qef Documentation

Release 0.2.3

Jose Borreguero

Jun 13, 2018

Contents

Contents:

CHAPTER 1

qef

quasielastic fitting

- Free software: MIT license
- Documentation: https://qef.readthedocs.io.

1.1 Features

- TODO

1.2 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 2

Installation

2.1 Stable release

To install qef, run this command in your terminal:

```
$ pip install qef
```

This is the preferred method to install qef, 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 qef can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/jmborr/qef
```

Or download the tarball:

```
$ curl  -OL https://github.com/jmborr/qef/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

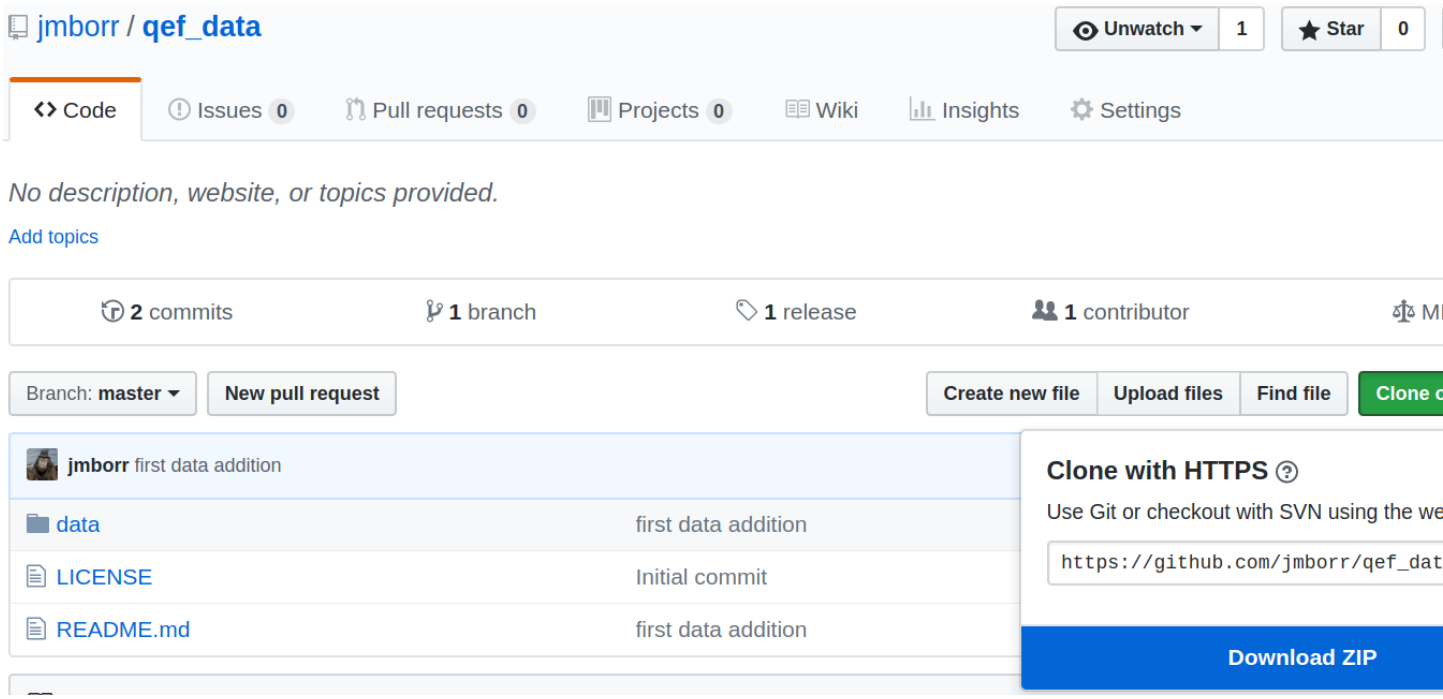
2.3 Testing & Tutorials Data

The external repository *qef_data <https://github.com/jmborr/qef_data>* contains all data files used in testing, examples, and tutorials. There are several ways to obtain this dataset:

