to sequence FASTA file
- **copy_file** (*bool*) – If the FASTA file should be copied to the protein's sequences folder or the `outdir`, if protein folder has not been set
- **outdir** (*str*) – Path to output directory
- **set_as_representative** (*bool*) – If this sequence should be set as the representative one

Returns Sequence that was loaded into the `sequences` attribute

Return type *SeqProp*

load_pdb(*pdb_id*, *mapped_chains=None*, *pdb_file=None*, *file_type=None*, *is_experimental=True*, *set_as_representative=False*, *representative_chain=None*, *force_rerun=False*)

Load a structure ID and optional structure file into the structures attribute.

Parameters

- **pdb_id** (*str*) – PDB ID
- **mapped_chains** (*str, list*) – Chain ID or list of IDs which you are interested in
- **pdb_file** (*str*) – Path to PDB file
- **file_type** (*str*) – Type of PDB file
- **is_experimental** (*bool*) – If this structure file is experimental
- **set_as_representative** (*bool*) – If this structure should be set as the representative structure
- **representative_chain** (*str*) – If `set_as_representative` is `True`, provide the representative chain ID
- **force_rerun** (*bool*) – If the PDB should be reloaded if it is already in the list of structures

Returns The object that is now contained in the structures attribute

Return type *PDBProp*

load_uniprot(*uniprot_id*, *uniprot_seq_file=None*, *uniprot_xml_file=None*, *download=False*, *outdir=None*, *set_as_representative=False*, *force_rerun=False*)

Load a UniProt ID and associated sequence/metadata files into the sequences attribute.

Sequence and metadata files can be provided, or alternatively downloaded with the download flag set to True. Metadata files will be downloaded as XML files.

Parameters

- **uniprot_id** (*str*) – UniProt ID/ACC
- **uniprot_seq_file** (*str*) – Path to FASTA file
- **uniprot_xml_file** (*str*) – Path to UniProt XML file
- **download** (*bool*) – If sequence and metadata files should be downloaded
- **outdir** (*str*) – Output directory for sequence and metadata files
- **set_as_representative** (*bool*) – If this sequence should be set as the representative one

- **force_rerun** (*bool*) – If files should be redownloaded and metadata reloaded

Returns Sequence that was loaded into the `sequences` attribute

Return type *UniProtProp*

map_seqprop_resnums_to_structprop_resnums(*resnums*, *seqprop=None*, *structprop=None*, *chain_id=None*, *use_representatives=False*)

Map a residue number in any SeqProp to the structure's residue number for a specified chain.

Parameters

- **resnums** (*int, list*) – Residue numbers in the sequence
- **seqprop** (SeqProp) – SeqProp object
- **structprop** (StructProp) – StructProp object
- **chain_id** (*str*) – Chain ID to map to
- **use_representatives** (*bool*) – If the representative sequence and structure should be used. If True, seqprop, structprop, and chain_id do not need to be defined.

Returns Mapping of sequence residue numbers to structure residue numbers

Return type dict

map_structprop_resnums_to_seqprop_resnums(*resnums*, *structprop=None*, *chain_id=None*, *seqprop=None*, *use_representatives=False*)

Map a residue number in any StructProp + chain ID to any SeqProp's residue number.

Parameters

- **resnums** (*int, list*) – Residue numbers in the structure
- **structprop** (StructProp) – StructProp object
- **chain_id** (*str*) – Chain ID to map from
- **seqprop** (SeqProp) – SeqProp object
- **use_representatives** (*bool*) – If the representative sequence and structure should be used. If True, seqprop, structprop, and chain_id do not need to be defined.

Returns Mapping of structure residue numbers to sequence residue numbers

Return type dict

map_uniprot_to_pdb(*seq_ident_cutoff=0.0*, *outdir=None*, *force_rerun=False*)

Map the representative sequence's UniProt ID to PDB IDs using the PDBe "Best Structures" API. Will save a JSON file of the results to the protein sequences folder.

The "Best structures" API is available at https://www.ebi.ac.uk/pdbe/api/doc/sifts.html The list of PDB structures mapping to a UniProt accession sorted by coverage of the protein and, if the same, resolution.

Parameters

- **seq_ident_cutoff** (*float*) – Sequence identity cutoff in decimal form
- **outdir** (*str*) – Output directory to cache JSON results of search
- **force_rerun** (*bool*) – Force re-downloading of JSON results if they already exist

Returns A rank-ordered list of PDBProp objects that map to the UniProt ID

Return type list

num_sequences
int – Return the total number of sequences

num_structures
int – Return the total number of structures

num_structures_experimental
int – Return the total number of experimental structures

num_structures_homology
int – Return the total number of homology models

pairwise_align_sequences_to_representative(*gapopen=10*, *gapextend=0.5*, *outdir=None*, *engine='needle'*, *parse=True*, *force_rerun=False*)
Pairwise all sequences in the sequences attribute to the representative sequence. Stores the alignments in the `sequence_alignments` DictList attribute.

