PyHDX Documentation

Release 0.2.1

Jochem Smit

Sep 30, 2020

CONTENTS:

CHAPTER

ONE

PYHDX

PyHDX is a software package that extract protection factors from HDX-MS data.

Currently the project functional but in beta. Please refer to docs/installation.rst for installation instructions.

1.1 Web Application

A beta version of the web application is available for testing: http://pyhdx.jhsmit.org/main

A test file can be downloaded from here.

Two other web applications are available. To upload fitting results from the main application and vizualize: http://pyhdx.jhsmit.org/single To upload multiple fitting result datasets and compare and vizualize: http://pyhdx.jhsmit.org/diff

CHAPTER

TWO

INSTALLATION

2.1 Stable release

(Currently no stable release available. This section will updated soon)

To install PyHDX, run this command in your terminal:

```
$ pip install pyhdx
```

This is the preferred method to install PyHDX, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for PyHDX can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/Jhsmit/pyhdx
```

Or download the tarball:

```
$ curl  -OL https://github.com/Jhsmit/pyhdx/tarball/master
```

pyHDX can then be installed with `conda` (requires `conda build`):

```
$ conda develop pyhdx
```

or `pip`:

```
$ pip install pyhdx
```

To launch the web application:

```
$ panel serve panel/main.py
```

2.3 Dependencies

The requirements for PyHDX are listed in requirements.txt and can be installed from either pip or conda, with the exception of expfact. This is a GPL package and at the moments it is recommended to manually install this by downloading the *constants.py* and *kint.py* files from *expfact/python* directory on the GitHub repository and placing them in pyhdx/expfact

CHAPTER

THREE

FITTING

The main feature of pyHDX is the fitting of rate equations describing deuterium uptake to a kinetic series of measured peptides each covering a section of residues with a corresponding amount of deuterium uptake per peptide-timepoint.

3.1 Overfitting

Overfitting occurs when more parameters are added to the model but the supplied data has insufficient independent datapoints to be able to accurately and uniquely determine the value of these parameters. Typical signs of overfitting are large variations along residues in the obtained rates, such as for residue 43 in Figure Fig. 3.1.

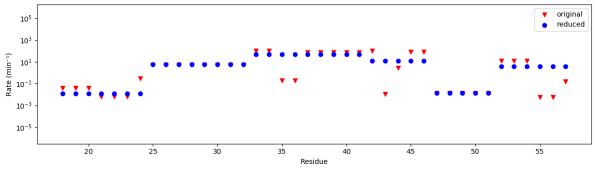


Fig. 3.1: XX not really a great example of overfitting

To determine if overfitting has occurs, the number of fitting parameters should be varied while checking the effect of adding and removing fit parameters againts goodness-of-fit parameters. This is a laborious and time consuming process and further streamlining and automating this process is planned to be part of a future release.

In the current implementation, fitting accuracy and residue resolution is sacrificed in order to make sure overfitting is unlikely. Block size is increased and the number of exchange rate time constants is limited to 2. The downside of this approach is that the fits can be poor in the case of residues exchanging with more than two distinct rate constants per block, or that features consisting of only several residues can be missed. Examples of how to customize the defintion of fitting blocks can be found in the examples section.

3.2 Non-identifyability

Consider a block of 5 amino acids which all exchange deuterium with very distinct exchange rates and a set of measurements where the timepoints sufficiently cover these exchange rates. In this scenario, although its possible to extract all 5 kinetic rates by fitting the uptake curve, it is impossible to assign these kinetics rates to individual amino acids. This is referred to as the non-identifyability issue (XX REF) and this can only be overcome by increasing the number of peptides such that each amino acid occurs in a unique set of peptides.

CHAPTER

FOUR

EXAMPLES

4.1 pyHDX basics

[2]:
```
from pyhdx import PeptideMasterTable, read_dynamx
from pathlib import Path
```

We can use the `read_dynamx` function to read the file. This function returns a `numpy` structured array where each entry corresponds to one peptide, in this example 567 peptides.

[12]:
```
fpath = Path() / '..' / '..' / 'tests' / 'test_data' / 'ecSecB_apo.csv'
data = read_dynamx(fpath)
len(data)
```

[12]:
```
567
```

This array is loaded into the `PeptideMasterTable` class, which is the main data entry class. By specifying `drop_first` the number of n-terminal residues to remove can be changed and with `ignore_prolines` prolines residues, which do not have exchanging amide hydrogens, can be ignored.

[16]:
```
master_table = PeptideMasterTable(data, drop_first=1, ignore_prolines=True)
```

This master table allows us to control how the deuterium uptake content is determined. The method `set_control` can be used to choose which set of peptides is used as the fully deuterated (FD) control. This adds a new field called 'uptake' which is the normalized (to 100%) deuterium uptake of each peptide.

[17]:
```
master_table.set_control(('Full deuteration control', 0.167))
master_table.data['uptake'][:50]
```

[17]:
```
array([ 0.      ,  0.      ,  5.0734  ,  2.486444,  2.857141,  3.145738,
        3.785886,  4.08295 ,  4.790625,  0.      ,  0.      ,  3.642506,
        1.651437,  1.860919,  2.107151,  2.698036,  2.874801,  3.449561,
        0.      ,  0.      ,  5.264543,  1.839924,  2.508343,  2.969332,
        3.399092,  3.485568,  4.318144,  0.      ,  0.      ,  6.3179  ,
        2.532099,  3.306167,  3.996718,  4.38941 ,  4.379495,  5.283969,
        0.      ,  0.      ,  6.812215,  3.11985 ,  3.874881,  4.342807,
        4.854057,  4.835639,  5.780219,  0.      ,  0.      , 10.8151  ,
        5.432395,  6.1318  ])
```

Next we'll split the data and group them by their different states. This returns a dictionary where the values are all peptides for a given state. The peptides for each state are grouped by their exposure time, forming a `KineticSeries` object

[19]:
```
states = master_table.groupby_state()
for key, value in states.items():
    print(key, value)
```

```
Full deuteration control <pyhdx.models.KineticsSeries object at 0x0000014774911FC8>
SecB WT apo <pyhdx.models.KineticsSeries object at 0x000001477428F908>
```

[6]:
```
series = states['SecB WT apo']
type(series), len(series), series.timepoints
```

```
dict_keys(['Full deuteration control', 'SecB WT apo'])
```

Iterating over a `KineticSeries` object returns a set of `PeptideMeasurements` each with their own attributes describing the topology of the coverage. When all `PeptideMeasurements` in the series have identical coverage, the series is said to be `uniform`, which can be checked by the `uniform` property. Series can be made `uniform` by default, removing peptides which are not found in all timepoints. `KineticsSeries` are required to be uniform before fitting them.

[ ]:
```
print(series.uniform)
series.make_uniform()  # This series already is uniform
```

4.2 Under construction

[1]:
```
# Topics:
# X matrix
# removing prolines and n terminal resiudes
# r number vector
# weighted averaged scores
# splitting series
```

4.3 Fitting

[22]:
```
%matplotlib qt
import matplotlib.pyplot as plt
from pyhdx import PeptideMasterTable, read_dynamx, KineticsFitting
from pathlib import Path
import numpy as np
```

[21]:
```
import pyhdx
print(pyhdx.__file__)
pyhdx.__git_sha__
```

