Virtual Patient Cohorts

Release 2024

Alisa Ebert, David Hasse

Apr 17, 2024

CONTENTS:

CHAPTER

ONE

MAIN

main.**handle_leftclick**(*event: Event*) → None

Helper function to set the focus to the widget associated with the current event `event`.

Is used to set the focus to whatever widget is clicked by the user.

Parameters

event (*Event*) – The event, i.e. left mouse click.

main.**main**()

Main function.

Setting up `argparse` to catch the `--debug` flag on startup, running `setup_logging()` and finally running the program, i.e. its `mainloop()`.

main.**setup_logging**(*debug: bool = False*)

Logging setup for the program.

Logs are saved in a `./logs/` directory relative to the current working directory, which will be created if it's not present. Logging config uses a 'logging_config.json' file in this module's directory.

Parameters

debug (*bool, optional*) – flag that gets passed through by argparse to enable logging on `debug` level, defaults to False

CHAPTER

TWO

CTKINTERFACE MODULE

class src.CTkInterface.**MainApp**

Bases: CTk

Main interface of the program. Providing ways for the user to input information that gets checked for to finally be able to fit a model to data.

Uses a `tkinter` extension called `customtkinter` to achieve a modern look.

__init__() → None

Setup of the main interface window seen after starting the program, arranging widgets and setting default values.

browse_files() → None

Opens a `tkinter.filedialog` to let the user select a file containing fitting data. Sets internal variables to the file path and name when a file is successfully selected.

check_inputs_populated() → str

Checks the various fields for missing user input.

Returns
Message with missing or invalid entried, empty if okay.

Return type
str

check_inputs_sensible() → str

Checks whether the user input 'makes sense', e.g. if there are unfitted constants in the model expression.

Returns
Message with apparent issues in the user's input.

Return type
str

compute_params() → None

Runs *ModelFitter.fit()* on the model and data, opens a window with the results.

confirm_input() → None

Checks inputs and displays any issues. If okay, sets internal variables.

create_interpretation_string(*model:* VPCModel | *None* = *None*, ***kwargs: str* | *None*) → str

Creates the interpretation string of the user input.

Parameters

- **model** (VPCModel, *optional*) – Model to be iterpreted, defaults to None.

- **kwargs** (*str | None*) – Additional keyword arguments, defaults to None.

Returns
A multi-line f-string containing the entered information.

Return type
str

`create_tooltip_for`(*widget: Widget*, *msg: str*) → ToolTip

Helper Function to create a `ToolTip` for and attach it to a `tkinter.Widget`.

Parameters

- **widget** (*Widget*) – The `tkinter.Widget` the `ToolTip` is binded to.
- **msg** (*str*) – What the `ToolTip` should say.

Returns
A new `ToolTip` instance.

Return type
ToolTip

`display_interpreted_input`(*msg: str*) → None

Shows the interpretation of the user input.

Parameters
msg (*str*) – interpretation of the user input

`missing_independent_variables`() → list[str]

Returns list of all characters or strings that were entered in the independent parameter field that are not present in the model string. If the field was left empty 't' is defaulted to.

Returns
List of missing symbols.

Return type
list

`remove_compute_tooltip`() → None

Removes the tooltip and enables the 'compute parameters' button.

`reset_state`() → None

Resets the program to an initial state without closing and reopening it.

Raises
AttributeError – if one of the internal attributes that should have been set are not found and therefore couldn't be reset.

`save_results`() → int

Saves relevant inputs and outputs of the program in a file.

Returns
0 if `FileHandler.write_file()` finished normally, 1 otherwise.

Return type
int

CHAPTER

THREE

CTKRESULTINTERFACE MODULE

class src.CTkResultInterface.**FailedFitInterface**(*main_window:* MainApp)

Bases: CTkToplevel

Intereface for displaying a message for a failed fit.

__init__(*main_window:* MainApp)

Initialize the failed fit interface. :param main_window: Parent window. :type main_window: MainApp

reset_app() → None

Reset the parent application's state and close this window.

class src.CTkResultInterface.**ResultInterface**(*main_window:* MainApp, *model:* VPCModel, *fitted_model:* VPCModel, *data: list[list[int | float]]*)

Bases: CTkToplevel

Interface for results created by ModelFitter.

Provides a way to save results locally and view a plot of the residuals of the fit if possible.

Uses a tkinter extension called customtkinter to achieve a modern look.

__init__(*main_window:* MainApp, *model:* VPCModel, *fitted_model:* VPCModel, *data: list[list[int | float]]*) → None

Setup of the result interface window, arranging widgets and setting values.

