UNIVERSITY OF SOUTHERN CALIFORNIA
DEPARTMENT OF PHYSICS AND ASTRONOMY

Kojo Boakye-Nimako

PHYS 495: Senior Lab

Data Visualization and Curve Fitting

Advisor: Prof. Eli Levenson-Falk

LOS ANGELES, MAY 2023

Contents

1 Introduction

Skip to Documentation for a traditional documentation. This report outlines the code that was used to create the python module complex_curve_fit_gui. This is meant to be a supplement to the README found on PyPi and on GitHub at complex_curve_fit_gui GitHub. It explains use cases of the code, addresses limitations of the code, and describes the internal setup.

2 Installation

Installation instructions can be found on PyPi at complex_curve_fit_gui. The module can be installed with pip or conda using a python interpreter >= version 3.7. It is recommended to install the most recent version of the module, as versions <= 1.1.5 only work on M1 machines. However versions >=1.1.6 are OS and processor independent.

Another option for installation is to clone the git repository, and create a python script to be ran in the root folder. The code can be found at complex_curve_fit_gui GitHub.

3 Usage

Please find the readme at https://github.com/boakyeni/data-visualization-and-curve-fitting for a complete user guide

- curve_fit_gui

```
from complex_curve_fit_gui import curve_fit_gui
import numpy as np

class Main:
    def __init__(self):
        """
        This function could be any user generate function
        """

        def cos(x, a, b, c, d):
            return a * np.cos(b * x + c) + d

        self.xdata = np.linspace(0, 4, 50)

        # when y data is set to None dummy data generated in backend that complies
        with above xdata array size

        printout = curve_fit_gui(
            cos,
            self.xdata,
            None,
            title="Sin Graph",
            xlabel="y axis",
            ylabel="Imaginary",
            fitline_color="red",
            color="black",
            method="leastsq",
            p0=[3, 3, 3, 3, 3],
            add=[(np.array([1, 2, 3]), np.array([4, 5, 6]), "data2")]
        )
w = Main()
```

To run enter python [filename.py] or python3 [filename.py] or python3 [filename.py]& or python [filename.py]& into the command line. The p0 array for initial parameters is always recommended to get an optimal fit.

3.1 Possible Fitting methods

These are the possible values for keyword parameter method

- "leastsq" : Levenberg-Marquardt $default$
- "least_squares" : Least-Squares minimization, using Trust Region Reflective method
- "basinhopping" : basinhopping
- "ampgo" : Adaptive Memory Programming for Global Optimization
- "brute" : brute force method
- "slsqp" : Sequential Linear Squares Programming

More methods can be found at https://lmfit.github.io/lmfit-py/fitting.html. However the above are the ones that will work without change to the backend.

4 PyQt5 and Widgets

PyQt is a python library for creating gui applications using Qt Toolkit. It is the basis of the front end of this project. This part will explain the widgets, which are the building blocks of a PyQt gui, that were used to construct the front end. A widget is any component of the gui that a user can interact with. In this project the widgets hold inputs, the plots, and the report.

4.1 Param Widget

The Param Widget is a class defined in widgets.py. An instance of the Param widget is initialized for each parameter of the model function that is either supplied in the function call that starts the program, or chosen after the execution of the program. It has two functions read_value and update_value. The user can use the gui to change the value or fix the value. The parameter initialized is a reference to the parameter used in in the Model Object that is used for fit. Programmatically there are multiple ways to access this widget from the the main window or another part of code.

4.2 Model Widget

The Model Widget is the top right portion of gui that allows for editing of the parameters used for the fit. It also contains two buttons INITIAL GUESS and FIT. INITIAL GUESS will plot a line using the chosen model and parameters without trying to fit. FIT performs the fit function on the chosen data and model. Within the source code in the _gui.py file Model Widget is initialized within the init_GUI function and reinitialized under the refresh function when a different model is selected with the SELECT DATA modal. The wrapper functions for the INITIAL GUESS and FIT buttons are located with the Main Window Class.