```
C:\Users\jhsmi\pp\PyHDX\pyhdx\__init__.py
```

[21]:
```
'2f1502d'
```

We load the sample SecB dataset, apply the control, and split the dataset into `KineticSeries`.

[2]:
```
fpath = Path() / '..' / '..' / 'tests' / 'test_data' / 'ecSecB_apo.csv'
data = read_dynamx(fpath)
master_table = PeptideMasterTable(data, drop_first=1, ignore_prolines=True)
```

(continues on next page)

(continued from previous page)

```
master_table.set_control(('Full deuteration control', 0.167))
states = master_table.groupby_state()
series = states['SecB WT apo']
series.make_uniform()
```

From this `KineticsSeries` object we can make a `KineticsFitting` object. The `bounds` parameter defines the upper and lower limit of the kinetic rates which are fitted. Temperature (in Kelvin) and pH of the D-labelling step are used to calculate the intrinsic D-exchange rate.

[3]:
```
kf = KineticsFitting(series, bounds=(1e-2, 300), temperature=303.15, pH=8.)
```

We can now start the first step of fitting, by weighted averaging. The `RuntimeWarning` messages are normal and can be ignored.

[4]:
```
result_wt_avg = kf.weighted_avg_fit()
```

```
C:\Users\jhsmi\Miniconda3\envs\py37_panel_dev\lib\site-packages\symfit\core\
↪objectives.py:321: RuntimeWarning: overflow encountered in square
  (dep_var_value - dep_data) ** 2 / sigma ** 2
<string>:2: RuntimeWarning: overflow encountered in exp
C:\Users\jhsmi\Miniconda3\envs\py37_panel_dev\lib\site-packages\scipy\optimize\
↪optimize.py:2116: RuntimeWarning: invalid value encountered in double_scalars
  tmp2 = (x - v) * (fx - fw)
C:\Users\jhsmi\Miniconda3\envs\py37_panel_dev\lib\site-packages\scipy\optimize\
↪minpack.py:175: RuntimeWarning: The iteration is not making good progress, as␣
↪measured by the
  improvement from the last ten iterations.
  warnings.warn(msg, RuntimeWarning)
<string>:2: RuntimeWarning: overflow encountered in exp
<string>:2: RuntimeWarning: overflow encountered in exp
<string>:2: RuntimeWarning: invalid value encountered in subtract
```

The return value is a `KineticsFitResult` object. This object has a list of models, intervals in withing the protein sequence to which these models apply, and their corresponding `symfit` fit result with parameter values. The effective exchange rate can be extracted, as well as other fit parameters, from this object:

[25]:
```
output = result_wt_avg.output
output.dtype.names
```

[25]:
```
('r_number', 'rate', 'k1', 'k2', 'r')
```

[26]:
```
fig, ax = plt.subplots()
ax.set_yscale('log')
ax.scatter(output['r_number'], output['rate'])
ax.set_xlabel('Residue number')
ax.set_ylabel('Rate (min$^{-1}$)')
None
```

We can now use the weighted averaging fitted result as initial guesses for the global fitting step. This returns a `TFFitResult` object, which has only one interval and model.

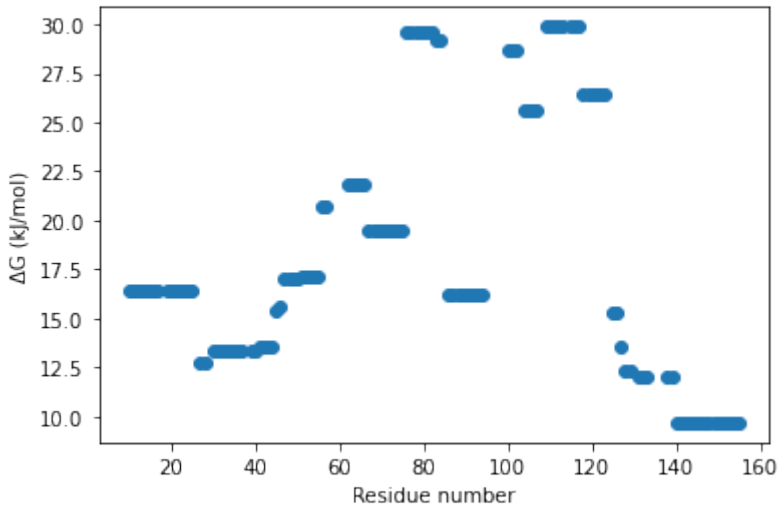
[7]:
```
result_global = kf.global_fit(output)
```

We can obtain protection factors and ΔG values from the result. The protection factors are in log (base 10) format.

[10]:
```
tf_output = result_global.output
print(tf_output.dtype.names)
deltaG = 8.3*303.15*(tf_output['log_P'] / np.log10(np.e))
```

