lsqfitgui

Release 0.1.0

@ckoerber, @millernb

Nov 11, 2021

CONTENTS:

CHAPTER

ONE

QUICK START

1.1 Install

To install the package and its dependencies, clone the repository and run the following command in the repository root directory

```
pip install --upgrade numpy scipy
pip install [-e] .
```

or equivalently *(let pip do the cloning)*

```
pip install --upgrade numpy scipy
pip install git+https://github.com/ckoerber/lsqfit-gui@master
```

(`lsqfit` and `gvar` require `numpy` and `scipy` as build dependencies. Installing these packages before install `gvar` ensures a smooth installation.)

1.2 Usage

The GUI can be started either:

1. directly from another python script

```
# some_script.py
from lsqfit import nonlinear_fit
from lsqfitgui import run_server
...
fit = nonlinear_fit(data, fcn=fcn, prior=prior)
run_server(fit)
```

1. or over the entry point, which requires a `gvar` pickled fit and an import path to the fit function. As an example, first export the fit running the following script,

```
#other_script.py
import gvar as gv
from lsqfit import nonlinear_fit

def fcn(x, p):
    y = ...
    return y
```

(continues on next page)

(continued from previous page)

```
...

fit = nonlinear_fit(data, fcn=fcn, prior=prior)
gv.dump(fit, "fit.p")
```

and run the GUI with

```
lsqfitgui [--function other_script.py:fcn] fit.p
```

1.3 Advanced usage

See also the `examples/` folder and *documentation*.

CHAPTER

TWO

EXAMPLES

In the `examples/` directory, we provide different examples which can be started by running

```
python example/{script}.py
```

and visiting http://localhost:8000.

2.1 First steps

2.1.1 Launching the dashboard via script

In the most simple case, `lsqfitgui` requires a `lsqfit.nonlinear_fit` object as its only argument.

An example script capable of launching the GUI for a simple quadratic fit model is given by

```
#my_fit.py
import numpy as np

from lsqfit import nonlinear_fit
from gvar import gvar

from lsqfitgui import run_server

x = np.linspace(0, 1, 10)
# first argument is the mean, second the standard deviation
y = gvar(x**2, np.ones(len(x)) * 0.1)

def fcn(x, p):
    return p["a0"] + p["a1"] * x + p["a2"] * x ** 2

prior = {"a0": gvar(0, 2), "a1": gvar(0, 2), "a2": gvar(0, 2)}

fit = nonlinear_fit(data=(x, y), fcn=fcn, prior=prior)

run_server(fit)
```

Running `python my_fit.py` launches the dashboard locally at http://localhost:8000.

This GUI allows for varying prior parameters (mean and standard deviations) and inspecting their effects on the fit.

2.1.2 Launching the dashboard via CLI

Additionally, for a dumped fit object

```
from gvar import dump

...
dump(fit, outputfile="fit.p")
```

it is possible to launch the GUI by running

```
lsqfitgui --function my_fit.py:fcn fit.p
```

2.2 Meta parameters

For some fits, you may not only want to vary the prior parameters but also entirely change the model (or data). This is done by the providing a `fit_setup_function` and respective configuration to the *lsqfitgui.run_server()* method.

```
#my_fit.py
import numpy as np

from lsqfit import nonlinear_fit
from gvar import gvar

from lsqfitgui import run_server

x = np.linspace(0, 1, 10)
y = gvar(x**2, np.ones(len(x)) * 0.1)

# Modify the fit function to be flexible in the number of parameters
def fcn(x, p):
    return sum([p[f"a{n}"] * x **n for n in range(len(p))])

def generate_fit(n_order=2):
    prior = {f"a{n}": gvar(0, 2) for n in range(n_order+1)}
    return nonlinear_fit(data=(x, y), fcn=fcn, prior=prior)

run_server(fit_setup_function=generate_fit)
```

Running this script will launch the same dashboard as before. To update the meta parameter `n_order` in the GUI, we have to provide an initial value

```
fit_setup_kwargs = {"n_order": 2}
```

and parameters to set up a form

```
meta_config = [{"name": "n_order", "type": "number", "min": 1, "max": 10, "step": 1}]
```

so that the actual run command is

```
run_server(
    fit_setup_function=generate_fit,
    fit_setup_kwargs=fit_setup_kwargs,
    meta_config=meta_config
)
```

Note: You can provide more than just one meta setup parameter of arbitrary type. These parameters can have any type allowed to be parsed from HTML widgets. You can find more details about allowed `meta_config` parameters at the `dcc.Input` documentation. Note that the `id` of the widget is automatically provided by the `name` keyword from the `meta_config` dictionary, which must match the `fit_setup_kwargs` keyword and the `generate_fit` keyword.).