4.3 Plot Widget

The Plot Widget holds the matplotlib canvas and the tools for interacting with the plot, such as the top toolbar. It also holds a pyqt signal named re_init that communicates with the Main Window when the user decides to switch data or model.

4.4 Plot Canvas

The Plot Canvas is a class that holds the matplotlib plots. It is here where the data is plotted, labels are added and legend created and updated. Changes to visual aspects within the plot happen in this class

4.5 Range Selector

Range Selector is defined in the widgets.py file and allows the user to select the range of of data points to be fitted. Within the gui with the Range Selector button is clicked two draggable vertical lines appear to select the range of the data points. Range Selector is initialized in the Plot Canvas class under the function toggle_rangeselector. When the fit button is pressed a mask is created and compared to original data to determine which values to fit. This happens in the get_range function of the Plot Canvas class. This range also determines the range of the residuals.

4.6 Main Window

The Main window holds together all of the widgets. When the user makes changes within the gui, the Main Window catches these changes through a pyqt signal which then calls the refresh function. This function switches the model and or data by reinitializing the either the Model Widget or the Fitter Data depending on user input.

5 Documentation

5.1 ModelWidget

This code defines a custom Qt widget called ModelWidget, which is used to show and control the fit model parameters.

5.1.1 Functions

- `init(self, model, weightoptions)`: initializes the `ModelWidget` instance with a given `model` and `weightoptions`, and sets up the GUI using the `initGUI` function.
- `initGUI(self, weightoptions)`: sets up the GUI elements of the `ModelWidget`, including a group box, vertical and horizontal boxes, and a combo box for selecting the weight options.
- `disable_weight(self)`: disables the combo box for selecting weight options.
- `enable_weight(self)`: enables the combo box for selecting weight options.
- `get_weight(self)`: returns the currently selected weight option as a string.
- `set_weight(self)`: sets the currently selected weight option in the combo box based on the `weight` attribute of the `model`.
- `read_values(self)`: reads values from the user input and updates the `model`.
- `update_values(self)`: updates the values in the `ModelWidget` to reflect the current values of the `model`.

5.1.2 Attributes

- `model`: the fit model associated with the `ModelWidget`.
- `parviews`: a list of `ParamWidget` instances, which are used to display and edit the parameters of the fit model.
- `WeightLabel`: a `QLabel` instance displaying the text "Weighted Fit:".
- `Yweightcombobox`: a `QComboBox` instance for selecting the weight options.
- `HBox`: a `QHBoxLayout` instance for laying out the `WeightLabel` and `Yweightcombobox` widgets horizontally.
- `VBox`: a `QVBoxLayout` instance for laying out the `parviews` and `HBox` widgets vertically.

5.2 ParamWidget

Qt widget to show and change a fitparameter.

5.2.1 Functions

- `init(self, par)`: initializes the `ParamWidget` instance with a given `par` fit parameter, and sets up the GUI using the `initGUI` function.
- `read_value(self)`: reads the user input (value and fixed) and updates the corresponding values in the `par` fit parameter.
- `update_value(self)`: updates the value displayed in the GUI text box to reflect the current value of the `par` fit parameter.

5.2.2 Attributes

- `par`: the fit parameter associated with the `ParamWidget`.
- `label`: a `QLabel` instance displaying the name of the fit parameter.
- `edit`: a `QLineEdit` instance for editing the value of the fit parameter.
- `check`: a `QCheckBox` instance for toggling whether the fit parameter is fixed or not.

5.3 ReportWidget

5.3.1 Functions

- `init(self)`: initializes the `ReportWidget` instance with an empty text box, and sets the font to the settings specified in `settings`.
- `update_report(self, fitreport)`: updates the text of the text box with the content of the `fitreport` dictionary, recursively printing nested dictionaries.