```
('r_number', 'log_P_full', 'log_P')
```

[13]:
```
fig, ax = plt.subplots()
#ax.set_yscale('log')
ax.scatter(tf_output['r_number'], deltaG*1e-3)
ax.set_xlabel('Residue number')
ax.set_ylabel('ΔG (kJ/mol)')
None
```



CHAPTER

FIVE

MODULE DOCUMENTATION

This page contains the full API docs of PyHDX

5.1 Models

class pyhdx.models.**Coverage**(*data*)

Object describing layout and coverage of peptides and generating the corresponding matrices. Peptides should all belong to the same state and have the same exposure time.

Parameters

data [~class:~*numpy.ndarray*] Numpy structured array with input peptides

Attributes

start [int]

Index of residue first appearing in the peptides (first residue is 1)

end [int] Index of last residue appearing in the peptides (inclusive)

r_number [ndarray] Array of residue numbers which are covered by the peptides, excluding prolines if they are set to be ignored.

prot_len [int] Total number of residues the peptides covering, excluding prolines if they are set to be ignored.

X [ndarray] N x M matrix where N is the number of peptides and M equal to *prot_len*. Values are 1/(ex_residues) where there is coverage, so that rows sum to 1

Methods

calc_kint(temperature, pH, c_term)	Calculates the intrinsic rate of the sequence.
get_sections([gap_size])	get the intervals of sections of coverage intervals are inclusive, exclusive
split([gap_size])	Splits the dataset into independent parts which have no overlapping peptides between them.

property X_norm

ndarray: *X* coefficient matrix normalized column wise.

property X_red

ndarray: Reduced NxM coefficient matrix, with N the number of peptides and M the number of blocks.

Elements are equal to the length of the block.

property X_red_norm
ndarray: *X_red* blocks coefficient matrix normalized column wise.

property block_coverage
ndarray: Boolean array with True values where blocks have coverage.

property block_length
ndarary: Lengths of unique blocks of residues in the peptides map, along the *r_number* axis

calc_kint (*temperature*, *pH*, *c_term*)
Calculates the intrinsic rate of the sequence. Values of no coverage or prolines are assigned a value of -1 The rates run are for the first residue (1) up to the last residue that is covered by peptides

When the previous residue is unknown the current residue is also assigned a value of -1.g

Parameters

temperature: [float] Temperature of the labelling reaction (Kelvin)

pH [float] pH of the labelling reaction

c_term [int] index of the last residue in the sequence (first residue is 1)

Returns

k_int [~class:*~numpy.ndarray*] Array of intrisic exchange rates

get_sections (*gap_size=- 1*)
get the intervals of sections of coverage intervals are inclusive, exclusive

gap_size: int

Gaps of this size between adjacent peptides is not considered to overlap. A value of -1 means that peptides with exactly zero overlap are separated. With gap_size=0 peptides with exactly zero overlap are not separated, and larger values tolerate larger gap sizes.

property has_coverage
ndarray: Boolean array indicating if the residues along r_number have coverage

property sequence
str: String of the full protein sequence. One letter coding where X marks regions of no coverage

property sequence_r_number
~class:*numpy.ndarray*: Array of r numbers corresponding to residues in sequence

split (*gap_size=- 1*)
Splits the dataset into independent parts which have no overlapping peptides between them. To determine overlap, the modified 'start' and 'end' fields are used which take into account N-terminal non-exchanging residues and prolines.

Returns

output: **dict** Dictionary where keys are {start}_{end} (inclusive, exclusive) of the corresponding sections, values are of the type of the current instance.

gap_size: **int** Gaps of this size between adjacent peptides is not considered to overlap. A value of -1 means that peptides with exactly zero overlap are separated. With gap_size=0 peptides with exactly zero overlap are not separated, and larger values tolerate larger gap sizes.

class pyhdx.models.**KineticsSeries** (*data*, *make_uniform=True*, ***metadata*)
A series of PeptideMeasurements which correspond to the same state but with different exposures.

Parameters

data [ndarray or list] Numpy structured array with peptide entries corresponding to a single state, or list of PeptideMeasurements

make_uniform [bool] If *True* the returned KineticSeries is made uniform

Attributes

state [str] State of the kinetic series

timepoints [ndarray] Array with exposure times (sorted)

Methods

make_uniform([in_place])	Removes entries from time points, ensuring that all time points have equal coverage
set_control(control_100[, control_zero, …])	Apply a control dataset to the underlying PeptideMeasurements of this object.
split([gap_size])	Splits the dataset into independent parts which have no overlapping peptides between them

property full_data
returns the full dataset of all timepoints

property k_int
this might need to move somewhere else, eg coverage object although if series are not uniform, k_int has to be on the main object as it wont have global coverages

make_uniform(*in_place=True*)
Removes entries from time points, ensuring that all time points have equal coverage

property scores_stack
uptake scores to fit in a 2d stack

set_control(*control_100*, *control_zero=None*, *remove_nan=True*)
Apply a control dataset to the underlying PeptideMeasurements of this object. A *scores* attribute is added to the PeptideMeasurement by normalizing its uptake value with respect to the control uptake value to 100%. Entires which are in the measurement and not in the control or vice versa are deleted. Optionally, control_zero can be specified which is a datasets whose uptake value will be set to zero.

Parameters

control_100 [ndarray] Numpy structured array with control peptides to use for normalization to 100%

control_zero [ndarray] Numpy structured array with control peptides to use for normalization to 0%

remove_nan [Bool] If *True*, *NaN* entries are removed from the controls

Returns

——-

split(*gap_size=- 1*)
Splits the dataset into independent parts which have no overlapping peptides between them

Returns

output [dict]

Output dictionary with individual kinetic series. Keys are '{start}_{stop}', (including, excluding) values are `KineticSeries` objects.

gap_size: `int` Gaps of this size between adjacent peptides is not considered to overlap. A value of -1 means that peptides with exactly zero overlap are separated. With gap_size=0 peptides with exactly zero overlap are not separated, and larger values tolerate larger gap sizes.

property uniform

Returns `True` if for all time point coverages are equal

property uptake_corrected

matrix shape N_t, N_p

class pyhdx.models.**PeptideMasterTable**(*data*, *drop_first=1*, *ignore_prolines=True*, *sort=True*, *remove_nan=True*)

Main peptide input object. The input numpy structured array *data* must have the following entires for each peptide:

start: Residue number of the first amino acid in the peptide end: Residue number of the last amino acid in the peptide (inclusive) sequence: Amino acid sequence of the peptide (one letter code) exposure: Typically the time the sample was exposed to a deuterated solution. This can correspond to other times if

the kinetics of the experiment are set up differently

state: String describing to which state (experimental conditions) the peptide belongs uptake: Number of deuteriums the peptide has taken up

The following fields are added to the *data* array upon initialization:

_start: Unmodified copy of initial start field _end: Unmodified copy of initial end field _sequence: Unmodified copy of initial sequence ex_residues: Number of residues that undergo deuterium exchange. This number is calculated using the *drop_first* and

ignore_prolines parameters

N-terminal residues which are removed because they are either within *drop_first* or they are N-terminal prolines are marked with 'x' in the *sequence* field. Prolines which are removed because they are in the middle of a peptide are marked with a lower case 'p' in the sequence field.