Parameters

- **gapopen** (*int*) – Only for `engine='needle'` - Gap open penalty is the score taken away when a gap is created
- **gapextend** (*float*) – Only for `engine='needle'` - Gap extension penalty is added to the standard gap penalty for each base or residue in the gap
- **outdir** (*str*) – Only for `engine='needle'` - Path to output directory. Default is the protein sequence directory.
- **engine** (*str*) – `biopython` or `needle` - which pairwise alignment program to use. `needle` is the standard EMBOSS tool to run pairwise alignments. `biopython` is Biopython's implementation of needle. Results can differ!
- **parse** (*bool*) – Store locations of mutations, insertions, and deletions in the alignment object (as an annotation)
- **force_rerun** (*bool*) – Only for `engine='needle'` - Default False, set to True if you want to rerun the alignment if outfile exists.

pdb_downloader_and_metadata(*outdir=None*, *pdb_file_type=None*, *force_rerun=False*)
Download ALL mapped experimental structures to the protein structures directory.

Parameters

- **outdir** (*str*) – Path to output directory, if protein structures directory not set or other output directory is desired
- **pdb_file_type** (*str*) – Type of PDB file to download, if not already set or other format is desired
- **force_rerun** (*bool*) – If files should be re-downloaded if they already exist

Returns List of PDB IDs that were downloaded

Return type list

Todo:

- Parse mmtf or PDB file for header information, rather than always getting the cif file for header info

---

pdb_file_type = None
str – `pdb`, `pdb.gz`, `mmcif`, `cif`, `cif.gz`, `xml.gz`, `mmtf`, `mmtf.gz` - choose a file type for files downloaded from the PDB

prep_itasser_modeling(*itasser_installation*, *itlib_folder*, *runtype*, *create_in_dir=None*, *execute_from_dir=None*, *print_exec=False*, ***kwargs*)
Prepare to run I-TASSER homology modeling for the representative sequence.

Parameters

- **itasser_installation** (*str*) – Path to I-TASSER folder, i.e. `~/software/I-TASSER4.4`
- **itlib_folder** (*str*) – Path to ITLIB folder, i.e. `~/software/ITLIB`
- **runtype** – How you will be running I-TASSER - local, slurm, or torque
- **create_in_dir** (*str*) – Local directory where folders will be created
- **execute_from_dir** (*str*) – Optional path to execution directory - use this if you are copying the homology models to another location such as a supercomputer for running
- **all_genes** (*bool*) – If all genes should be prepped, or only those without any mapped structures
- **print_exec** (*bool*) – If the execution statement should be printed to run modelling

---

Todo:

- Document kwargs - extra options for I-TASSER, SLURM or Torque execution
- Allow modeling of any sequence in sequences attribute, select by ID or provide SeqProp?

---

protein_dir
str – Protein folder

protein_statistics
Get a dictionary of basic statistics describing this protein

representative_chain = None
str – Chain ID in the representative structure which best represents a sequence

representative_chain_seq_coverage = None
float – Percent identity of sequence coverage for the representative chain

representative_sequence = None
SeqProp – Sequence set to represent this protein

representative_structure = None
StructProp – Structure set to represent this protein, usually in monomeric form

root_dir
str – Path to where the folder named by this protein's ID will be created. Default is current working directory.

sequence_alignments = None
DictList – Pairwise or multiple sequence alignments stored as `Bio.Align.MultipleSeqAlignment` objects

sequence_dir
str – Directory where sequence related files are stored

sequence_mutation_summary(*alignment_ids=None*, *alignment_type=None*)
Summarize all mutations found in the sequence_alignments attribute.

Returns 2 dictionaries, single_counter and fingerprint_counter.

single_counter: Dictionary of `{point mutation: list of genes/strains}` Example:

```
{
    ('A', 24, 'V'): ['Strain1', 'Strain2', 'Strain4'],
    ('R', 33, 'T'): ['Strain2']
}
```

Here, we report which genes/strains have the single point mutation.

fingerprint_counter: Dictionary of `{mutation group: list of genes/strains}` Example:

```
{
    (('A', 24, 'V'), ('R', 33, 'T')): ['Strain2'],
    (('A', 24, 'V')): ['Strain1', 'Strain4']
}
```

Here, we report which genes/strains have the specific combinations (or "fingerprints") of point mutations

Parameters

- **alignment_ids** (*str, list*) – Specified alignment ID or IDs to use
- **alignment_type** (*str*) – Specified alignment type contained in the `annotation` field of an alignment object, `seqalign` or `structalign` are the current types.

