Contents

1 Introduction

The quantum fitter is developed to help academics implement different fit function with ease. The whole project is mainly based on lmfit, which is already convenient to use. The lmfit has two occasionally use classes, one is Model, and the other is Parameter, respectively control model functions and parameter properties.

Before start, please make sure python 3.8+ and the packages listed are correctly installed in your python:

```
numpy - 1.20.1
matplotlib - 3.3.4
scipy - 1.6.2
lmfit - 1.0.2
```

Lower version of python and packages are not guaranteed for running the code, but it's worth a try.

After all the preparation, you can now implement

```
import quantum_fitter as qf
import numpy as np
```

then with x_data and y_data as numpy array

```
t5 = qf.Qfit(x_data, y_data, 'GaussianModel')
t5.do_fit()
t5.pretty_print()
```

to start your journey.

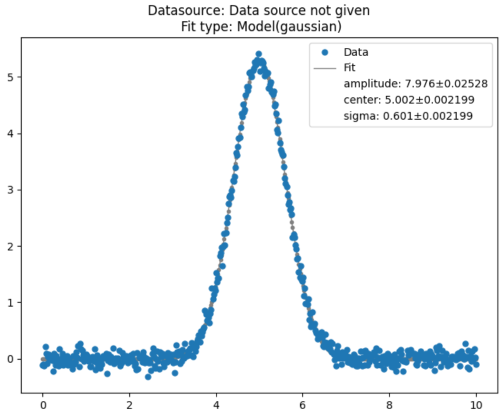

Figure 1: Pretty print for random generate Gaussian distribution

2 Functions

2.1 Qfit

```
class Qfit(data_x, data_y, model=None, params_init=None, method='leastsq')
```

The data have to be numpy array. Model parameter can be either function or build-in model string (see Appendix A). It's also possible to pass in a set or list of model, the Qfit will add the model up.

The params_init can be either dictionary with name and value (Recommends) or list along with sequence from your own function.

Method list can be found in here in the minimize method.

```
@property
def params(init_dict: dict)
```

Set and get initial parameters.

```
def set_params(self, name: str, value: float = None, vary: bool = True,
    minimum=None, maximum=None, expression=None, brute_step=None)
```

Set individual parameters' properties. See lmfit's Parameter to get more info.

```
def add_models(self, model, merge: str = '+')
```

Do models operation on existing functions. Merge can be $+, -, *, /$.

```
def wash(self, method='savgol', **kwargs):
```

To do filter on data (currently only savgol is implemented).

```
def do_fit()
```

Start fitting with current setting.

```
def pretty_print(self, plot_settings: dict=None)
```

Do print. Plot setting contains:

- **x_label**: Define the x label
- **y_label**: Define the y label
- **plot_title**: Define plot title
- **fit_color**: Define fit curve color
- **data_color**: Define data curve color
- **fig_size**: Define figure size
- **x_lim**: Define the x limit of the figure
- **y_lim**
- **show_fig**: Show figure or not. Default is None (which means show). Type in anything will halt this action.
- **return_fig**: Return fig, ax or not. Default is None, type in anything will return the variables.

A plot setting example:

```python
plot_settings = {
    'x_label': 'Time (us)',
    'y_label': 'Voltage (mV)',
    'plot_title': 'datasource',
    'fit_color': 'C4',
    'fig_size': (8, 6),
}
```

```python
def pdf_print(self, file_dir, filename, plot_settings=None):
```

Output PDF to local storage.

```python
def fit_params(name: str = None)
```

Return fit parameter's value or dict.

```python
def err_params(name: str = None):
```

Return fit parameter's stderr value or dict.

```python
def fit_values()
```

Return best fitting y values.

2.2 Others

```python
def params(name: str)
```

Return build-in model's parameters.

A gaussian fit example:

```python
import quantum_fitter as qf
import numpy as np

def gaussian(x, amp, cen, wid):
    return (amp / (np.sqrt(2*np.pi) * wid)) * np.exp(-(x-cen)**2 / (2*wid**2))


# Generate random number from gaussian distribution
x = np.linspace(0, 10, 500)
y = gaussian(x, 8, 5, 0.6) + np.random.randn(500) * 0.1

plot_set = {
    'x_label': 'Time (us)',
    'y_label': 'Voltage (mV)',
    'plot_title': 'datasource',
    'fit_color': 'C4',
    'fig_size': (8, 6),
}

gm = qf.QFit(x, y, 'GaussianModel')
gm.do_fit()
gm.pretty_print(plot_set)
```

A Build-in models

Click here to see all the build-in models.

GaussianModel

LorentzianModel

SplitLorentzianModel

VoigtModel

PseudoVoigtModel

MoffatModel

Pearson7Model

StudentsTModel

BreitWignerModel

LognormalModel

DampedOscillatorModel

DampedHarmonicOscillatorModel

ExponentialGaussianModel

SkewedGaussianModel

SkewedVoigtModel

ThermalDistributionModel

DoniachModel