5.3.2 Attributes

- `setFont(font)`: sets the font of the text box to `font`.
- `setReadOnly(True)`: sets the text box to be read-only.
- `clear()`: clears the text box.
- `insertPlainText(text)`: inserts `text` into the text box without formatting.

5.4 PlotCanvas

5.4.1 Functions

- `__init__(self, fitter, xlabel, ylabel, **kwargs)`: Initializes a new PlotCanvas object with the specified fitter (an instance of a curve fitting class), xlabel and ylabel. Additional keyword arguments can be used to customize the plot. The method creates the figure and axes for the plot and initializes the state variables.

- `plot_data(self, **kwargs)`: Plots the data on the first subplot. Additional keyword arguments can be used to customize the plot.
- `plot_fit(self, **kwargs)`: Plots the fit on the first subplot. Additional keyword arguments can be used to customize the plot.
- `plot_residuals(self, **kwargs)`: Plots the residuals on the second subplot. Additional keyword arguments can be used to customize the plot.
- `plot_additional_data(self)`: Plots additional data on the first subplot. This is useful when comparing multiple sets of data.
- `set_data_plot_default(self, **kwargs)`: Sets the default plot settings for the data plot. Additional keyword arguments can be used to override the defaults.
- `set_fit_plot_default(self, **kwargs)`: Sets the default plot settings for the fit plot. Additional keyword arguments can be used to override the defaults.
- `set_res_plot_default(self, **kwargs)`: Sets the default plot settings for the residuals plot. Additional keyword arguments can be used to override the defaults.
- `update_fit_line(self, **kwargs)`: Updates the fit line with new plot settings. Additional keyword arguments can be used to customize the plot.
- `set_result_box_text(self, text)`: Sets the text in the result box.

5.4.2 Attributes

- `fitter`: An instance of a curve fitting class.
- `xlabel`: The x-axis label.
- `data`: The data to be plotted (contains x, y, and error data).
- `fitline`: The fit line (if available).
- `residuals`: The residuals (if available).
- `complex_residuals`: The complex residuals (if available).
- `initial_guess_line`: The initial guess line.
- `kwargs`: Additional keyword arguments passed to the class.
- `additional_data`: Additional data to be plotted.
- `fitline_kwargs`: Keyword arguments used to customize the fit line.
- `complex`: A boolean indicating whether the data is complex.
- `ax1_title`: The title of the first subplot.
- `range_selector`: A range selector.

- `ax1`: The first subplot (contains the data).
- `ax2`: The second subplot (contains the residuals).
- `data_line`: The line used to plot the data.
- `fitted_line`: The line used to plot the fit.
- `residual_line`: The line used to plot the residuals.
- `zero_res`: A dashed horizontal line used to indicate zero in the residual plot.
- `result_box`: The annotation box that holds the fit results.

5.5 PlotWidget

The code defines a PlotWidget class which is a widget to hold a matplotlib canvas and tools for interacting with the plot.

The class has several attributes, including canvas which is an instance of a PlotCanvas class, toolbar which is a NavigationToolbar instance, and selected_curve, selected_model, and input_func which are all initially set to None.

The class also defines several methods including resizeEvent which emits a signal when the widget is resized, update_plot which updates the plot, _toggle_showselector which toggles the range selector on the plot canvas, and toggle_residual_visibility which toggles the visibility of the residual plot on the canvas.

Lastly, the class defines several functions which are used as models for fitting to the data. These functions include linear, exp_decay, cosine_function, decaying_oscillation, decaying_oscillation2, euler, and complex_function. These functions are stored in a dictionary called function_map, where the keys are string representations of the functions, and the values are the function objects.

5.5.1 Functions

- `__init__(self, fitter, xlabel, ylabel, **kwargs)`: Initializes the PlotWidget object by setting up the layout, creating a PlotCanvas object and NavigationToolbar object, setting up actions for the toolbar, and connecting signals to update the plot when the window is resized.
- `resizeEvent(self, event)`: Emits the resized signal when the window is resized.
- `update_plot(self)`: Calls the update_plot method of the canvas attribute to update the plot.
- `_toggle_showselector(self)`: Toggles the visibility of the rangeselector tool in the plot.
- `toggle_residual_visibility(self)`: Toggles the visibility of the residual plot and updates the x-label of the main plot.
- `switchData`: A QtWidgets.QAction object that allows the user to switch between different sets of data.