Parameters

- **main_window** (MainApp) – Parent window.
- **model** (VPCModel) – Model that was fitted to the sample data.
- **fitted_model** (VPCModel) – Fitted model as a means to get easy access to the fitted model's lambda function.
- **data** (*list[list[int | float]]*) – The data the input was fitted to.

create_difference_dict(*list_actual: list[tuple[float, ...]]*, *list_predicted: list[tuple[float, ...]]*) → dict[int, list[float]]

Used to compute the differences (residuals) of two input lists created by process_lists(). Stores the differences at each index in a dictionary with index:difference pairs.

Parameters

- **list_actual** (*list[tuple[float, ...]]*) – list of the actual values.
- **list_predicted** (*list[tuple[float, ...]]*) – list of the predicted values.

Raises

- **ValueError** – If the input lists somehow have different lengths.
- **ValueError** – If a pair of tuples somehow have different lengths.

Returns
Dictionary with the differences at their corresponding index.

Return type
dict[int,list[float]]

graph_residuals() → None

Method that uses matplotlib to graph the residuals of the fitted model against the input data after the graph button is pressed.

process_lists(*lst: list[list[float]]*) → list[float] | list[tuple[float, ...]]

Converts a list of lists of floats into a list of floats, if there is only a single sub-list.

If there are multiple sub-lists inside the input list, will zip them and return a list of tuples of floats.

Parameters
lst (*list[list[float]]*) – List that is to be converted, typically the data the model was fitted to, or part of it.

Raises
ValueError – If there are types other than `list` in the outer list.

Returns
List of floats or list of tuples of floats.

Return type
list[float] | list[tuple[float, ...]]

reset_app() → None

Method that calls the parent's `reset_state()` method to reset the app to an initial state after the reset button is pressed.

save_as() → None

Method that calls the parent's `save_results()` method and sets the saved_message_label's value accordingly after the save button is pressed.

set_result_label_text() → None

Method that sets the text inside the results textbox.

CHAPTER

FOUR

FILEHANDLER MODULE

This module provides basic functionality to convert data stored in .xlsx or .csv files into python lists.

class src.FileHandler.**FileExtensions**(*value*, *names=None*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: Enum

Enum class that has all valid file extension names and suffixes.

CSV = 'CSV'

EXCEL = 'XLSX'

src.FileHandler.**dataframe_tolist**(*data_frame: DataFrame*) → list[list[float | int]]

Convert a *pd.DataFrame* to a list of lists containing its values.

Parameters
data_frame (*pd.DataFrame*) – *DataFrame* to be converted.

Raises
ValueError – If the *data_frame* is None, empty, contains empty cells, or non-numeric values.

Returns
List of lists containing the DataFrame's values.

Return type
list[list[float | int]]

src.FileHandler.**get_valid_filename**() → str

Create a valid, hopefully non-duplicate, string to use as a file name.

Returns
Stringified time from *datetime.now()* in the form of *%Y-%m-%d-result-from-%Hh%Mm*.

Return type
str

src.FileHandler.**is_extension_supported**(*file_path: str*) → bool

Check if the file extension at the given path is supported.

Parameters
file_path (*str*) – Path pointing to the file.

Returns
True if supported, False otherwise.

Return type
bool

src.FileHandler.**read_file**(*file_path: str*) → DataFrame

Read a *.csv* or *.xlsx* file from the given path and return it as a *pd.DataFrame*.

Uses pathlib's *is_file()* method to ensure there is a file at the given path. Then tries to obtain the file's suffix and checks for *.csv* or *.xlsx* formats, for which the corresponding pandas read method is called.

Parameters
file_path (*str*) – Path to the file to be read.

Raises

- **FileNotFoundError** – If no file exists at *file_path*.
- **ValueError** – If *file_path* has no suffix.
- **TypeError** – If the file is not either an Excel table or CSV file.

Returns
A *pd.DataFrame* containing all information that is read from the excel or csv file.

Return type
pd.DataFrame

src.FileHandler.**write_file**(*data_frame: DataFrame*, *file_format:* FileExtensions = *FileExtensions.EXCEL*, *destination: str = './res/'*) → None

Write the provided *pd.DataFrame* as either *.xlsx* or *.csv* to the *./res/* directory.

If there is no *./res/* directory relative to where the program was started from, that directory will be created.

Parameters

- **data_frame** (*pd.DataFrame*) – The data frame to be written.
- **file_format** (FileExtensions, *optional*) – Format of the written file, defaults to *FileExtensions.EXCEL*.
- **destination** – Destination directory, default to "./res/".