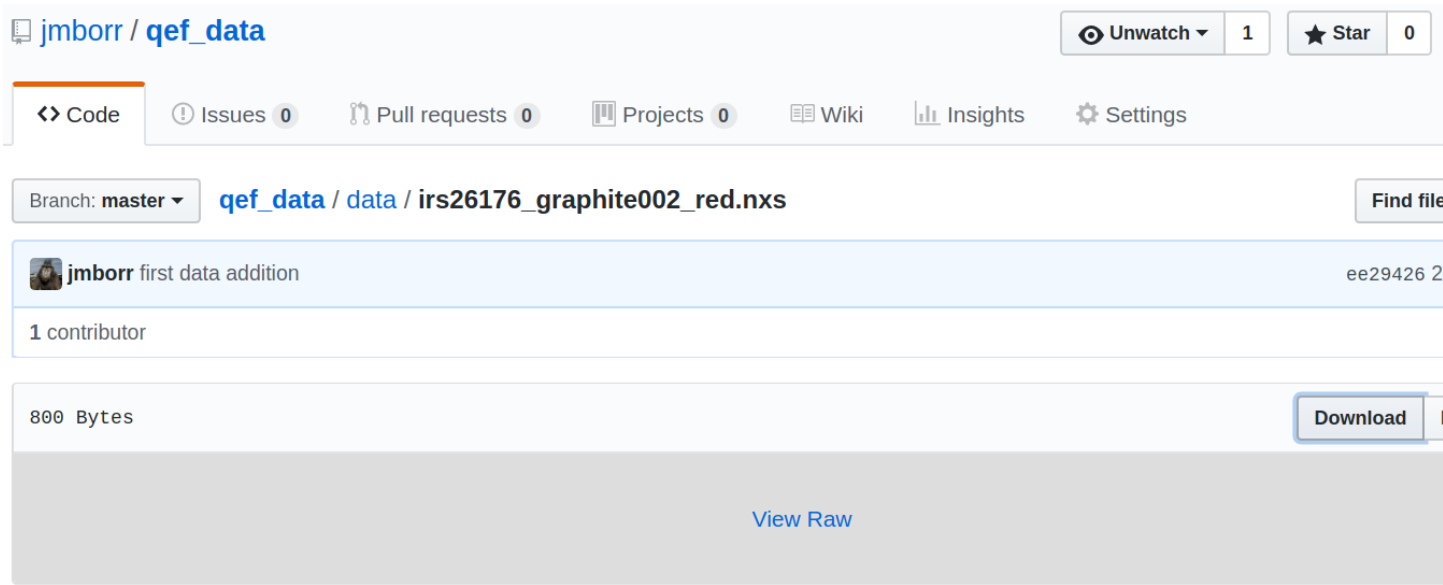
1. Clone the repository with a git command in a terminal:

```
cd some/directory/
git clone https://github.com/jmborr/qef_data.git
```

2. Download all data files as a zip file using GitHub's web interface:



3. Download individual files using GitHub's web interface by browsing to the file, then click in Download button



CHAPTER 3

Usage

To use qef in a project:

```
import qef
```

CHAPTER 4

Modules

4.1 Models

4.1.1 DeltaDiracModel

class qef.models.deltadirac.**DeltaDiracModel**(*independent_vars=['x'], prefix='', missing=None, name=None, **kwargs*)

Bases: lmfit.model.Model

A function that is zero everywhere except for the x-value closest to the center parameter.

At value-closest-to-center, the model evaluates to the amplitude parameter divided by the x-spacing. This last division is necessary to preserve normalization with integrating the function over the X-axis

Fitting parameters:

- integrated intensity `amplitude` A
- position of the peak `center` E_0

guess(*y*, *x=None*, ***kwargs*)

Guess starting values for the parameters of a model.

Parameters

- **y** (ndarray) – Intensities
- **x** (ndarray) – energy values
- **kwargs** (*dict*) – additional optional arguments, passed to model function.

Returns parameters with guessed values

Return type Parameters

qef.models.deltadirac.**delta_dirac**(*x*, *amplitude=1.0*, *center=0.0*)

function is zero except for the x-value closest to center.

At value-closest-to-center, the function evaluates to the amplitude divided by the x-spacing.