Returns single_counter, fingerprint_counter

Return type dict, dict

sequences = None

DictList – Stored protein sequences which are related to this protein

set_representative_sequence(*force_rerun=False*)

Automatically consolidate loaded sequences (manual, UniProt, or KEGG) and set a single representative sequence.

Manually set representative sequences override all existing mappings. UniProt mappings override KEGG mappings except when KEGG mappings have PDBs associated with them and UniProt doesn't.

Parameters **force_rerun** (*bool*) – Set to True to recheck stored sequences

Returns Which sequence was set as representative

Return type *SeqProp*

set_representative_structure(*seq_outdir=None*, *struct_outdir=None*, *pdb_file_type=None*, *engine='needle'*, *always_use_homology=False*, *rez_cutoff=0.0*, *seq_ident_cutoff=0.5*, *allow_missing_on_termini=0.2*, *allow_mutants=True*, *allow_deletions=False*, *allow_insertions=False*, *allow_unresolved=True*, *clean=True*, *keep_chemicals=None*, *force_rerun=False*)

Set a representative structure from a structure in the structures attribute.

Each gene can have a combination of the following, which will be analyzed to set a representative structure.

- Homology model(s)
- Ranked PDBs

- BLASTed PDBs

If the `always_use_homology` flag is true, homology models are always set as representative when they exist. If there are multiple homology models, we rank by the percent sequence coverage.

Parameters

- **seq_outdir** (*str*) – Path to output directory of sequence alignment files, must be set if Protein directory was not created initially
- **struct_outdir** (*str*) – Path to output directory of structure files, must be set if Protein directory was not created initially
- **pdb_file_type** (*str*) – `pdb`, `pdb.gz`, `mmcif`, `cif`, `cif.gz`, `xml.gz`, `mmtf`, `mmtf.gz` - choose a file type for files downloaded from the PDB
- **engine** (*str*) – `biopython` or `needle` - which pairwise alignment program to use. `needle` is the standard EMBOSS tool to run pairwise alignments. `biopython` is Biopython's implementation of needle. Results can differ!
- **always_use_homology** (*bool*) – If homology models should always be set as the representative structure
- **rez_cutoff** (*float*) – Resolution cutoff, in Angstroms (only if experimental structure)
- **seq_ident_cutoff** (*float*) – Percent sequence identity cutoff, in decimal form
- **allow_missing_on_termini** (*float*) – Percentage of the total length of the reference sequence which will be ignored when checking for modifications. Example: if 0.1, and reference sequence is 100 AA, then only residues 5 to 95 will be checked for modifications.
- **allow_mutants** (*bool*) – If mutations should be allowed or checked for
- **allow_deletions** (*bool*) – If deletions should be allowed or checked for
- **allow_insertions** (*bool*) – If insertions should be allowed or checked for
- **allow_unresolved** (*bool*) – If unresolved residues should be allowed or checked for
- **clean** (*bool*) – If structure should be cleaned
- **keep_chemicals** (*str, list*) – Keep specified chemical names if structure is to be cleaned
- **force_rerun** (*bool*) – If sequence to structure alignment should be rerun

Returns Representative structure from the list of structures. This is a not a map to the original structure, it is copied and optionally cleaned from the original one.

Return type *StructProp*

structure_alignments = None
DictList – Pairwise or multiple structure alignments - currently a placeholder

structure_dir
str – Directory where structure related files are stored

structures = None
DictList – Stored protein structures which are related to this protein

5.7.3 StructProp

class ssbio.protein.structure.structprop.**StructProp**(*ident*, *description=None*, *chains=None*, *mapped_chains=None*, *is_experimental=False*, *structure_path=None*, *file_type=None*)

Generic class to represent information for a protein structure.