2.3 Adding plots

If you want to add plots, you can use the `additional_plots` keyword. The default options must provide a name and a function which takes a fit object and returns a plotly.graph_objects.Figure

```
from plotly.graph_objects import Figure

def plot_fcn(fit):
    fig = Figure()
    ...
    return fig

additional_plots = [{"name": "My plot", "fcn": plot_fcn}]
run_server(
    fit_setup_function=generate_fit,
    fit_setup_kwargs=fit_setup_kwargs,
    meta_config=meta_config,
    additional_plots=additional_plots,
)
```

This method adds plots to the already existing plots container.

For convenience, we have also added methods for creating figures directly from `gvars`. For example, if you want to add a custom function `my_func` to the plot, you can add the lines

```
from lsqfitgui import plot_gvar

def my_func(x, p):
    yy = ...
    return yy

def plot_my_func(fit):
    y_fit = my_func(fit.x, fit.p)
    return plot_gvar(fit.x, y_fit, kind="errorband")

additional_plots = [{"name": "My function", "fcn": plot_log_log}]
```

Additionally, if you have a function that already takes `x` and `p` arguments as the fit function of the `nonlinear_fit` object, you can use

```
from lsqfitgui import wrap_plot_gvar

@wrap_plot_gvar(kind="band")
def plot_my_func_wrapped(x, p):
    return my_func(x, p)

additional_plots = [{
    "name": "My function",
    "fcn": plot_my_func_wrapped,
    "description": "This figure displays my function.", # optional
}]
```

which does effectively the same thing.

2.3.1 Additional kwargs

The `additional_plots` list also allows dictionaries with more keyword arguments. Allowed are

- **name** *(str)*: The name presented in the tabs.
- **fcn** *(Callable[[nonlinear_fit], Figure])*: The function used to generate the plot. Must take a plot and kwargs as an input.
- **kwargs** *(Dict[str, Any])*: A dictionary passed to the provided function.
- **static_plot_gvar** *(Dict[str, Any])*: Static data passed to *`lsqfitgui.plot_gvar()`* added to the same figure (i.e., to also plot data as an comparison).

For example, one setup could be

```
x_data, y_data = ... # some static values to compare against

additional_plots = [{
    "name": "My function",
    "fcn": plot_my_func_wrapped,
    "static_plot_gvar": {
        "x": x_data,
        "y": y_data,
        "kind": "errorbars",
        "scatter_kwargs": {"name": "Data"} # Is passed to go.Scatter
    }
}]
```

2.4 Further usage

See also the `example` directory or API docs.

CHAPTER

THREE

API: LSQFITGUI

3.1 run_server

run_server(*fit=None*, *name='Lsqfit GUI'*, *fit_setup_function=None*, *fit_setup_kwargs=None*, *meta_config=None*, *use_default_content=True*, *get_additional_content=None*, *additional_plots=None*, *run_app=True*, *debug=True*, *host='localhost'*, *port=8000*)

Initialize the GUI and start the dash app.

Requires either a *fit* object or a *fit_setup_function*.

Parameters

- **fit** (*Optional[lsqfit.nonlinear_fit]*) – Non-linear fit object.
- **name** (*str*) – Name of the app displayed as title and browser tab title.
- **fit_setup_function** (*Optional[Callable[[Any], lsqfit.nonlinear_fit]]*) – Function which returns a non-linear fit object. Its keywords are provided by *fit_setup_kwargs*.
- **fit_setup_kwargs** (*Optional[Dict]*) – Initial kwargs which are passed to the *fit_setup_function* for creating the first fit object.
- **meta_config** (*Optional[List[Dict]]*) – Configuration for the fit_setup_kwargs represented in the GUI. These must match dcc.Input arguments.
- **use_default_content** (*Optional[bool]*) – Add default elements like the function documentation and plot tabs to the GUI.
- **get_additional_content** (*Optional[Callable[[lsqfit.nonlinear_fit], dash.html.Base.Base]]*) – Function used to determine dynamic content depending on fit results.
- **additional_plots** (*Optional[Dict[str, Callable]]*) – List of dictionaries specifying plots rendered in the tab element. Must contain at least the *name: str* and *fcn:Callable[[nonlinear_fit], Figure]* items. This populates *FitGUI.plots*. See also the `lsqfitgui.frontend.content.DEFAULT_PLOTS`.
- **run_app** (*bool*) – Call run server on the dash app.
- **debug** (*bool*) – Run the dash app in debug mode. Only used if *run_app=True*.
- **host** (*str*) – The hosting address of the dash app. Only used if *run_app=True*.
- **port** (*int*) – The port of the dash app. Only used if *run_app=True*.