Parameters

- **x :class:‘~numpy:numpy.ndarray‘** – domain of the function, energy
- **amplitude** (*float*) – Integrated intensity of the curve
- **center** (*float*) – position of the peak

4.1.2 Resolution models

class qef.models.resolution.**TabulatedResolutionModel**(*xs*, *ys*, **args*, ***kwargs*)

Bases: *qef.models.tabulatedmodel.TabulatedModel*

Interpolator of resolution data with no fit parameters

Parameters

- **xs** (ndarray) – given domain of the function, energy
- **ys** (ndarray) – given domain of the function, intensity

4.1.3 StretchedExponentialFTModel – Fourier transform of the stretched exponential

class qef.models.strexpft.**StretchedExponentialFTModel**(*independent_vars=['x']*, *prefix=''*, *missing=None*, *name=None*, ***kwargs*)

Bases: lmfit.model.Model

Fourier transform of the symmetrized stretched exponential

$$S(E) = A\int_{-\infty}^{\infty} dt/h e^{-i2\pi(E-E_0)t/h} e^{|\frac{x}{\tau}|^\beta}$$

Normalization and maximum at $E = E_0$:

$$\int_{-\infty}^{\infty} dE S(E) = Amax(S) = A\frac{\tau}{\beta}\Gamma(\beta^{-1})$$

Uses scipy.fftpack.fft for the Fourier transform

Fitting parameters:

- integrated intensity `amplitude` A
- position of the peak `center` E_0
- nominal relaxation time `tau` ` τ
- stretching exponent `beta` β

If the time unit is picoseconds, then the reciprocal energy unit is mili-eV

guess(*y*, *x=None*, ***kwargs*)

Guess starting values for the parameters of a model.

Parameters

- **y** (ndarray) – Intensities
- **x** (ndarray) – energy values
- **kwargs** (*dict*) – additional optional arguments, passed to model function.

Returns parameters with guessed values

Return type Parameters

qef.models.strexpft.**strexpft**(*x*, *amplitude=1.0*, *center=0.0*, *tau=10.0*, *beta=1.0*)
Fourier transform of the symmetrized stretched exponential

$$S(E) = A \int_{-\infty}^{\infty} dt/h e^{-i2\pi(E-E_0)t/h} e^{|\frac{x}{\tau}|^\beta}$$

Normalization and maximum at $E = E_0$:

$$\int_{-\infty}^{\infty} dES(E) = A$$

$$max(S) = A\frac{\tau}{\beta}\Gamma(\beta^{-1})$$

Uses fft() for the Fourier transform

Parameters

- **x** (ndarray) – domain of the function, energy
- **amplitude** (*float*) – Integrated intensity of the curve
- **center** (*float*) – position of the peak
- **tau** (*float*) – relaxation time.
- **beta** (*float*) – stretching exponent
- **If the time units are picoseconds, then the energy units are mili-eV.**

Returns **values** – function over the domain

Return type ndarray

4.1.4 TabulatedModel – linear interpolator for a numerical table of intensity values

class qef.models.tabulatedmodel.**TabulatedModel**(*xs*, *ys*, **args*, ***kwargs*)
Bases: lmfit.model.Model

fitting the tabulated Model to some arbitrary points

Parameters

- **xs** (ndarray) – given domain of the function, energy
- **ys** (ndarray) – given domain of the function, intensity
- **Fitting parameters** –
  - rescaling factor amplitude
  - shift along the X-axis center

guess(*data*, *x*, ***kwargs*)
Guess starting values for the parameters of a model.

Parameters

- **data** (ndarray) – data to be fitted
- **x** (ndarray) – energy domain where the interpolation required
- **kwargs** (*dict*) – additional optional arguments, passed to model function.

Returns parameters with guessed values

Return type Parameters

4.1.5 TeixeiraWater – jump-diffusion model for water

class qef.models.teixeira.**TeixeiraWaterModel**(*independent_vars=['x'], q=0.0, prefix='', missing=None, name=None, **kwargs*)

Bases: lmfit.model.Model

This fitting function models the dynamic structure factor for a particle undergoing jump diffusion.