Provides access to the 3D coordinates using a Biopython Structure object through the method `parse_structure`. The main functionality added is the ability to set and load directly from any supported structure and metadata file. Additionally, the `mapped_chains` attribute allows for analysis of a subset of chains, which will map to a gene of interest. Also provides methods through `nglview` to view the structure in a Jupyter notebook.

id
str – Unique identifier for this protein structure

name
str – Optional name for this structure

description
str – Optional description for this structure

is_experimental
bool – Flag to note if this structure is an experimental model or a homology model

chains
DictList – A DictList of chains have their sequence stored in them, along with residue-specific annotations

mapped_chains
list – A simple list of chain IDs (strings) that will be used to subset analyses

file_type
str – Type of structure file

structure_file
str – Name of the structure file

add_chain_ids(*chains*)
Add chains by ID into the chains attribute

Parameters **chains** (*str, list*) – Chain ID or list of IDs

add_mapped_chain_ids(*mapped_chains*)
Add chains by ID into the mapped_chains attribute

Parameters **mapped_chains** (*str, list*) – Chain ID or list of IDs

add_residues_highlight_to_nglview(*view*, *structure_resnums*, *chain=None*, *res_color='red'*)
Add a residue number or numbers to an NGLWidget view object.

Parameters

- **view** (*NGLWidget*) – NGLWidget view object
- **structure_resnums** (*int, list*) – Residue number(s) to highlight, structure numbering

- **chain** (*str, list*) – Chain ID or IDs of which residues are a part of. If not provided, all chains in the mapped_chains attribute will be used. If that is also empty, and exception is raised.
- **res_color** (*str*) – Color to highlight residues with

add_scaled_residues_highlight_to_nglview(*view*, *structure_resnums*, *chain=None*, *color='red'*, *unique_colors=False*, *opacity_range=(0.5, 1)*, *scale_range=(0.7, 10)*)

Add a list of residue numbers (which may contain repeating residues) to a view, or add a dictionary of residue numbers to counts. Size and opacity of added residues are scaled by counts.

Parameters

- **view** (*NGLWidget*) – NGLWidget view object
- **structure_resnums** (*int, list, dict*) – Residue number(s) to highlight, or a dictionary of residue number to frequency count
- **chain** (*str, list*) – Chain ID or IDs of which residues are a part of. If not provided, all chains in the mapped_chains attribute will be used. If that is also empty, and exception is raised.
- **color** (*str*) – Color to highlight residues with
- **unique_colors** (*bool*) – If each mutation should be colored uniquely (will override color argument)
- **opacity_range** (*tuple*) – Min/max opacity values (residues that have higher frequency counts will be opaque)
- **scale_range** (*tuple*) – Min/max size values (residues that have higher frequency counts will be bigger)

clean_structure(*out_suffix='_clean'*, *outdir=None*, *force_rerun=False*, *remove_atom_alt=True*, *keep_atom_alt_id='A'*, *remove_atom_hydrogen=True*, *add_atom_occ=True*, *remove_res_hetero=True*, *keep_chemicals=None*, *keep_res_only=None*, *add_chain_id_if_empty='X'*, *keep_chains=None*)

Clean the structure file associated with this structure, and save it as a new file. Returns the file path.

Parameters

- **out_suffix** (*str*) – Suffix to append to original filename
- **outdir** (*str*) – Path to output directory
- **force_rerun** (*bool*) – If structure should be re-cleaned if a clean file exists already
- **remove_atom_alt** (*bool*) – Remove alternate positions
- **keep_atom_alt_id** (*str*) – If removing alternate positions, which alternate ID to keep
- **remove_atom_hydrogen** (*bool*) – Remove hydrogen atoms
- **add_atom_occ** (*bool*) – Add atom occupancy fields if not present
- **remove_res_hetero** (*bool*) – Remove all HETATMs
- **keep_chemicals** (*str, list*) – If removing HETATMs, keep specified chemical names

- **keep_res_only** (*str, list*) – Keep ONLY specified resnames, deletes everything else!
- **add_chain_id_if_empty** (*str*) – Add a chain ID if not present
- **keep_chains** (*str, list*) – Keep only these chains

Returns Path to cleaned PDB file

Return type str

find_disulfide_bridges(*threshold=3.0*)

Run Biopython's search_ss_bonds to find potential disulfide bridges for each chain and store in ChainProp.

get_dict_with_chain(*chain*, *only_keys=None*, *chain_keys=None*, *exclude_attributes=None*, *df_format=False*)

get_dict method which incorporates attributes found in a specific chain. Does not overwrite any attributes in the original StructProp.

Parameters

- **chain** –
- **only_keys** –
- **chain_keys** –
- **exclude_attributes** –
- **df_format** –

Returns attributes of StructProp + the chain specified

Return type dict

get_dssp_annotations(*outdir*, *force_rerun=False*)

Run DSSP on this structure and store the DSSP annotations in the corresponding ChainProp SeqRecords

Calculations are stored in the ChainProp's `letter_annotations` at the following keys:

- `SS-dssp`
- `RSA-dssp`
- `ASA-dssp`
- `PHI-dssp`
- `PSI-dssp`

Parameters

- **outdir** (*str*) – Path to where DSSP dataframe will be stored.
- **force_rerun** (*bool*) – If DSSP results should be recalculated

---

Todo:

- Also parse global properties, like total accessible surface area. Don't think Biopython parses those?