5.5.2 Attributes

- `resized`: A QtCore.pyqtSignal that emits when the widget is resized.
- `re_init`: A QtCore.pyqtSignal that emits when the plot is updated.
- `canvas`: A PlotCanvas object that holds the matplotlib canvas and methods for interacting with the plots.
- `toolbar`: A NavigationToolbar object that provides tools for navigating and interacting with the plots.
- `selected_curve`: A string that stores the label of the currently selected curve.
- `initial_complex_curve`: A string that stores the label of the initial complex curve (edge case).
- `selected_model`: A string that stores the name of the currently selected model.
- `input_func`: A string that stores the name of the currently selected input function.
- `data_map`: A dictionary that maps curve labels to their corresponding data.
- `hideRes`: A QtWidgets.QAction object that toggles the visibility of the residual plot.
- `ACshowselector`: A QtWidgets.QAction object that toggles the visibility of the rangeselector tool.
- `function_map`: A dictionary that maps function names to their corresponding mathematical functions.

5.6 RangeSelector

A class that creates a range selector in a plot consisting of two draggable vertical lines.

5.6.1 Functions

- `__init__(self, ax, pos1, pos2)`: Initializes the RangeSelector object by setting up the axes that hold the lines, the initial positions of the lines, and creating the two draggable vertical lines.
- `get_range(self)`: Returns the current positions of the two draggable lines, sorted in ascending order.
- `remove(self)`: Removes the two draggable vertical lines from the plot.

5.6.2 Attributes

- `ax`: An axes object that holds the two draggable vertical lines.
- `pos`: A list of two floats representing the initial positions of the two draggable vertical lines.
- `drag_lines`: A list of two DraggableVLine objects representing the two draggable vertical lines.

5.7 DraggableVLine

5.7.1 Functions

- `init(self, ax, x, linewidth=4, linestyle="--", color="gray")`: Initializes the DraggableVLine object by creating a vertical line with the given parameters and connecting to the required events.
- `get_pos(self)`: Returns the x-coordinate of the vertical line.
- `remove(self)`: Removes the vertical line from the plot.
- `connect(self)`: Connects the DraggableVLine object to the required events.
- `on_press(self, event)`: Callback function for the mouse button press event. Sets the initial position and locks the DraggableVLine object to allow dragging.
- `on_motion(self, event)`: Callback function for the mouse motion event. Moves the DraggableVLine object along with the mouse cursor.
- `on_release(self, event)`: Callback function for the mouse button release event. Releases the DraggableVLine object to allow further dragging.
- `disconnect(self)`: Disconnects the DraggableVLine object from the required events.

5.7.2 Attributes

- `line`: A matplotlib vertical line object.
- `press`: A tuple representing the initial position of the DraggableVLine object and the mouse cursor.
- `cidpress`: A callback ID for the mouse button press event.
- `cidrelease`: A callback ID for the mouse button release event.
- `cidmotion`: A callback ID for the mouse motion event.
- `lock`: A class-level variable to allow locking of the DraggableVLine object to prevent multiple lines from being dragged at once.

5.8 FitParameter

5.8.1 Attributes

- `name: str` - The name of the fit parameter.
- `value: float = 1.0` - The initial value of the fit parameter. Defaults to 1.0.
- `sigma: float = 0.0` - The uncertainty in the fit parameter. Defaults to 0.0.
- `fixed: bool = False` - A boolean indicating whether the fit parameter is fixed or not. If set to True, the parameter value is held fixed during the fit. Defaults to False.