10. Teixeira, M.-C. Bellissent-Funel, S. H. Chen, and A. J. Dianoux. Phys. Rev. A, 31:1913-1917

$$S(Q,E) = \frac{A}{\pi} \cdot \frac{\Gamma}{\Gamma^2 + (E - E_0)^2}$$

$$\Gamma = \frac{\hbar \cdot D \cdot Q^2}{1 + D \cdot Q^2 \cdot \tau}$$

Γ is the HWHM of the lorentzian curve.

At 298K and 1atm, water has $D = 2.3010^{-5} cm^2/s$ and $\tau = 1.25ps$.

A jump length l can be associated: $l^2 = 2N \cdot D \cdot \tau$, where N is the dimensionality of the diffusion problem ($N = 3$ for diffusion in a volume).

Fitting parameters:

- integrated intensity `amplitude` A
- position of the peak `center` E_0
- residence time `center` τ
- diffusion coefficient `dcf` D

Attributes:

- Momentum transfer `q`

fwhm_expr
Constraint expression for FWHM

guess(*y*, *x=None*, ***kwargs*)
Guess starting values for the parameters of a model.

Parameters

- **y** (ndarray) – Intensities
- **x** (ndarray) – energy values
- **kwargs** (*dict*) – additional optional arguments, passed to model function.

Returns parameters with guessed values

Return type Parameters

height_expr
Constraint expression for maximum peak height.

qef.models.teixeira.**teixeira_water**(*x*, *amplitude=1.0*, *center=1.0*, *tau=1.0*, *dcf=1.0*, *q=1.0*)

4.2 Operators

4.2.1 Convolution operator

class qef.operators.convolve.**Convolve**(*resolution*, *model*, ***kws*)
Bases: lmfit.model.CompositeModel

Convolution between model and resolution.

It is assumed that the resolution FWHM is energy independent. Non-symmetric energy ranges are allowed (when the range of negative values is different than that of positive values).

The convolution requires multiplication by the X-spacing to preserve normalization

eval(*params=None*, ***kwargs*)
TODO: docstring in public method.

qef.operators.convolve.**convolve**(*model*, *resolution*)
Convolution of resolution with model data.

It is assumed that the resolution FWHM is energy independent. We multiply by spacing dx of independent variable x.

$$(model \otimes resolution)[n] = dx * \sum_m model[m] * resolution[m - n]$$

Parameters

- **model** (*numpy.ndarray*) – model data
- **resolution** (*numpy.ndarray*) – resolution data

Returns

Return type numpy.ndarray

4.3 Input / Ouptut

4.3.1 Data Loaders

qef.io.loaders.**histogram_to_point_data**(*values*)
Transform histogram(s) to point data

Parameters **values** (ndarray) – Array with histogram data

Returns Array with point data

Return type ndarray

qef.io.loaders.**load_dave**(*file_name*, *to_meV=True*)

Parameters

- **file_name** (*str*) – Path to file
- **to_meV** (*bool*) – Convert energies from micro-eV to mili-eV

Returns keys are q(momentum transfer), x(energy or time), y(intensities), and errors(e)

Return type dict

qef.io.loaders.**load_nexus**(*file_name*)

Parameters **file_name** (*str*) – Absolute path to file

qef.io.loaders.**load_nexus_processed**(*file_name*)
Load data from a Mantid Nexus processed file

Parameters **file_name** (*str*) – Path to file

Returns keys are q(momentum transfer), x(energy or time), y(intensities), and errors(e)

Return type dict

qef.io.loaders.**search_attribute**(*handle*, *name*, *ignore_case=False*)
Find HDF5 entries containing a particular attribute

Parameters

- **handle** – Root entry from which to start the search
- **name** (*str*) – Regular expression pattern to search in attributes' names

Returns All entries with a matching attribute name. Each entry of the form (HDF5-instance, (attribute-key, attribute-vale))

Return type list

4.4 Widgets

4.4.1 Parameter

class qef.widgets.parameter.**ParameterCallbacksMixin**
Bases: object

Implement relationships between the different components of an ipywidget exposing all or some of the parameter attributes

The methods in this Mixin expects attribute `facade`, a dictionary whose keys coincide with tuple *widget_names* and whose values are either `None` or references to ipywidgets. Attribute `facade` can be created with function *add_widget_facade()*.

expr_value_change(*change*)
enable/disable `min`, `max`, and `value`

inf = inf
Representation of infinity value

initialize_callbacks()
Register callbacks to sync widget components

max_value_change(*change*)
Notify other widgets if `min` changes.