---

get_freesasa_annotations(*outdir*, *include_hetatms=False*, *force_rerun=False*)

Run `freesasa` on this structure and store the calculated properties in the corresponding ChainProps

get_residue_depths(*outdir*, *force_rerun=False*)
Run MSMS on this structure and store the residue depths/ca depths in the corresponding ChainProp SeqRecords

get_structure_seqs(*model*)
Gather chain sequences and store in their corresponding `ChainProp` objects in the `chains` attribute.

Parameters **model** (*Model*) – Biopython Model object of the structure you would like to parse

load_structure_path(*structure_path*, *file_type*)
Load a structure file and provide pointers to its location

Parameters

- **structure_path** (*str*) – Path to structure file
- **file_type** (*str*) – Type of structure file

parse_structure()
Read the 3D coordinates of a structure file and return it as a Biopython Structure object

Also create ChainProp objects in the chains attribute

Returns Biopython Structure object

Return type Structure

view_structure(*only_chains=None*, *opacity=1.0*, *recolor=False*, *gui=False*)
Use NGLviewer to display a structure in a Jupyter notebook

Parameters

- **only_chains** (*str, list*) – Chain ID or IDs to display
- **opacity** (*float*) – Opacity of the structure
- **recolor** (*bool*) – If structure should be cleaned and recolored to silver
- **gui** (*bool*) – If the NGLview GUI should show up

Returns NGLviewer object

5.7.4 SeqProp

class ssbio.protein.sequence.seqprop.**SeqProp**(*seq*, *id*, *name='<unknown name>'*, *description='<unknown description>'*, *sequence_path=None*, *metadata_path=None*, *feature_path=None*)
Generic class to represent information for a protein sequence.

Extends the Biopython SeqRecord class. The main functionality added is the ability to set and load directly from sequence, metadata, and feature files. Additionally, methods are provided to calculate and store sequence properties in the `annotations` and `letter_annotations` field of a SeqProp. These can then be accessed for a range of residue numbers.

id
str – Unique identifier for this protein sequence

seq
Seq – Protein sequence as a Biopython Seq object

name
str – Optional name for this sequence

description
str – Optional description for this sequence

bigg
str, list – BiGG IDs mapped to this sequence

kegg
str, list – KEGG IDs mapped to this sequence

refseq
str, list – RefSeq IDs mapped to this sequence

uniprot
str, list – UniProt IDs mapped to this sequence

gene_name
str, list – Gene names mapped to this sequence

pdbs
list – PDB IDs mapped to this sequence

go
str, list – GO terms mapped to this sequence

pfam
str, list – PFAMs mapped to this sequence

ec_number
str, list – EC numbers mapped to this sequence

sequence_file
str – FASTA file for this sequence

metadata_file
str – Metadata file (any format) for this sequence

feature_file
str – GFF file for this sequence

features
list – List of protein sequence features, which define regions of the protein

annotations
dict – Annotations of this protein sequence, which summarize global properties

letter_annotations
RestrictedDict – Residue-level annotations, which describe single residue properties

---

Todo:

- Properly inherit methods from the Object class…

---

blast_pdb(*seq_ident_cutoff=0*, *evalue=0.0001*, *display_link=False*, *outdir=None*, *force_rerun=False*)
BLAST this sequence to the PDB

equal_to(*seq_prop*)
Test if the sequence is equal to another SeqProp's sequence

Parameters **seq_prop** – SeqProp object

Returns If the sequences are the same

Return type bool

features
list – Get the features stored in memory or in the GFF file

get_aggregation_propensity(*email*, *password*, *cutoff_v=5*, *cutoff_n=5*, *run_amylmuts=False*, *outdir=None*)
Run the AMYLPRED2 web server to calculate the aggregation propensity of this protein sequence, which is the number of aggregation-prone segments on the unfolded protein sequence.

Stores statistics in the `annotations` attribute, under the key *aggprop-amylpred*.

See *ssbio.protein.sequence.properties.aggregation_propensity* for instructions and details.

get_biopython_pepstats()
Run Biopython's built in ProteinAnalysis module and store statistics in the `annotations` attribute.

get_dict(*only_attributes=None*, *exclude_attributes=None*, *df_format=False*)
Get a dictionary of this object's attributes. Optional format for storage in a Pandas DataFrame.