The field *scores* is used in calculating exchange rates and can be set by either the *set_backexchange* or *set_control* methods.

Parameters

data [~:class:*np.ndarray*] Numpy recarray with peptide entries.

drop_first [`int`] Number of N-terminal amino acids to ignore. Default is 1.

ignore_prolines: :obj:`bool` Boolean to toggle ignoring of proline residues. When True these residues are treated as if they're not present in the protein.

sort: :obj:`bool` Set to `True` to sort the input. Sort order is 'start', 'end', 'sequence', 'exposure', 'state'.

remove_nan: :obj`bool` Set to `True` to remove NaN entries in uptake

Attributes

exposures

~classs:np.ndarray Array with unique exposures

states

~classs:np.ndarray Array with unique states

Methods

get_data(state, exposure)	Get all peptides matching *state* and *exposure*.
groupby_state([make_uniform])	Groups measurements in the dataset by state and returns them in a dictionary as a `KineticSeries`.
isin_by_idx(array, test_array)	Checks if entries in *array* are in *test_array*, by *start* and *end* field values.
set_backexchange(back_exchange)	Sets the normalized percentage of uptake through a fixed backexchange value for all peptides.
set_control(control_100[, control_0])	Apply a control dataset to this object.

return_by_name	

property exposures

~classs:np.ndarray Array with unique exposures

get_data(*state*, *exposure*)

Get all peptides matching *state* and *exposure*.

Parameters

state [`str`] Measurement state

exposure [`float`] Measurement exposure time

Returns

output_data [`ndarray`] Numpy structured array with selected peptides

groupby_state(*make_uniform=True*)

Groups measurements in the dataset by state and returns them in a dictionary as a `KineticSeries`.

Parameters

make_uniform [`bool`] If *True* the returned `KineticSeries` is made uniform

Returns

out [`dict`] Dictionary where keys are state names and values are `KineticSeries`

static isin_by_idx(*array*, *test_array*)

Checks if entries in *array* are in *test_array*, by *start* and *end* field values.

Parameters

array [`ndarray`] Numpy input structured array

test_array [`ndarray`] Numpy structured array to test againts

Returns

isin: ndarray, bool Boolean array of the same shape as *array* where entries are *True* if they are in *test_array*

set_backexchange(*back_exchange*)

Sets the normalized percentage of uptake through a fixed backexchange value for all peptides.

Parameters

back_exchange [**`obj`:float:**] Percentage of back exchange

set_control(*control_100*, *control_0=None*)

Apply a control dataset to this object. A *scores* attribute is added to the object by normalizing its uptake value with respect to the control uptake value to 100%. Entries which are in the measurement and not in the control or vice versa are deleted. Optionally, `control_zero` can be specified which is a dataset whose uptake value will be used to zero the uptake.

#todo insert math

Parameters

control_100 [tuple] tuple with (*state*, *exposure*) for peptides to use for normalization to 100% Numpy structured array with control peptides to use for normalization to 100%

control_0 [tuple, optional] tuple with (*state*, *exposure*) for peptides to use for zeroing uptake values to 100%

property states

~classs:np.ndarray Array with unique states

class pyhdx.models.**PeptideMeasurements**(*data*)

Class with subset of peptides corresponding to only one state and exposure

Parameters

data [:class`~numpy.ndarray`] Numpy structured array with input data

scores [`ndarray`] Array with D/H uptake scores, typically in percentages or absolute uptake numbers.

Attributes

start [`int`] First peptide starts at this residue number (starting from 1)

stop [`int`] Last peptide ends at this residue number (incusive)

prot_len [`int`] Total number of residues in this set of peptides, not taking regions of no coverage into account.

exposure [`float`] Exposure time of this set of peptides (minutes)

state [`string`] State describing the experiment

bigX

X

properties:

big_x_norm

x_norm

scores nnls

scores lsq

Methods

calc_scores(residue_scores)	Calculates uptake scores per peptide given an array of individual residue scores
scores_nnls()	DEPRECATED
scores_nnls_tikonov(reg)	DEPRECATED
set_control(control_100[, control_0, remove_nan])	Apply a control dataset to this object.

calc_scores(*residue_scores*)
Calculates uptake scores per peptide given an array of individual residue scores

Parameters

residue_scores [ndarray] Array of scores per residue of length *prot_len*

Returns

scores [:class`~numpy.ndarray`] Array of scores per peptide

property scores_lstsq
DEPRECATED

scores_nnls()
DEPRECATED

scores_nnls_tikonov(*reg*)
DEPRECATED

set_control(*control_100*, *control_0=None*, *remove_nan=True*)
Apply a control dataset to this object. A *scores* attribute is added to the object by normalizing its uptake value with respect to the control uptake value to 100%. Entries which are in the measurement and not in the control or vice versa are deleted. Optionally, `control_zero` can be specified which is a datasets whose uptake value will be set to zero.

Parameters

control_100 [ndarray] Numpy structured array with control peptides to use for normalization to 100%

control_0 [ndarray] Numpy structured array with control peptides to use for normalization to 0%

remove_nan [Bool] If *True*, *NaN* entries are removed from the controls

class pyhdx.models.**TFCoverage**(*data*)
Object describing layout and coverage of peptides and generating the corresponding matrices. Peptides should all belong to the same state and have the same exposure time.

Parameters

data [~class:~*numpy.ndarray*] Numpy structured array with input peptides

Attributes

start [int]

Index of residue first appearing in the peptides (first residue is 1)

end [int] Index of last residue appearing in the peptides (inclusive)

r_number [ndarray] Array of residue numbers which are covered by the peptides, excluding prolines if they are set to be ignored.

prot_len [int] Total number of residues the peptides covering, excluding prolines if they are set to be ignored.

X [ndarray] N x M matrix where N is the number of peptides and M equal to *prot_len*. Values are 1/(ex_residues) where there is coverage, so that rows sum to 1

Methods

calc_kint(temperature, pH, c_term)	Calculates the intrinsic rate of the sequence.
get_sections([gap_size])	get the intervals of sections of coverage intervals are inclusive, exclusive
split([gap_size])	Splits the dataset into independent parts which have no overlapping peptides between them.

get_kint_array

property X_norm
ndarray: *X* coefficient matrix normalized column wise.

calc_kint(*temperature*, *pH*, *c_term*)
Calculates the intrinsic rate of the sequence. Values of no coverage or prolines are assigned a value of -1 The rates run are for the first residue (1) up to the last residue that is covered by peptides

When the previous residue is unknown the current residue is also assigned a value of -1.g

Parameters

temperature: [float] Temperature of the labelling reaction (Kelvin)

pH [float] pH of the labelling reaction

c_term [int] index of the last residue in the sequence (first residue is 1)

Returns

k_int [~class:~*numpy.ndarray*] Array of intrisic exchange rates

property cov_sequence
amino acids one letter codes corresponding to r_number array

get_sections(*gap_size=- 1*)
get the intervals of sections of coverage intervals are inclusive, exclusive

gap_size: int

Gaps of this size between adjacent peptides is not considered to overlap. A value of -1 means that peptides with exactly zero overlap are separated. With gap_size=0 peptides with exactly zero overlap are not separated, and larger values tolerate larger gap sizes.

property has_coverage
ndarray: Boolean array indicating if the residues along r_number have coverage

property sequence
str: String of the full protein sequence. One letter coding where X marks regions of no coverage

property sequence_r_number
~class:*numpy.ndarray*: Array of r numbers corresponding to residues in sequence

split(*gap_size=- 1*)
Splits the dataset into independent parts which have no overlapping peptides between them. To determine overlap, the modified 'start' and 'end' fields are used which take into account N-terminal non-exchanging residues and prolines.