0. Reject change if `max` becomes smaller than `min`
1. Uncheck `nomax` if new value is entered in `max`
2. Update `value.value` if it becomes bigger than `max.value`

min_value_change(*change*)
Notify other widgets if `min` changes.

0. Reject change if `min` becomes bigger than `max`
1. Uncheck `nomin` if new value is entered in `min`

2. Update `value.value` if it becomes smaller than `min.value`

nomax_value_change(*change*)
Set `max` to ∞ if `nomax` is checked

nomin_value_change(*change*)
Set `min` to $-\infty$ if `nomin` is checked

validate_facade()
Ascertain that keys of `facade` attribute are contained in *widget_names*

value_value_change(*change*)
Validate `value` is within bounds. Otherwise set `value` as the closest bound value

vary_value_change(*change*)
enable/disable editing of `min`, `max`, `value`, and `expr`

widget_names = ('nomin', 'min', 'value', 'nomax', 'max', 'vary', 'expr')

class qef.widgets.parameter.**ParameterWidget**(*show_header=True*)
Bases: ipywidgets.widgets.widget_box.Box

One possible representation of a fitting parameter. Inherits from ipywidgets.widgets.widget_box.Box

Parameters **show_header** (*Bool*) – Hide or show names of the widget components *min*, *value*,...

class qef.widgets.parameter.**ParameterWithTraits**(*name=None*, *value=None*, *vary=True*, *min=-inf*, *max=inf*, *expr=None*, *brute_step=None*, *user_data=None*)
Bases: lmfit.parameter.Parameter, traitlets.traitlets.HasTraits

Wrapper of `Parameter` with `TraitType` allows synchronization with ipywidgets

Same signature for initialization as that of `Parameter`.

Parameters

- **name** (*str, optional*) – Name of the Parameter.
- **value** (*float, optional*) – Numerical Parameter value.
- **vary** (*bool, optional*) – Whether the Parameter is varied during a fit (default is True).
- **min** (*float, optional*) – Lower bound for value (default is *-numpy.inf*, no lower bound).
- **max** (*float, optional*) – Upper bound for value (default is *numpy.inf*, no upper bound).
- **expr** (*str, optional*) – Mathematical expression used to constrain the value during the fit.
- **brute_step** (*float, optional*) – Step size for grid points in the *brute* method.
- **user_data** (*optional*) – User-definable extra attribute used for a Parameter.

classmethod attr_to_trait(*attr*)
From `Parameter` attribute name to `TraitType` name

classmethod feature_to_trait(*feature*)
From `Parameter` feature name to `TraitType` name

link_widget(*widget*, *mapping=None*)
Link the value of a single ipywidget to one trait, or the values of the element widgets of a composite ipywidget to different traits. The specific traits can be specified with the `mapping` argument.

Parameters

- **widget** (ipywidgets.widgets.widget.Widget)

- **mapping** (*str, dict, or None*) – if *str*, `mapping` denotes the widget name to be associated with the widget. If *dict*, then `mapping` values are attribute names of *widget*, referencing the simple ipywidgets to be associated to standard *widget_names*. The widget names are the keys of `mapping`. If `None`, an inspection of *widget* attributes will be performed, looking for names that coincide with standard *widget_names*. If the inspection is unsuccessful, the widget will be associated with the standard widget name 'value' to represent the values taken by the fitting parameter.

param_attrs = ('_val', 'min', 'max', 'vary', '_expr')
`Parameter` attribute names

param_features = ('value', 'min', 'max', 'vary', 'expr')
`Parameter` feature names

texpr
`Unicode` trailet wrapping `Parameter` attribute `_expr`

tmax
`Float` trailet wrapping `Parameter` attribute `min`

tmin
`Float` trailet wrapping `Parameter` attribute `_val`

trait_names = ('tvalue', 'tmin', 'tmax', 'tvary', 'texpr')
`TraitType` instances in sync with `Parameter` attributes

classmethod trait_to_attr(*name*)
From `TraitType` name to `Parameter` attribute name

tvalue
`Float` trailet wrapping `Parameter` attribute `value`

tvary
`Bool` trailet wrapping `Parameter` attribute `vary`

qef.widgets.parameter.**add_widget_callbacks**(*widget*, *mapping=None*)
Extend the widget's type with *ParameterCallbacksMixin*