Parameters

- **only_attributes** (*str, list*) – Attributes that should be returned. If not provided, all are returned.
- **exclude_attributes** (*str, list*) – Attributes that should be excluded.
- **df_format** (*bool*) – If dictionary values should be formatted for a dataframe (everything possible is transformed into strings, int, or float - if something can't be transformed it is excluded)

Returns Dictionary of attributes

Return type dict

get_emboss_pepstats()
Run the EMBOSS pepstats program on the protein sequence.

Stores statistics in the `annotations` attribute. Saves a `.pepstats` file of the results where the sequence file is located.

get_kinetic_folding_rate(*secstruct*, *at_temp=None*)
Run the FOLD-RATE web server to calculate the kinetic folding rate given an amino acid sequence and its structural classficiation (alpha/beta/mixed)

Stores statistics in the `annotations` attribute, under the key *kinetic_folding_rate_<TEMP>-foldrate*.

See `ssbio.protein.sequence.properties.kinetic_folding_rate.get_foldrate()` for instructions and details.

get_residue_annotations(*start_resnum*, *end_resnum=None*)
Retrieve letter annotations for a residue or a range of residues

Parameters

- **start_resnum** (*int*) – Residue number
- **end_resnum** (*int*) – Optional residue number, specify if a range is desired

Returns Letter annotations for this residue or residues

Return type dict

get_thermostability(*at_temp*)
Run the thermostability calculator using either the Dill or Oobatake methods.

Stores calculated (dG, Keq) tuple in the `annotations` attribute, under the key *thermostability_<TEMP>-<METHOD_USED>*.

See `ssbio.protein.sequence.properties.thermostability.get_dG_at_T()` for instructions and details.

num_pdbs
int – Report the number of PDB IDs stored in the `pdbs` attribute

seq
Seq – Dynamically loaded Seq object from the sequence file

seq_len
int – Get the sequence length

seq_str
str – Get the sequence formatted as a string

write_fasta_file(*outfile*, *force_rerun=False*)
Write a FASTA file for the protein sequence, `seq` will now load directly from this file.

Parameters

- **outfile** (*str*) – Path to new FASTA file to be written to
- **force_rerun** (*bool*) – If an existing file should be overwritten

write_gff_file(*outfile*, *force_rerun=False*)
Write a GFF file for the protein features, `features` will now load directly from this file.

Parameters

- **outfile** (*str*) – Path to new FASTA file to be written to
- **force_rerun** (*bool*) – If an existing file should be overwritten

5.7.5 PDBProp

class ssbio.databases.pdb.**PDBProp**(*ident*, *description=None*, *chains=None*, *mapped_chains=None*, *structure_path=None*, *file_type=None*)
Class to parse through PDB properties

ssbio.databases.pdb.**best_structures**(*uniprot_id*, *outname=None*, *outdir=None*, *seq_ident_cutoff=0.0*, *force_rerun=False*)
Use the PDBe REST service to query for the best PDB structures for a UniProt ID.

More information found here: https://www.ebi.ac.uk/pdbe/api/doc/sifts.html Link used to retrieve results: https://www.ebi.ac.uk/pdbe/api/mappings/best_structures/:accession The list of PDB structures mapping to a UniProt accession sorted by coverage of the protein and, if the same, resolution.

Here is the ranking algorithm described by the PDB paper: https://nar.oxfordjournals.org/content/44/D1/D385.full

"Finally, a single quality indicator is also calculated for each entry by taking the harmonic average of all the percentile scores representing model and model-data-fit quality measures and then subtracting 10 times the numerical value of the resolution (in Angstrom) of the entry to ensure that resolution plays a role in characterising the quality of a structure. This single empirical 'quality measure' value is used by the PDBe query system to sort results and identify the 'best' structure in a given context. At present, entries determined by methods other

than X-ray crystallography do not have similar data quality information available and are not considered as 'best structures'."

Parameters

- **uniprot_id** (*str*) – UniProt Accession ID
- **outname** (*str*) – Basename of the output file of JSON results
- **outdir** (*str*) – Path to output directory of JSON results
- **seq_ident_cutoff** (*float*) – Cutoff results based on percent coverage (in decimal form)
- **force_rerun** (*bool*) – Obtain best structures mapping ignoring previously downloaded results

Returns

Rank-ordered list of dictionaries representing chain-specific PDB entries. Keys are:

- pdb_id: the PDB ID which maps to the UniProt ID
- chain_id: the specific chain of the PDB which maps to the UniProt ID
- coverage: the percent coverage of the entire UniProt sequence
- resolution: the resolution of the structure
- start: the structure residue number which maps to the start of the mapped sequence
- end: the structure residue number which maps to the end of the mapped sequence
- unp_start: the sequence residue number which maps to the structure start
- unp_end: the sequence residue number which maps to the structure end
- experimental_method: type of experiment used to determine structure
- tax_id: taxonomic ID of the protein's original organism

Return type list

ssbio.databases.pdb.**blast_pdb**(*seq*, *outfile=''*, *outdir=''*, *evalue=0.0001*, *seq_ident_cutoff=0.0*, *link=False*, *force_rerun=False*)

Returns a list of BLAST hits of a sequence to available structures in the PDB.