Raises
TypeError – If *file_format* was neither Excel nor CSV.

CHAPTER

FIVE

MODELDATA MODULE

class src.ModelData.**ModelData**(*fitted_model: str | None = None*, *fitted_consts: dict[str, float] | str | None = None*, *model: str = 'f(t) = ...'*, *user_input_model: str = 'f(t) = ...'*, *parameter: list[str] = <factory>*, *user_input_parameter: str = '...'*, *consts: list[str] = <factory>*, *user_input_consts: list[str] = <factory>*, *user_input_path: str = 'path/to/data'*)

Bases: `object`

Container class that stores information for writing into the results file.

Parameters

- **fitted_model** (*str | None*) – The fitted model as a string.
- **fitted_consts** (*dict[str, float] | str | None*) – Dictionary of constants with keys of type *str* and values of type *float*. Alternatively, a string containing the same information. Defaults to None.
- **model** (*str, optional*) – The model the program worked with. Defaults to "f(t) = …".
- **user_input_model** (*str, optional*) – The exact model the user entered. Defaults to "f(t) = …".
- **parameter** (*list[str], optional*) – The independent variable the program worked with. Defaults to ["…"].
- **user_input_parameter** (*str, optional*) – The exact independent variables the user entered. Defaults to "…".
- **consts** (*list[str], optional*) – The constants the program worked with. Defaults to ["…"].
- **user_input_consts** (*list[str], optional*) – The exact constants the user entered. Defaults to ["…"].
- **user_input_path** (*str, optional*) – The path to the data the user provided. Defaults to "path/to/data".

__init__(*fitted_model: str | None = None*, *fitted_consts: dict[str, float] | str | None = None*, *model: str = 'f(t) = ...'*, *user_input_model: str = 'f(t) = ...'*, *parameter: list[str] = <factory>*, *user_input_parameter: str = '...'*, *consts: list[str] = <factory>*, *user_input_consts: list[str] = <factory>*, *user_input_path: str = 'path/to/data'*) → None

consts: list[str]

create_dataframe_for(*format:* FileExtensions = *FileExtensions.EXCEL*) → DataFrame

Create a *pd.DataFrame* for the specified format, i.e. either Excel or CSV. The DataFrame contains information about the user inputs into the program and what the program made of those. It also contains the fit of the model if possible. If no fitted model string was given, a warning will be logged and the field in the DataFrame will read 'N/A'.

Parameters
format (FileExtensions, *optional*) – The format for which the DataFrame is constructed. Defaults to FileExtensions.EXCEL.

Returns
A DataFrame containing all input and output information.

Return type
pd.DataFrame

fitted_consts: dict[str, float] | str | None = None

fitted_model: str | None = None

model: str = 'f(t) = ...'

parameter: list[str]

user_input_consts: list[str]

user_input_model: str = 'f(t) = ...'

user_input_parameter: str = '...'

user_input_path: str = 'path/to/data'

CHAPTER

SIX

MODELFITTER MODULE

This module provides functions to fit a model to data.

The fitting routines are based on the assumption that the model is of type VPCModel with provided data consisting of a list of lists where each sub-list is a column from a datasheet.

src.ModelFitter.**check_model_is_valid_vector**(*model:* VPCModel, *columns: int*) → bool

Checks if the model is valid when comparing it the the sample data.

A model is an invalid if there are not enough columns in the data to account for all independent variables and the result components.

Parameters

- **model** (VPCModel) – The model to be checked.
- **columns** (*int*) – The number of columns in the provided data.

Raises

Exception – If there are too few or too many columns in the data. Too few make it impossible to fit the model, too many make it ambiguous as to what the extra columns are supposed to mean, or which columns are even to be regarded and which ones not.

Returns

Whether the model is a vector.

Return type

bool

src.ModelFitter.**evaluate_fit**(*pcov: ndarray[Any, dtype[_ScalarType_co]]*) → dict[str, float]

Evaluate the goodness of fit based on the covariance matrix.

Parameters

pcov (*2D-array*) – Covariance matrix of the fit.

Returns

Dictionary containing evaluation metrics.

Return type

dict

src.ModelFitter.**fit**(*model:* VPCModel, *data: list[list[int | float]]*) → None

Main entry function for the fitting process. Delegates `model` and `data` to the correct fitting routine based on whether it's an ordinary differential equation or not.

Parameters

- **model** (VPCModel) – The model that is supposed to be fitted.
- **data** (*list[list[int | float]]*) – The data the model is supposed to be fitted to.