Returns

output: **dict** Dictionary where keys are {start}_{end} (inclusive, exclusive) of the corresponding sections, values are of the `type` of the current instance.

gap_size: **int** Gaps of this size between adjacent peptides is not considered to overlap. A value of -1 means that peptides with exactly zero overlap are separated. With gap_size=0 peptides with exactly zero overlap are not separated, and larger values tolerate larger gap sizes.

pyhdx.models.**contiguous_regions**(*condition*)
Finds contiguous True regions of the boolean array "condition". Returns a 2D array where the first column is the start index of the region and the second column is the end index.

5.2 Fitting

class pyhdx.fitting.**EmptyResult**(*chi_squared*, *params*)

Attributes

chi_squared Alias for field number 0

params Alias for field number 1

property chi_squared
Alias for field number 0

property params
Alias for field number 1

class pyhdx.fitting.**KineticsFitResult**(*series*, *intervals*, *results*, *models*)
this fit results is only for wt avg fitting

Attributes

model_type

output

rate Returns an array with the exchange rates

tau Returns an array with the exchange rates

Methods

`__call__`(timepoints)	call the result with timepoints to get fitted uptake per peptide back
get_d(t)	calculate d at timepoint t only for lsqkinetics (refactor glocal) type fitting results (scores per peptide)
get_p(t)	Calculate P at timepoint t.
get_param(name)	Get an array of parameter with name *name* from the fit result.

get_output	

get_d(*t*)
calculate d at timepoint t only for lsqkinetics (refactor glocal) type fitting results (scores per peptide)

get_p(*t*)
Calculate P at timepoint t. Only for wt average type fitting results

get_param(*name*)
Get an array of parameter with name *name* from the fit result. The length of the array is equal to the number of amino acids.

Parameters

name [str] Name of the parameter to extract

Returns

par_arr [ndarray] Array with parameter values

property rate
Returns an array with the exchange rates

property tau
Returns an array with the exchange rates

class pyhdx.fitting.**KineticsModel**(*bounds*)
Base class for kinetics models. Main function is to generate symfit Variables and Parameters. The class attributes *par_index* and *var_index* are used to make sure names used by symfit are unique and their mapping to user-defined names are stored in the *names* dictionary.

Parameters

bounds [tuple] Tuple of default *min*, *max* parameters to use.

Attributes

names [dict] Dictionary which maps human-readable names (keys) to dummy names (values)

sf_model [Model] The *symfit* model which describes this model. Implemented by subclasses.

Methods

get_parameter(name)	Get the parameter with the Human-readable name *name*
make_parameter(name[, value, min, max])	Create a new :class:~symfit.Parameter.
make_variable(name)	Create a new :class:~symfit.Variable.

get_parameter(*name*)
Get the parameter with the Human-readable name *name*

Parameters

name [str] Name of the parameter to retrieve

Returns

parameter [Parameter]

make_parameter(*name*, *value=None*, *min=None*, *max=None*)
Create a new :class:~symfit.Parameter.

Parameters

name: :obj:`str` Human-readable name for the parameter

value: :obj:`float` Initial guess value

min: :obj:`float` Lower bound value. If *None*, the value from *bounds* is used.

max: :obj:`float` Lower bound value. If *None*, the value from *bounds* is used.

Returns

p [Parameter]

make_variable(*name*)
Create a new :class:~symfit.Variable.

Parameters

name: :obj:`str` Human-readable name for the variable

Returns

p [Variable]

property r_names
dict: Reverse dictionary of the variable and parameter names

class pyhdx.fitting.**LSQKinetics**(*initial_result*, *k_series*, *blocks*, *bounds*, *model_type='association'*)

Methods

__call__(t, **params)	returns the callled model at time t for params, returns uptake values of peptides
get_param_values(name, **params)	returns a list of parameters with name name which should have been indexed parameters params repeat during blocks

continues on next page

Table 5.8 – continued from previous page

get_rate(**params)	**Parameters**
get_tau(**params)	**Parameters**

min_func	

get_param_values (*name*, ***params*)
returns a list of parameters with name name which should have been indexed parameters params repeat during blocks

get_rate (***params*)

Parameters

params

key value where keys are the dummy names

get_tau (***params*)

Parameters

params

key value where keys are the dummy names

class pyhdx.fitting.**OneComponentAssociationModel** (*bounds*)
One component Association

Methods

__call__(t, **params)	call model at time t, returns uptake values of peptides
initial_guess(t, d)	Calculates initial guesses for fitting of two-component kinetic uptake reaction

get_rate	
get_tau	

initial_guess (*t*, *d*)
Calculates initial guesses for fitting of two-component kinetic uptake reaction

Parameters

t [:class:~`numpy.ndarray`] Array with time points

d [:class:~`numpy.ndarray`] Array with uptake values

class pyhdx.fitting.**OneComponentDissociationModel** (*bounds*)
One component Association

Methods

`__call__`(t, **params)	call model at time t, returns uptake values of peptides
initial_guess(t, d)	Calculates initial guesses for fitting of two-component kinetic uptake reaction

get_rate	
get_tau	

initial_guess(*t*, *d*)
Calculates initial guesses for fitting of two-component kinetic uptake reaction

Parameters

t [:class:~`numpy.ndarray`] Array with time points

d [:class:~`numpy.ndarray`] Array with uptake values

class pyhdx.fitting.**SingleKineticModel**(*bounds*)
Base class for models which fit only a single set (slice) of time, uptake points

class pyhdx.fitting.**TwoComponentAssociationModel**(*bounds*)
Two componenent Association

Methods

`__call__`(t, **params)	call model at time t, returns uptake values of peptides
get_rate(**params)	**Parameters**
get_tau(**params)	**Parameters**
initial_guess(t, d)	Calculates initial guesses for fitting of two-component kinetic uptake reaction

initial_grid	
min_func	

get_rate(***params*)

Parameters

params

key value where keys are the dummy names

get_tau(***params*)

Parameters

params

key value where keys are the dummy names

initial_guess(*t*, *d*)
Calculates initial guesses for fitting of two-component kinetic uptake reaction

Parameters

t [:class:~`numpy.ndarray`] Array with time points

d [:class:~`numpy.ndarray`] Array with uptake values

class pyhdx.fitting.**TwoComponentDissociationModel**(*bounds*)
Two componenent Association

Methods

__call__(t, **params)	call model at time t, returns uptake values of peptides
get_rate(**params) **Parameters**	
get_tau(**params) **Parameters**	
initial_guess(t, d)	Calculates initial guesses for fitting of two-component kinetic uptake reaction

initial_grid	
min_func	

get_rate(***params*)

Parameters

params

key value where keys are the dummy names

get_tau(***params*)

Parameters

params

key value where keys are the dummy names

initial_guess(*t*, *d*)
Calculates initial guesses for fitting of two-component kinetic uptake reaction

Parameters

t [:class:~`numpy.ndarray`] Array with time points

d [:class:~`numpy.ndarray`] Array with uptake values

pyhdx.fitting.**fit_kinetics**(*t*, *d*, *model*, *chisq_thd*)
Fit time kinetics with two time components and corresponding relative amplitude.

Parameters

t [ndarray] Array of time points

d [ndarray] Array of uptake values

chisq_thd: :obj:`float` Threshold chi squared above which the fitting is repeated with the Differential Evolution algorithm.

Returns

res [FitResults] Symfit fitresults object.

pyhdx.fitting.**func_long_ass**(*k*, *tt*, *A*, *k1*)
Function to estimate the short time component

Parameters

k [float] rate

tt [float] Selected time point

A [float] Target amplitude

k1: [obj:*float*] Rate of fast time component

Returns

A_t [float] Amplitude difference given tau, tt, A, tau1

pyhdx.fitting.**func_long_dis**(*k*, *tt*, *A*, *k1*)
Function to estimate the short time component

Parameters

k [float] rate

tt [float] Selected time point

A [float] Target amplitude