Parameters

- **seq** (*str*) – Your sequence, in string format
- **outfile** (*str*) – Name of output file
- **outdir** (*str, optional*) – Path to output directory. Default is the current directory.
- **evalue** (*float, optional*) – Cutoff for the E-value - filters for significant hits. 0.001 is liberal, 0.0001 is stringent (default).
- **seq_ident_cutoff** (*float, optional*) – Cutoff results based on percent coverage (in decimal form)
- **link** (*bool, optional*) – Set to True if a link to the HTML results should be displayed
- **force_rerun** (*bool, optional*) – If existing BLAST results should not be used, set to True. Default is False

Returns Rank ordered list of BLAST hits in dictionaries.

Return type list

ssbio.databases.pdb.**blast_pdb_df**(*blast_results*)
Make a dataframe of BLAST results

ssbio.databases.pdb.**download_biological_assemblies**(*pdb_id*, *outdir*)
Downloads biological assembly file from: *ftp://ftp.wwpdb.org/pub/pdb/data/biounit/coordinates/divided/*

Parameters **outdir** (*str*) – Output directory of the decompressed assembly

ssbio.databases.pdb.**download_sifts_xml**(*pdb_id*, *outdir=''*, *outfile=''*)
Download the SIFTS file for a PDB ID.

Parameters

- **pdb_id** –
- **outdir** –
- **outfile** –

Returns:

ssbio.databases.pdb.**download_structure**(*pdb_id*, *file_type*, *outdir=''*, *outfile=''*, *only_header=False*, *force_rerun=False*)
Download a structure from the RCSB PDB by ID. Specify the file type desired.

Parameters

- **pdb_id** – PDB ID
- **file_type** – pdb, pdb.gz, mmcif, cif, cif.gz, xml.gz, mmtf, mmtf.gz
- **outdir** – Optional output directory
- **outfile** – Optional output name
- **only_header** – If only the header file should be downloaded
- **force_rerun** – If the file should be downloaded again even if it exists

Returns Path to outfile

Return type str

ssbio.databases.pdb.**get_release_date**(*pdb_id*)
Quick way to get the release date of a PDB ID using the table of results from the REST service

Returns None if the release date is not available.

Returns Organism of a PDB ID

Return type str

ssbio.databases.pdb.**get_resolution**(*pdb_id*)
Quick way to get the resolution of a PDB ID using the table of results from the REST service

Returns infinity if the resolution is not available.

Returns resolution of a PDB ID in Angstroms

Return type float

Todo:

- Unit test

---

ssbio.databases.pdb.**map_uniprot_resnum_to_pdb**(*uniprot_resnum*, *chain_id*, *sifts_file*)
Map a UniProt residue number to its corresponding PDB residue number.

This function requires that the SIFTS file be downloaded, and also a chain ID (as different chains may have different mappings).

Parameters

- **uniprot_resnum** (*int*) – integer of the residue number you'd like to map
- **chain_id** (*str*) – string of the PDB chain to map to
- **sifts_file** (*str*) – Path to the SIFTS XML file

Returns

tuple containing:

mapped_resnum (int): Mapped residue number is_observed (bool): Indicates if the 3D structure actually shows the residue

Return type (tuple)

ssbio.databases.pdb.**parse_mmcif_header**(*infile*)
Parse a couple important fields from the mmCIF file format with some manual curation of ligands.

If you want full access to the mmCIF file just use the MMCIF2Dict class in Biopython.

Parameters **infile** – Path to mmCIF file

Returns Dictionary of parsed header

Return type dict

ssbio.databases.pdb.**parse_pdb_header**(*infile*)
Parse a couple important fields from the mmCIF file format with some manual curation of ligands.

If you want full access to the mmCIF file just use the MMCIF2Dict class in Biopython.