Parameters

- **widget** (ipywidgets.widgets.widget.Widget)
- **mapping** (*str, dict, or None*) – if *str*, `mapping` denotes the widget name to be associated with the widget. If *dict*, then `mapping` values are attribute names of *widget*, referencing the simple ipywidgets to be associated to standard *widget_names*. The widget names are the keys of `mapping`. If `None`, an inspection of *widget* attributes will be performed, looking for names that coincide with standard *widget_names*. If the inspection is unsuccessful, the widget will be associated with the standard widget name 'value' to represent the values taken by the fitting parameter.

qef.widgets.parameter.**add_widget_facade**(*widget*, *mapping=None*)
Create `facade` dictionary where keys are standard *widget_names* and whose values are simple ipywidgets that control the fitting parameter attributes denoted by the standard *widget_names*. This dictionary is added to the input widget as an attribute.

Parameters

- **widget** (ipywidgets.widgets.widget.Widget)
- **mapping** (*str, dict, or None*) – if *str*, mapping denotes the widget name to be associated with the widget. If *dict*, then *mapping* values are attribute names of *widget*, referencing the simple ipywidgets to be associated to standard widget names. The widget names are the

keys of *mapping*. If `None`, an inspection of *widget* attributes will be performed, looking for names that coincide with standard widget names. If the inspection is unsuccessful, the widget will be associated with the standard widget name 'value' to represent the values taken by the fitting parameter.

Returns **widget** – Reference to input widget

Return type `Widget`

`qef.widgets.parameter.`**`create_facade`**(*widget*, *mapping=None*)

Create `facade` dictionary where keys are standard *`widget_names`* and whose values are simple ipywidgets that control the fitting parameter attributes denoted by the standard *`widget_names`*.

Parameters

- **widget** (ipywidgets.widgets.widget.Widget)
- **mapping** (*str, dict, or None*) – if *str*, mapping denotes the widget name to be associated with the widget. If *dict*, then *mapping* values are attribute names of *widget*, referencing the simple ipywidgets to be associated to standard widget names. The widget names are the keys of *mapping*. If `None`, an inspection of *widget* attributes will be performed, looking for names that coincide with standard widget names. If the inspection is unsuccessful, the widget will be associated with the standard widget name 'value' to represent the values taken by the fitting parameter.

Returns **facade**

Return type dict

CHAPTER 5

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:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/jmborr/qef/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.

5.1.2 Fix Bugs

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

5.1.3 Implement Features

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

5.1.4 Write Documentation

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

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/jmborr/qef/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 :)

5.2 Get Started!

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

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

```
$ git clone git@github.com:your_name_here/qef.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 qef
$ cd qef/
$ 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 qef 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.

5.3 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 2.6, 2.7, 3.3, 3.4 and 3.5, and for PyPy. Check https://travis-ci.org/jmborr/qef/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ py.test tests.test_strexpft
```

CHAPTER 6

Credits

6.1 Development Lead

- Jose Borreguero <borreguero@gmail.com>

6.2 Contributors

None yet. Why not be the first?

CHAPTER 7

History

7.1 0.3.0 ()

- Subscription manager of ipywidgets to a ParameterWithTraits (PR #60)
- ipywidget to represent a parameter (PR #59)
- Endow lmfit.Parameter instances with traitlets (PR #57)
- Load DAVE files (PR #51)
- Docs for data repository (PR #45)

7.2 0.2.3 (2018-04-10)

- Include only qef directory in release

7.3 0.2.2 (2018-04-10)

- Exclude tests directory from release

7.4 0.2.1 (2018-04-10)

- Include subdirectories of qef in release

7.5 0.2.0 (2018-04-09)

- Notebook for global fitting (PR #40)

- Load Mantid Nexus data (PR #38)
- Tabulated resolution model (PR #43)

7.6 0.1.0 (2017-12-13)

- First release on PyPI.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Donwload the documentation

Python Module Index

q

Index

Q

S

T

V

W