Return type *lsqfitgui.lsqfitgui.FitGUI*

Example

The most basic example just requires a nonlinear_fit object:

```
fit = lsqfit.nonlinear_fit(data, fcn=fcn, prior=prior)
app = run_server(fit)
```

More sophisticated examples, where also meta arguments are used, are:

```
def generate_fit(n_exp=3):
    ...
    return lsqfit.nonlinear_fit(data, fcn=fcn, prior=prior)

fit_setup_kwargs = {"n_exp": 3}
meta_config = [{"name": "n_exp", "type": "number", "min": 1, "max": 10, "step": 1}]

fit_gui = run_server(
    fit_setup_function=generate_fit,
    fit_setup_kwargs=fit_setup_kwargs,
    meta_config=meta_config
)
```

3.2 plot_gvar

plot_gvar(*x*, *y*, *fig=None*, *kind='band'*, *add_log_menu=False*, *scatter_kwargs=None*)

Plot gvars as go.Figures including their uncertainties.

Parameters

- **x** (*Union[numpy.ndarray, Dict[str, numpy.ndarray]]*) – The independent variable. Can be either an array or a dictionary of arrays where keys must match the keys of the dependent variable. If `kind="band"`, tries to interpolate the values.
- **y** (*gvar._bufferdict.BufferDict*) – The dependent variable. If it is a dictionary of gvar arrays, the figure will contain several subfigures.
- **fig** (*Optional[plotly.graph_objs._figure.Figure]*) – Figure to add traces to. If not specified, creates a new figure.
- **kind** (*str*) – Either `"band"` or `"errorbars"`.
- **add_log_menu** (*bool*) – Add a menu to switch from a linear to a log scale. Only available if `y` is not a dictionary.
- **scatter_kwargs** (*Optional[Dict]*) – Keyword arguments passed to `go.Scatter`.

Return type plotly.graph_objs._figure.Figure

3.3 wrap_plot_gvar

wrap_plot_gvar(*kind='band'*, *add_log_menu=False*, *scatter_kwargs=None*)

Wraps functions taking `x` and `p` arguments such that they can be used by the *lsqfitgui.FitGUI.plots* to generate plots of gvars.

Parameters

- **kind** (*str*) – Allowed values: `"band"`, `"errorbar"`. Returned figure will contain error bars or error bands.
- **add_log_menu** (*bool*) – Should the returned figure have a menu allowing to change from regular to log y-axis?
- **scatter_kwargs** (*Optional[Dict]*) – Keyword arguments passed to `go.Scatter()`.

Return type Callable[[lsqfit.nonlinear_fit], plotly.graph_objs._figure.Figure]

Example

The code below presents how to use the wrapper to add new plots to the GUI:

```
def fcn(x, p):
    yy = ...
    return yy

def plot_fcn(fit):
    yy = fcn(fit.x, fit.p)
    return plot_gvar(fit.x, yy, kind="band")

@wrap_plot_gvar(kind="band")
def wrapped_fcn(x, p):
    return fcn(x, p)

gui.plots.append({"name": "Fcn results", "fcn": wrapped_fcn})
```

Both functions, *wrapped_fcn* and *plot_fcn* will produce the same plot when added to the gui.

3.4 FitGUI

class FitGUI(*fit=None*, *fit_setup_function=None*, *fit_setup_kwargs=None*, *meta_config=None*, *use_default_content=True*)

Class which initializes the dashboard.

Initialize the fit gui.

You must either provide a *fit* object or a *fit_setup_function* function to initialize this class. Note that this dose not create a `Dash` app; the app is created by calling *FitGUI.setup_app()* (which is implicitly called by *FitGUI.run_server()*).