CHAPTER

SEVEN

VPCMODEL MODULE

class src.VPCModel.**VPCModel**(*_model_string: str*, *_independent_var: list[str]*)

Bases: object

Dataclass to store a model and extract relevant information from it.

__init__(*_model_string: str*, *_independent_var: list[str]*) → None

property components: int

Property to return the number of components the result has, i.e. if it is a vector.

A property without an accompanying setter is used to prohibit setting this value.

Returns
Number of components of the model expression.

Return type
int

property constants: list[str]

Property to return the list of constants in the model expression that are to be fitted.

A property without an accompanying setter is used to prohibit setting this value.

Returns
List of all constants in the model expression.

Return type
list[str]

cut_off_lhs() → str

Cuts off the left hand side of an equation, indicated by an equals sign. Also strips leading and trailing spaces and replaces some characters if needed.

Returns
Model equation without left hand side, including the equals sign.

Return type
str

cut_off_rhs() → str

Cuts off the right hand side of an equation, indicated by an equals sign. Also strips leading and trailing spaces and replaces some characters if needed.

Returns
Model equation without right hand side, including the equals sign.

Return type
str

property expression_string: str

Property to return the expression string (right-hand side) of the class instance.

A property without an accompanying setter is used to prohibit setting this value.

Returns
The expression string of the model.

Return type
str

extract_symbols(*sorting_prio: list[str] | None = None*) → list[str]

Extract all unique symbols out of the model equations right-hand side.

Unique symbols are single or multi-character sequences consisting of letters. If a sorting prio is given, the characters in it are returned at the front of the output, the rest is ordered alphabetically.

Parameters
sorting_prio (*list[str] | None, optional*) – Characters to be placed at the start of the output , defaults to None

Returns
All unique symbols in the expression on the right-hand of the equation.

Return type
list[str]

property fitted_consts: dict[str, float]

Property to return the dictionary of fitted constants of the function. If this value is not set at the time of accessing it through this property, an empty dictionary will be returned.

A property without an accompanying setter is used to prohibit setting this value.

Returns
Dictionary with constant:value as its pairs.

Return type
dict[str, float]

format_eq(*equation: str*) → str

Replaces some characters for others in a string. Strips leading and trailing spaces.

Parameters
equation (*str*) – The string in which characters will be replaced.

Returns
Stripped input string with replaced characters.

Return type
str

property independent_var: list[str]

Property to return the list of independent variables of the class instance.

A property without an accompanying setter is used to prohibit setting this value.

Returns
List of all independent variables of the model.

Return type
list[str]

`is_ode()` → bool

Determines whether the model is an ODE of at most 2nd order.

Returns
True if at most 2nd order ODE, False otherwise.

Return type
bool

`is_vector()` → bool

Determine whether the function is a vector based on the number of its components.

Raises
`Exception` – If components are less than 1.

Returns
True if the model is a vector, False otherwise.

Return type
bool

`property model_function: FunctionClass`

Property to return the lambdified model function of the class instance.

A property without an accompanying setter is used to prohibit setting this value.

Returns
FunctionClass / lambda expression of the model function.

Return type
sympy.FunctionClass

`property model_string: str`

Property to return the model string of the class instance.

A property without an accompanying setter is used to prohibit setting this value.

Returns
The model string without leading/trailing spaces and some character replacements.

Return type
str

`model_string_to_function()` → FunctionClass

Create a lambda function based on the model equation.

Extracts all symbols from the expression and converts them to SymPy symbols. Finally lambdifies the expression to return a callable model function.

Returns
Lambdified model expression.

Return type
FunctionClass

`property resulting_function: str`

Property to return the stringified function with the computed constants in it. If this value is not set at the time of accessing it through this property, an empty string will be returned.

A property without an accompanying setter is used to prohibit setting this value.

Returns
The string of the final/resulting function after the fit.

Return type
str

`set_fit_information`(*fitted_consts: dict[str, float] | None = None*, *error: bool = False*) → None

Set the fitted model information to the model's internal variables.

Parameters

- **fitted_consts** (*dict[str, float]*) – The fitted model constants, defaults to None.
- **error** (*bool*) – Indicates if an error occurred during fitting, defaults to False.

`property symbols: list[str]`

Property to return the list of symbols that are in the expression of the class instance.

A property without an accompanying setter is used to prohibit setting this value.

Returns
List of all symbols in the model expression.

Return type
list[str]

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

m

s

INDEX

Symbols

B

C

D

E

F

G

H

I

M

P

R

S

U

V

W