Parameters **infile** – Path to mmCIF file

Returns Dictionary of parsed header

Return type dict

5.7.6 UniProtProp

class ssbio.databases.uniprot.**UniProtProp**(*seq*, *id*, *name='<unknown name>'*, *description='<unknown description>'*, *fasta_path=None*, *xml_path=None*, *gff_path=None*)
Generic class to store information on a UniProt entry, extended from a SeqProp object.

The main utilities of this class are to:

1. Download and/or parse UniProt text or xml files
2. Store extra parsed information in attributes

uniprot
str – Main UniProt accession code

alt_uniprots
list – Alternate accession codes that point to the main one

file_type
str – Metadata file type

reviewed
bool – If this entry is a "reviewed" entry. If None, then status is unknown.

ec_number
str – EC number

pfam
list – PFAM IDs

entry_version
str – Date of last update of the UniProt entry

seq_version
str – Date of last update of the UniProt sequence

download_metadata_file(*outdir*, *force_rerun=False*)
Download and load the UniProt XML file

download_seq_file(*outdir*, *force_rerun=False*)
Download and load the UniProt FASTA file

features
list – Get the features from the feature file, metadata file, or in memory

ranking_score()
Provide a score for this UniProt ID based on reviewed (True=1, False=0) + number of PDBs

Returns Scoring for this ID

Return type int

seq
Seq – Get the Seq object from the sequence file, metadata file, or in memory

ssbio.databases.uniprot.**blast_uniprot**(*seq_str*, *seq_ident=1*, *evalue=0.0001*, *reviewed_only=True*, *organism=None*)
BLAST the UniProt db to find what IDs match the sequence input

Parameters

- **seq_str** – Sequence string
- **seq_ident** – Percent identity to match
- **evalue** – E-value of BLAST hit

Returns:

ssbio.databases.uniprot.**download_uniprot_file**(*uniprot_id*, *filetype*, *outdir=''*, *force_rerun=False*)
Download a UniProt file for a UniProt ID/ACC

Parameters

- **uniprot_id** – Valid UniProt ID
- **filetype** – txt, fasta, xml, rdf, or gff
- **outdir** – Directory to download the file

Returns Absolute path to file

Return type str

ssbio.databases.uniprot.**get_fasta**(*uniprot_id*)
Get the protein sequence for a UniProt ID as a string.

Parameters **uniprot_id** – Valid UniProt ID

Returns String of the protein (amino acid) sequence

Return type str

ssbio.databases.uniprot.**is_valid_uniprot_id**(*instring*)
Check if a string is a valid UniProt ID.

See regex from: http://www.uniprot.org/help/accession_numbers

Parameters **instring** – any string identifier

Returns: True if the string is a valid UniProt ID

ssbio.databases.uniprot.**old_parse_uniprot_txt_file**(*infile*)
From: boscoh/uniprot github Parses the text of metadata retrieved from uniprot.org.

Only a few fields have been parsed, but this provides a template for the other fields.

A single description is generated from joining alternative descriptions.

Returns a dictionary with the main UNIPROT ACC as keys.

ssbio.databases.uniprot.**parse_uniprot_txt_file**(*infile*)
Parse a raw UniProt metadata file and return a dictionary.

Parameters **infile** – Path to metadata file

Returns Metadata dictionary

Return type dict

ssbio.databases.uniprot.**parse_uniprot_xml_metadata**(*sr*)
Load relevant attributes and dbxrefs from a parsed UniProt XML file in a SeqRecord.

Returns All parsed information

Return type dict

ssbio.databases.uniprot.**uniprot_ec**(*uniprot_id*)
Retrieve the EC number annotation for a UniProt ID.

Parameters **uniprot_id** – Valid UniProt ID

Returns:

ssbio.databases.uniprot.**uniprot_reviewed_checker**(*uniprot_id*)
Check if a single UniProt ID is reviewed or not.

Parameters **uniprot_id** –

Returns If the entry is reviewed

Return type bool

ssbio.databases.uniprot.**uniprot_reviewed_checker_batch**(*uniprot_ids*)
Batch check if uniprot IDs are reviewed or not

Parameters **uniprot_ids** – UniProt ID or list of UniProt IDs

Returns Boolean}

Return type A dictionary of {UniProtID

ssbio.databases.uniprot.**uniprot_sites**(*uniprot_id*)

Retrieve a list of UniProt sites parsed from the feature file

Sites are defined here: http://www.uniprot.org/help/site and here: http://www.uniprot.org/help/function_section

Parameters **uniprot_id** – Valid UniProt ID

Returns:

CHAPTER 6

Indices and tables

- genindex
- modindex

Python Module Index

s

Index

E

F

G

I

K

L

M

N

O

P

R

S

U

V

W