5.9 FitModel

5.9.1 Functions

- `__post_init__(self) -> None`: initializes lmfit model and parameters
- `evaluate(self, x)`: evaluates the model at a given x value
- `get_numfitpars(self)`: returns the number of fit parameters in the model
- `update(self)`: updates the values and fixed/varying status of the parameters in the model based on the values of the corresponding fit parameters in the list

5.9.2 Attributes

- `func`: Any - stores the model
- `weight`: str - specifies the weight of the model
- `fitpars`: List[FitParameter] - list of fit parameters
- `description`: str - optional description of the model

5.10 FitData

5.10.1 Functions

- `post_init(self)`: Private method that initializes the mask attribute.
- `get(self)`: Returns a tuple of x, y, xe, ye arrays with elements corresponding to mask.
- `set_mask(self, xmin, xmax)`: Sets the mask for values of x between xmin and xmax.
- `get_numfitpoints(self)`: Returns the number of x-values between xmin and xmax.

5.10.2 Attributes

- `x: np.array`: x-data
- `y: np.array`: y-data
- `xe: np.array = None`: error-data on x-values
- `ye: np.array = None`: error-data on y-values
- `mask: List[bool] = field(init=False)` Private attribute

5.11 Fitter

5.11.1 Functions

- `_init_data`: a private method that validates the input data and returns an instance of `FitData`
- `_init_model`: a private method that validates the function to be fitted and returns an instance of `FitModel`
- `fit(self)` - This function performs the fitting of the model to the data. It prepares the model and data by getting the user guesses and whether the guess is fixed or not. It checks whether there is at least one free fitparameter and whether the number of degrees of freedom is at least one. It sets the sigma value for the model weight and uses `curve_fit_wrapper()` function to fit the model to the data. It then processes the results, calculates the mean squared error, and creates a report. It returns the optimal values, the covariance matrix, and the lmfit model result.
- `get_curve(self, xmin=None, xmax=None, numpoints=settings["MODEL_NUMPOINTS"])` - This function generates a curve based on the fitted model using the minimum and maximum x-values and the number of points for the curve. If xmin and xmax are not specified, they are set to the minimum and maximum x-values of the data. If the model result is not yet available, it evaluates the model using the lmfit model; otherwise, it uses the lmfit model result.
- `get_model_result(self)` - This function returns the lmfit model result.
- `get_fitcurve(self, xmin=None, xmax=None, numpoints=settings["MODEL_NUMPOINTS"])` - This function returns the fit curve generated by the `get_curve()` function if the fit is valid; otherwise, it returns None.
- `get_residuals(self, check=True)` - This function returns the residuals as $y - f(x)$ if the fit is valid and there is an available lmfit model result; otherwise, it returns None. The residuals are computed using the `evaluate()` function of the model.
- `_degrees_of_freedom(self)` - This function returns the degrees of freedom of the model, which is the number of fitpoints minus the number of fit parameters.
- `_create_report(self)` - This function creates a dictionary containing information about the fit parameters, fit results, and statistics. It calls the `pars_to_dict()` function to create a dictionary of the fit parameters.
- `get_report(self)` - This function returns the report created by the `_create_report()` function.
- `get_weightoptions(self)` - This function returns the weight options for the model, which are determined based on whether there is error data or not.
- `change_data(self, x, y, xe, ye)` - This function changes the data used for the fitting by re-initializing the data object.
- `change_model(self, func, p0, absolute_sigma)` - This function changes the model used for the fitting by re-initializing the model object.