k1: [obj:*float*] Rate of fast time component

Returns

A_t [float] Amplitude difference given tau, tt, A, tau1

pyhdx.fitting.**func_short_ass**(*k*, *tt*, *A*)
Function to estimate the fast time component

Parameters

k [float] Lifetime

tt [float] Selected time point

A [float] Target amplitude

Returns

A_t [float] Amplitude difference given tau, tt, A

pyhdx.fitting.**func_short_dis**(*k*, *tt*, *A*)
Function to estimate the fast time component

Parameters

k [float] Lifetime

tt [float] Selected time point

A [float] Target amplitude

Returns

A_t [float] Amplitude difference given tau, tt, A

5.3 Fitting TensorFlow

class pyhdx.fitting_tf.**Between**(*min_value*, *max_value*)
Interval parameter constraint.

Constrains the values of parameters to the interval [min_value, max_value].

Parameters

min_value: :obj:`float` Lower bound for the allowed interval (optional *None*).

max_value: :obj:`float` Upper bound for the allowed interval (optional *None*).

Methods

__call__(w)	Call self as a function.

get_config	

class pyhdx.fitting_tf.**CurveFit**(*params*, *function*, ***kwargs*)

Methods

build(input_shape)	Creates the variables of the layer (optional, for subclass implementers).
call(inputs, **kwargs)	This is where the layer's logic lives.
compute_output_shape(input_shape)	Computes the output shape of the layer.

build(*input_shape*)
Creates the variables of the layer (optional, for subclass implementers).

This is a method that implementers of subclasses of *Layer* or *Model* can override if they need a state-creation step in-between layer instantiation and layer call.

This is typically used to create the weights of *Layer* subclasses.

Arguments:

input_shape: Instance of *TensorShape*, or list of instances of *TensorShape* if the layer expects a list of inputs (one instance per input).

call(*inputs*, ***kwargs*)
This is where the layer's logic lives.

Arguments: inputs: Input tensor, or list/tuple of input tensors. **kwargs: Additional keyword arguments.

Returns: A tensor or list/tuple of tensors.

compute_output_shape(*input_shape*)
Computes the output shape of the layer.

If the layer has not been built, this method will call *build* on the layer. This assumes that the layer will later be used with inputs that match the input shape provided here.

Arguments:

input_shape: Shape tuple (tuple of integers) or list of shape tuples (one per output tensor of the layer). Shape tuples can include None for free dimensions, instead of an integer.

Returns: An input shape tuple.

class pyhdx.fitting_tf.**L1L2Differential**(*l1=0.0*, *l2=0.0*)

A regularized that applies and L1 or L2 regularization penalty to the differential of a parameter vector.

Parameters

l1: :obj:`float` L1 regularization factor

l2: :obj:`float` L2 regularization factor

Methods

__call__(x)	Compute a regularization penalty from an input tensor.
get_config()	Returns the config of the regularizer.

get_config()

Returns the config of the regularizer.

An regularizer config is a Python dictionary (serializable) containing all configuration parameters of the regularizer. The same regularizer can be reinstantiated later (without any saved state) from this configuration.

This method is optional if you are just training and executing models, exporting to and from SavedModels, or using weight checkpoints.

This method is required for Keras *model_to_estimator*, saving and loading models to HDF5 formats, Keras model cloning, some visualization utilities, and exporting models to and from JSON.

Returns: Python dictionary.

class pyhdx.fitting_tf.**LossHistory**(*verbose=False*)

Methods

on_epoch_end(epoch[, logs])	Called at the end of an epoch.

on_epoch_end(*epoch*, *logs=None*)

Called at the end of an epoch.

Subclasses should override for any actions to run. This function should only be called during TRAIN mode.

Arguments: epoch: integer, index of epoch. logs: dict, metric results for this training epoch, and for the validation epoch if validation is performed. Validation result keys are prefixed with *val_*.

class pyhdx.fitting_tf.**NaNMeanSquaredError**(*reduction='auto'*, *name=None*)

MSE which ignores nan entries

Parameters

y_true:

Methods

call(y_true, y_pred)	Invokes the *Loss* instance.

call(*y_true*, *y_pred*)
Invokes the *Loss* instance.

Args: y_true: Ground truth values, with the same shape as 'y_pred'. y_pred: The predicted values.

class pyhdx.fitting_tf.**TFFitResult**(*series*, *intervals*, *funcs*, *weights*, *inputs*, *loss=None*)

Parameters

r_number list or r numbers these results cover

intervals (inclusive, exclusive) intervals which map results, models to r numbers (can be obtained from series)

funcs: assumed to be tghe same

assumed to be the same for all intervals

weights: list of weights (parameters) at lowest loss

Attributes

output

Methods

__call__(timepoints)	output: N x M array (peptides, timepoints)

class pyhdx.fitting_tf.**TFParameter**(*name*, *shape*, *initializer=None*, *regularizer=None*, *constraint=None*)
Parameter objects used in *CurveFit* TensorFlow Layer. Parameters are 'weights' in the context of Neural Networks.

Parameters

name: :obj:`str` Name of the parameter

shape: :obj:`tuple` Parameter shape

initializer: :class:`~tensorflow.python.keras.initializers.Initializer` Subclass of Keras Initializer to initialize parameter elements.

regularizer :class:`~tensorflow.python.keras.regularizers.Regularizer` Subclass of Keras Regularizer applied to parameter elements.

constraint :class:`~tensorflow.python.keras.constraints.Constraint` Subclass of keras Constraint applied to parameter elements.

5.4 FileIO

5.5 Output

class pyhdx.output.**Report**(*output*, *name=None*, *doc=None*, *add_date=True*)
.pdf output document

Methods

rm_temp_dir()	Remove the temporary directory specified in _tmp_path.

add_coverage_figures	
add_peptide_figures	
generate_pdf	
make_subfigure	
make_temp_dir	
test_mpl	
test_subfigure	

rm_temp_dir()
Remove the temporary directory specified in _tmp_path.

5.6 Support

pyhdx.support.**autowrap**(*coverage*, *margin=4*)
Automatically finds wrap value for coverage to not have overlapping peptides within margin

pyhdx.support.**colors_to_pymol**(*r_number*, *color_arr*, *c_term=None*, *no_coverage='#8c8c8c'*)
coverts colors (hexadecimal format) and corresponding residue numbers to pml script to color structures in pymol residue ranges in output are inclusive, incluive

c_term: optional residue number of the c terminal of the last peptide doedsnt cover the c terminal

pyhdx.support.**gen_subclasses**(*cls*)
Recursively find all subclasses of cls

pyhdx.support.**grouper**(*3*, *'abcdefg'*, *'x') --> ('a'*, *'b'*, *'c')*, *('d'*, *'e'*, *'f')*, *('g'*, *'x'*, *'x'*)

pyhdx.support.**make_color_array**(*rates*, *colors*, *thds*, *no_coverage='#8c8c8c'*)

Parameters

- **rates** – array of rates
- **colors** – list of colors (slow to fast)
- **thds** – list of thresholds

no_coverage: color value for no coverage :return:

pyhdx.support.**make_monomer**(*input_file*, *output_file*)
reads input_file pdb file and removes all chains except chain A and all water

pyhdx.support.**multi_otsu**(**rates*, *classes=3*)
global otsu thesholding of multiple rate arrays in log space

Parameters

rates: iterable iterable of numpy structured arrays with a 'rate' field

classes: :obj:`int` Number of classes to divide the data into

Returns

thds: `obj`:tuple: tuple with thresholds

pyhdx.support.**reduce_inter**(*args*, *gap_size=- 1*)

gap_size: int Gaps of this size between adjacent peptides is not considered to overlap. A value of -1 means that peptides with exactly zero overlap are separated. With gap_size=0 peptides with exactly zero overlap are not separated, and larger values tolerate larger gap sizes.

https://github.com/brentp/interlap/blob/3c4a5923c97a5d9a11571e0c9ea5bb7ea4e784ee/interlap.py#L224 # MIT Liscence >>> reduce_inter([(2, 4), (4, 9)]) [(2, 4), (4, 9)] >>> reduce_inter([(2, 6), (4, 10)]) [(2, 10)]

pyhdx.support.**scale**(*x*, *out_range=- 1, 1*)
rescale input array x to range *out_range*

pyhdx.support.**series_intersection**(*series_list*)
finds and returns series where peptides are the intersection of all series