Parameters

- **fit** (*Optional[lsqfit.nonlinear_fit]*) – Non-linear fit object.
- **fit_setup_function** (*Optional[Callable]*) – Function which returns a non-linear fit object. Its keywords are provided by *fit_setup_kwargs*.

- **fit_setup_kwargs** (*Optional[Dict]*) – Initial kwargs which are passed to the *fit_setup_function* for creating the first fit object.
- **meta_config** (*Optional[List[Dict]]*) – Configuration for the fit_setup_kwargs represented in the GUI. These must match dcc.Input arguments.
- **use_default_content** (*bool*) – Add default elements like the function documentation and plot tabs to the GUI.

Example

The most basic example just requires a nonlinear_fit object:

```
fit = lsqfit.nonlinear_fit(data, fcn=fcn, prior=prior)
gui = FitGUI(fit)
```

More sophisticated examples, where also meta arguments are used, are:

```
from dash import Dash

def generate_fit(n_exp=3):
    ...
    return lsqfit.nonlinear_fit(data, fcn=fcn, prior=prior)

fit_setup_kwargs = {"n_exp": 3}
meta_config = [{"name": "n_exp", "type": "number", "min": 1, "max": 10, "step": 1}]

gui = FitGUI(
    fit_setup_function=generate_fit,
    fit_setup_kwargs=fit_setup_kwargs,
    meta_config=meta_config
)
fit_gui.run_server(host=host, debug=debug, port=port)
```

---

property app: dash.dash.Dash
Return plotly dash app.

property fit: lsqfit.nonlinear_fit
Return current fit object.

get_additional_content: Callable[[lsqfit.nonlinear_fit], dash.html.Base.Base]
Function used to determine dynamic content depending on fit results.

property initial_fit: lsqfit.nonlinear_fit
Return fit object used to initialize the app.

property layout: dash.html.Base.Base
Return the current layout.

name: str
Name of the app displayed as title and browser tab title.

plots: List[Dict[str, Any]]
List of dictionaries specifying plots rendered in the tab element. Must contain at least the *name: str* and *fcn:Callable[[nonlinear_fit], Figure]* items.

Example

Plot the fit results:

```
def plot_fcn(fit):
    yy = fit.fcn(fit.x, fit.p)
    return plot_gvar(fit.x, yy, kind="band")

gui.plots.append({"name": "Fit results", "fcn": plot_fcn})
```

Allowed keywords are

- **name** *(str)*: The name presented in the tabs.
- **fcn** *(Callable[[nonlinear_fit], Figure])*: The function used to generate the plot. Must take a plot and kwargs as an input.
- **description** *(str)*: Text displayed below figure (can contain latex using).
- **kwargs** *(Dict[str, Any])*: A dictionary passed to the above function.
- **static_plot_gvar** *(Dict[str, Any])*: Static data passed to *plot_gvar()* added to the same figure (i.e., to also plot data as an comparison).

See also the `lsqfitgui.frontend.content.DEFAULT_PLOTS`.

run_server(**args*, ***kwargs*)
Wrapper to self.app.run_server.

setup_app(*app=None*)
Initialize the dash app.

Sets up layout and callbacks and create a `Dash` instance if not provided.

Parameters **app** (*Optional[dash.dash.Dash]*) – The dash app which runs the server. If provided, requires to manually set up style sheets, scripts and assets.

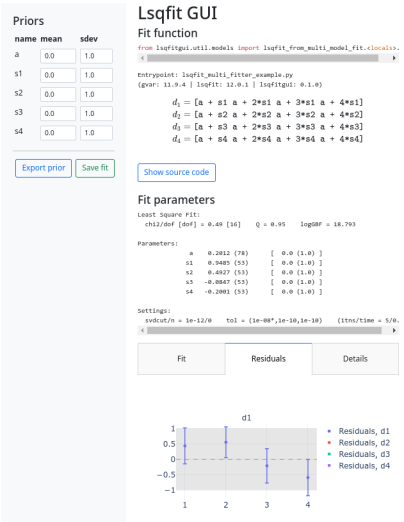
Raises RuntimeError if app already set up.

CHAPTER

FOUR

ABOUT

Graphical user interface for performing Bayesian Inference (bayesian fits) using `lsqfit` and `dash`.



This module aims at making complicated analysis projects

- more intuitive
- accessible to a broader range of people
- easily shareable

and thus effectively increasing transparency.

INDEX