5.11.2 Attributes

- `WEIGHTOPTIONS`: a tuple containing three strings to specify the type of weight (none, relative, absolute)
- `kwargs`: a dictionary containing additional arguments that can be passed to Fitter
- `data`: an instance of `FitData` containing the data to be fitted
- `model`: an instance of `FitModel` containing the function to be fitted and the fit parameters
- `model_result`: the result of the fit obtained from `curve_fit_wrapper`
- `fit_is_valid`: a boolean that is `True` if a valid fit was computed
- `mean_squared_error`: the mean squared error of the fit
- `fitreport`: a dictionary containing the fit report
- `is_complex`: a boolean specifying whether the function to be fitted is complex or not
- `method`: a string specifying the optimization method to be used (default is "leastsq")
- `orig_model`: a copy of the original model for possible later use
- `orig_complex`: a copy of the original value of `is_complex` for possible later use

5.12 curve_fit_wrapper

This is a Python function that wraps the lmfit model and performs curve fitting. The function takes in a model function (`model`) and its arguments (`pargs`) as input, and allows for specifying initial parameter values (`p0`) and whether certain parameters should be fixed (`pF`). The function returns the optimized parameter values (`popt`), the covariance matrix (`cov`), and the lmfit result object (`result`).

Here is a list of the parameters:

- `model`: a function that takes in an independent variable and one or more parameters as input, and returns a dependent variable. This function will be fitted to the data.
- `*pargs`: the dependent and independent variables to be passed to `model`. The first element is assumed to be the independent variable and the remaining elements are the dependent variables.
- `p0` (optional): a list of initial values for the parameters of `model`. If not provided, the initial values are set to 1 for all parameters except the first (independent variable).
- `pF` (optional): a list of Boolean values indicating whether each parameter of `model` should be fixed (`True`) or allowed to vary (`False`). If not provided, all parameters are allowed to vary.
- `method`: the optimization method to be used by lmfit.
- `**kwargs`: additional keyword arguments to be passed to lmfit.

The function first extracts the names of the arguments of `model` using the `inspect` module. It then sets the fixed parameters (`pF`) and initial parameter values (`p0`) to default values if they are not provided. It creates a new function that takes only the free parameters as input by removing the fixed parameters from the argument list of the original function. This new function is then fitted to the data using lmfit. The function then rebuilds the `popt` and `cov` arrays to include the fixed parameters. Finally, the function returns the optimized parameter values (`popt`), the covariance matrix (`cov`), and the lmfit result object (`result`).

5.13 Main Window

5.14 Functions

- `init(self, afitter, xlabel, ylabel, **kwargs)`: initializes the MainWindow object with some default settings and calls `initGUI()` function, also checks if the model is complex and calls the `fit()` function if it is.
- `closeEvent(self, event)`: overrides the `closeEvent` function to properly quit the application.
- `initGUI(self, **kwargs)`: initializes the GUI of the application, creates the widgets for the plot, the model and the report, and organizes them in a layout.
- `showdialog(self, message, icon, info="", details="")`: creates and shows a message dialog with a message and an icon.
- `set_output(self, output)`: sets the output of the application that will be returned when the application is closed.
- `get_output(self)`: gets the current output of the application.
- `evaluate(self)`: updates the model with the current parameter values, computes the model curve, and updates the widgets with the results.
- `fit(self)`: updates the model with the current parameter values, performs the fit, and updates the widgets with the results. If a curve is selected, it only fits that curve.
- `refresh(self, model)`: called upon clicking select within the SELECT DATA modal, updates the data to fitted and the model used in fit

5.15 Input parameters:

- `afitter`: an instance of the curve fitting class that has to be an object of CurveFitter class
- `xlabel`: a string representing the x-axis label of the plot
- `ylabel`: a string representing the y-axis label of the plot
- `**kwargs`: keyword arguments that can be passed to the `initGUI()` function

6 Documented Issues

6.1 Methods

When using a fitting method that is not a variant of least squares, there will be no standard error. This arises as the lmfit model is not producing a covariance matrix. Typically, as in with the least squares methods, the standard error is calculated by taking the diagonal of the covariance matrix and then taking that square root. A possible solution would be adding a different method of calculating the standard error in the _tools.py file under the fit function.

6.2 Complex and Sinusoidal

If the initial guess is far from the best fit, sometimes the fit can be very off. However when you click on SELECT DATA and the SELECT with the popup modal and then click FIT again in the main window, the fit can sometimes correct itself.