pyhdx.support.**try_wrap**(*coverage*, *wrap*, *margin=4*)
Check for a given coverage if the value of wrap is high enough to not have peptides overlapping within margin

CHAPTER

SIX

WEB APPLICATION

This page contains auto-generated docs for PyHDX' web application.

There are three applications available:

- **Main Application** Fitting of HDX-MS datasets, classification, visualization and exporting data.
- **Single Classification** Reload exported data from the main application for classification, visualization and exporting data.
- **Binary Comparison** Reload multiple exported datasets from the main application and calculate differences between pairs of datasets. The resulting differences can again be classified, visualized and exported.

The functionality in each app can be controlled by *Controllers* which can be found in the left sidebar. The functionality of every controller per app is listed in the sections below.

6.1 Main Application

class pyhdx.panel.controllers.**PeptideFileInputControl**(*parent*, ***params*)

Peptide Input

This controller allows users to input .csv file (Currently only DynamX format) of 'state' peptide uptake data. Users can then choose how to correct for back-exchange and which 'state' and exposure times should be used for analysis.

Add File (*Action*)
Add File

Clear Files (*Action*)
Clear files

Drop first (*Integer*, bounds=(0, None), default=1)
Select the number of N-terminal residues to ignore.

Ignore prolines (*Boolean*, bounds=(0, 1), default=True)
Prolines are ignored as they do not exchange D.

Load Files (*Action*)

Load the selected files

Norm mode (*Selector*, default='Exp', options=['Exp', 'Theory'])
Select method of normalization

Norm State (*Selector*, options=[])
State used to normalize uptake

Norm exposure (*Selector*, options=[])
Exposure used to normalize uptake

Back exchange percentage (*Number*, bounds=(0, 100), default=28.0)
Global percentage of back-exchange

Experiment State (*Selector*, options=[])
State for selected experiment

Experiment Exposures (*ListSelector*, default=[], options=[''])
Selected exposure time to use

Parse (*Action*)
Parse selected peptides for further analysis and apply back-exchange correction

class pyhdx.panel.controllers.**CoverageControl**(*parent*, ***params*)

Coverage

Wrap (*Integer*, bounds=(0, None), default=25)
Number of peptides vertically before moving to the next row.

Color map (*Selector*, default='jet', options=['jet', 'inferno', 'viridis', 'cividis', 'plasma', 'cubehelix'])
Color map for coloring peptides by their deuteration percentage.

Index (*Integer*, bounds=(0, 10), default=0)
Current index of coverage plot in time.

class pyhdx.panel.controllers.**InitialGuessControl**(*parent*, ***params*)

Initial Guesses

This controller allows users to derive initial guesses for D-exchange rate from peptide uptake data

Fitting model (*Selector*, default='Half-life (λ)', options=['Half-life (λ)', 'Association'])
Choose method for determining initial guesses.

Do fitting (*Action*)
Start initial guess fitting

class pyhdx.panel.controllers.**FitControl**(*parent*, ***params*)

Fitting

This controller allows users to execute TensorFlow fitting of the global data set.

Currently, repeated fitting overrides the old result.

Initial guess (*Selector*, options=[])
Name of dataset to use for initial guesses.

C term (*Integer*)
Residue number to which the last amino acid in the sequence corresponds.

Temperature (*Number*, default=293.15)
Deuterium labelling temperature in Kelvin

pH (*Number*, default=8.0)
Deuterium labelling pH

Stop loss (*Number*, bounds=(0, None), default=0.01)
Threshold loss difference below which to stop fitting.

Stop patience (*Integer*, bounds=(1, None), default=50)
Number of epochs where stop loss should be satisfied before stopping.

Learning rate (*Number*, bounds=(0, None), default=0.01)
Learning rate parameter for optimization.

Epochs (*Number*, bounds=(1, None), default=100000)

Maximum number of epochs (iterations.

L1 regularizer (*Number*, bounds=(0, None), default=20)
Value for l1 regularizer.

Do Fitting (*Action*)
Start TensorFlow global fitting

class pyhdx.panel.controllers.**ClassificationControl**(*parent*, ***param*)

Classification

This controller allows users classify 'mapping' datasets and assign them colors.

Coloring can be either in discrete categories or as a continuous custom color map.

Target (*Selector*, options=[])

Mode (*Selector*, default='Discrete', options=['Discrete', 'Continuous'])
Choose color mode (interpolation between selected colors).

Num colors (*Number*, bounds=(1, 10), default=3)
Number of classification colors.

Otsu (*Action*)
Automatically perform thresholding based on Otsu's method.

Linear (*Action*)
Automatically perform thresholding by creating equally spaced sections.

Log space (*Boolean*, bounds=(0, 1), default=True)
Boolean to set whether to apply colors in log space or not.

Show Thresholds (*Boolean*, bounds=(0, 1), default=True)
Toggle to show/hide threshold lines.

class pyhdx.panel.controllers.**FileExportControl**(*parent*, ***param*)

File Export

This controller allows users to export and download datasets.

All datasets can be exported as .txt tables. 'Mappable' datasets (with r_number column) can be exporeted as .pml pymol script, which colors protein structures based on their 'color' column.

Target dataset (*Selector*, options=[])
Name of the dataset to export

C term (*Integer*, bounds=(0, None), default=0)

class pyhdx.panel.controllers.**ProteinViewControl**(*parent*, ***params*)

Protein Viewer

This controller allows users control the Protein view figure. Structures can be specified either by RCSB ID or uploading a .pdb file.

Colors are assigned according to 'color' column of the selected dataset.

Target dataset (*Selector*, options=[])
Name of the dataset to apply coloring from

Input option (*Selector*, default='Upload File', options=['Upload File', 'RCSB PDB'])
Choose wheter to upload .pdb file or directly download from RCSB PDB.

Rcsb id (*String*, default='')
RCSB PDB identifier of protein entry to download and visualize.

No coverage (*Color*, default='#8c8c8c')
Color to use for regions of no coverage.

Representation (*Selector*, default='cartoon', options=['backbone', 'ball+stick', 'cartoon', 'hyperball', 'licorice', 'ribbon', 'rope', 'spacefill', 'surface'])
Representation to use to render the protein.

Spin (*Boolean*, bounds=(0, 1), default=False)
Rotate the protein around an axis.

class pyhdx.panel.controllers.**OptionsControl**(*parent*, ***param*)

Options

The controller is used for various settings.

Link xrange (*Boolean*, bounds=(0, 1), default=True)
Link the X range of the coverage figure and other linear mapping figures.

Log level (*Selector*, default='DEBUG', options=['DEBUG', 'INFO', 'WARN', 'ERROR', 'FATAL', 'OFF', 'TRACE'])
Set the logging level.

6.2 Single Classification

class pyhdx.panel.controllers.**MappingFileInputControl**(*parent*, ***params*)

File Input

This controller allows users to upload *.txt files where quantities (protection factors, Gibbs free energy, etc) are mapped to a linear sequence.

The column should be tab separated with on the last header line (starts with '#') the names of the columns. Columns should be tab-delimited.

Input file (*Parameter*)
Input file to add to available datasets

Dataset name (*String*, default='')
Name for the dataset to add. Defaults to filename

Add dataset (*Action*)
Add the dataset to available datasets

Datasets (*ListSelector*, options=[])
Current datasets

Remove dataset (*Action*)
Remove selected datasets

class pyhdx.panel.controllers.**SingleControl**(*parent*, ***params*)

Datasets

Dataset (*Selector*, options=[])
ds1

Dataset name (*String*, default=”)

Quantity (*Selector*, options=[])
Select a quantity to plot (column from input txt file)

Add dataset (*Action*)
Click to add this comparison to available comparisons

Dataset list (*ListSelector*, options=[])
Lists available comparisons

Remove dataset (*Action*)

class pyhdx.panel.controllers.**ClassificationControl**(*parent*, ***param*)

Classification

This controller allows users classify ‘mapping’ datasets and assign them colors.

Coloring can be either in discrete categories or as a continuous custom color map.

Target (*Selector*, options=[])

Mode (*Selector*, default=’Discrete’, options=[‘Discrete’, ‘Continuous’])
Choose color mode (interpolation between selected colors).

Num colors (*Number*, bounds=(1, 10), default=3)
Number of classification colors.

Otsu (*Action*)
Automatically perform thresholding based on Otsu’s method.

Linear (*Action*)
Automatically perform thresholding by creating equally spaced sections.

Log space (*Boolean*, bounds=(0, 1), default=True)
Boolean to set whether to apply colors in log space or not.

Show Thresholds (*Boolean*, bounds=(0, 1), default=True)
Toggle to show/hide threshold lines.

class pyhdx.panel.controllers.**ProteinViewControl** (*parent*, ***params*)

Protein Viewer

This controller allows users control the Protein view figure. Structures can be specified either by RCSB ID or uploading a .pdb file.

Colors are assigned according to 'color' column of the selected dataset.

Target dataset (*Selector*, options=[])
Name of the dataset to apply coloring from

Input option (*Selector*, default='Upload File', options=['Upload File', 'RCSB PDB'])
Choose wheter to upload .pdb file or directly download from RCSB PDB.

Rcsb id (*String*, default='')
RCSB PDB identifier of protein entry to download and visualize.

No coverage (*Color*, default='#8c8c8c')
Color to use for regions of no coverage.

Representation (*Selector*, default='cartoon', options=['backbone', 'ball+stick', 'cartoon', 'hyperball', 'licorice', 'ribbon', 'rope', 'spacefill', 'surface'])
Representation to use to render the protein.

Spin (*Boolean*, bounds=(0, 1), default=False)
Rotate the protein around an axis.

class pyhdx.panel.controllers.**DifferenceFileExportControl**(*parent*, ***param*)

File Export

Additional GUI elements on:

pyhdx.panel.controllers.FileExportControl: **target**, **c_term**

class pyhdx.panel.controllers.**OptionsControl**(*parent*, ***param*)

Options

The controller is used for various settings.

Link xrange (*Boolean*, bounds=(0, 1), default=True)
Link the X range of the coverage figure and other linear mapping figures.

Log level (*Selector*, default='DEBUG', options=['DEBUG', 'INFO', 'WARN', 'ERROR', 'FATAL', 'OFF', 'TRACE'])
Set the logging level.

6.3 Binary Comparison

class pyhdx.panel.controllers.**MappingFileInputControl**(*parent*, ***params*)

File Input

This controller allows users to upload *.txt files where quantities (protection factors, Gibbs free energy, etc) are mapped to a linear sequence.

The column should be tab separated with on the last header line (starts with '#') the names of the columns. Columns should be tab-delimited.

Input file (*Parameter*)
Input file to add to available datasets

Dataset name (*String*, default='')
Name for the dataset to add. Defaults to filename

Add dataset (*Action*)
Add the dataset to available datasets

Datasets (*ListSelector*, options=[])
Current datasets

Remove dataset (*Action*)
Remove selected datasets

class pyhdx.panel.controllers.**DifferenceControl**(*parent*, ***params*)

Differences

This controller allows users to select two datasets from available datasets, choose a quantity to compare between, and choose the type of operation between quantities (Subtract/Divide).

Dataset 1 (*Selector*, options=[])
First dataset to compare

Dataset 2 (*Selector*, options=[])
Second dataset to compare

Comparison name (*String*, default='')

Operation (*Selector*, default='Subtract', options=['Subtract', 'Divide'])
Select the operation to perform between the two datasets

Comparison quantity (*Selector*, options=[])
Select a quantity to compare (column from input txt file)

Add comparison (*Action*)
Click to add this comparison to available comparisons

Comparison list (*ListSelector*, options=[])
Lists available comparisons

Remove comparison (*Action*)
Remove selected comparisons from the list

class pyhdx.panel.controllers.**ClassificationControl**(*parent*, ***param*)

Classification

This controller allows users classify 'mapping' datasets and assign them colors.

Coloring can be either in discrete categories or as a continuous custom color map.

Target (*Selector*, options=[])

Mode (*Selector*, default='Discrete', options=['Discrete', 'Continuous'])
Choose color mode (interpolation between selected colors).

Num colors (*Number*, bounds=(1, 10), default=3)
Number of classification colors.

Otsu (*Action*)
Automatically perform thresholding based on Otsu's method.

Linear (*Action*)
Automatically perform thresholding by creating equally spaced sections.

Log space (*Boolean*, bounds=(0, 1), default=True)
Boolean to set whether to apply colors in log space or not.

Show Thresholds (*Boolean*, bounds=(0, 1), default=True)
Toggle to show/hide threshold lines.

class pyhdx.panel.controllers.**ProteinViewControl**(*parent*, ***params*)

Protein Viewer

This controller allows users control the Protein view figure. Structures can be specified either by RCSB ID or uploading a .pdb file.

Colors are assigned according to 'color' column of the selected dataset.

Target dataset (*Selector*, options=[])
Name of the dataset to apply coloring from

Input option (*Selector*, default='Upload File', options=['Upload File', 'RCSB PDB'])
Choose wheter to upload .pdb file or directly download from RCSB PDB.

Rcsb id (*String*, default='')
RCSB PDB identifier of protein entry to download and visualize.

No coverage (*Color*, default='#8c8c8c')
Color to use for regions of no coverage.

Representation (*Selector*, default='cartoon', options=['backbone', 'ball+stick', 'cartoon', 'hyperball', 'licorice', 'ribbon', 'rope', 'spacefill', 'surface'])
Representation to use to render the protein.

Spin (*Boolean*, bounds=(0, 1), default=False)
Rotate the protein around an axis.

class pyhdx.panel.controllers.**DifferenceFileExportControl**(*parent*, ***param*)

File Export

Additional GUI elements on:

pyhdx.panel.controllers.FileExportControl: **target**, **c_term**

class pyhdx.panel.controllers.**OptionsControl**(*parent*, ***param*)

Options

The controller is used for various settings.

Link xrange (*Boolean*, bounds=(0, 1), default=True)
Link the X range of the coverage figure and other linear mapping figures.

Log level (*Selector*, default='DEBUG', options=['DEBUG', 'INFO', 'WARN', 'ERROR', 'FATAL', 'OFF', 'TRACE'])
Set the logging level.

CHAPTER

SEVEN

CONTRIBUTING

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

7.1 Types of Contributions

7.1.1 Report Bugs

Report bugs at https://github.com/Jhsmit/pyhdx/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

7.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” and “help wanted” is open to whoever wants to implement it.

7.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with “enhancement” and “help wanted” is open to whoever wants to implement it.

7.1.4 Write Documentation

PyHDX could always use more documentation, whether as part of the official PyHDX docs, in docstrings, or even on the web in blog posts, articles, and such.

7.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/Jhsmit/pyhdx/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

7.2 Get Started!

Ready to contribute? Here's how to set up *pyhdx* for local development.

1. Fork the *pyhdx* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/pyhdx.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv pyhdx
$ cd pyhdx/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 pyhdx tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

7.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.7, 3.4, 3.5 and 3.6, and for PyPy. Check https://travis-ci.org/Jhsmit/pyhdx/pull_requests and make sure that the tests pass for all supported Python versions.

7.4 Tips

To run a subset of tests:

```
$ py.test tests.test_pyhdx
```

7.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bumpversion patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER

EIGHT

CREDITS

8.1 Development Lead

- Jochem Smit <jhsmit@gmail.com>

8.2 Contributors

None yet. Why not be the first?

CHAPTER

NINE

HISTORY

9.1 0.1.0 (2019-09-06)

- First release on PyPI.

CHAPTER

TEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

INDEX

H

I

K

L

M

N

O

P

R

S

T

U

X