squmfit Documentation

Release 0.1

Ben Gamari

Jun 26, 2018

Contents

This is the documentation for squmfit (pronounced "squim-fit"), a Python library for convenient non-linear least-squares fitting of multiple analytical models to data. This library is similar to the excellent lmfit package but is a fresh implementation designed with support for global fitting of multiple curves with parameter tying.

You may want to start at *Getting started with squmfit* or go directly to the `squmfit` module documentation.

Contents:

CHAPTER 1

Getting started with squmfit

squmfit is a general-purpose library for non-linear least-squares fitting of curves.

This documentation needs to be written. Hey you, reader! May you could write it! Pull requests accepted.

CHAPTER 2

API Reference

2.1 Building models

2.1.1 The `Expr` class

class squmfit.**Expr**

An expression capable of taking parameters from a packed parameter vector. All of the usual Python arithmetic operations are supported as well as a good fraction of the Numpy `ufuncs`. Note, however, that the ufuncs' `out` parameter is not supported.

evaluate(*params*, ***user_args*)

Evaluate the model with the given parameter values.

Parameters

- **params** (*array, shape = [n_params]*) – Packed parameters vector
- **user_args** (*kwargs*) – Keyword arguments from user

gradient

The gradient of the expression with respect to the fitted parameters or raise a `ValueError` if not applicable.

Return type *Expr* evaluating to array of shape `(Nparams,)`

parameters()

Return the set of fitted parameters used by this expression.

Return type `set` of `FittedParam`.

map(*f*)

Lift a function into an *Expr*.

Parameters **f** (Function of type *A* -> *B*) – The function to lift.

Returns Given an *Expr* evaulating to a value of type *A* and a function *A* -> *B*, returns an *Expr* of type *B*.

negative(*out=None*)

absolute(*out=None*)

rint(*out=None*)

sign(*out=None*)

conj(*out=None*)

exp(*out=None*)

log(*out=None*)

log2(*out=None*)

log10(*out=None*)

expm1(*out=None*)

log1p(*out=None*)

sqrt(*out=None*)

square(*out=None*)

reciprocal(*out=None*)

sin(*out=None*)

cos(*out=None*)

tan(*out=None*)

arcsin(*out=None*)

arccos(*out=None*)

arctan(*out=None*)

sinh(*out=None*)

cosh(*out=None*)

tanh(*out=None*)

arcsinh(*out=None*)

arccosh(*out=None*)

arctanh(*out=None*)

deg2rad(*out=None*)

rad2deg(*out=None*)

floor(*out=None*)

ceil(*out=None*)

trunc(*out=None*)

2.1.2 Evaluating Exprs

squmfit.**model**(*func*)

Transforms a function to ensure that all of its parameters are evaluated. Calls to the transformed function will result in an *Expr* when any of its parameters are Exprs.

2.1.3 The `FittedParam` class

class squmfit.parameter.**FittedParam**(*param_set*, *idx*, *name=None*, *initial=None*)
A parameter to be fitted to data.

evaluate(*params*, ***user_args*)
Evaluate the model with the given parameter values.

Parameters

- **params** (*array, shape = [n_params]*) – Packed parameters vector
- **user_args** (*kwargs*) – Keyword arguments from user

gradient()
The gradient of the expression with respect to the fitted parameters or raise a `ValueError` if not applicable.

Return type `Expr` evaluating to array of shape `(Nparams,)`

parameters()
Return the set of fitted parameters used by this expression.

Return type `set` of *FittedParam*.

2.2 Performing fits

2.2.1 The `Fit` class

class squmfit.**Fit**
This represents a fit configuration.

add_curve(*name*, *model*, *data*, *weights=None*, ***user_args*)
Add a curve to the fit.

Parameters

- **name** (*str*) – A unique name for the curve.
- **model** (*Expr*) – The analytical model which will be fit to the curve.
- **data** (*array, shape = [n_samples]*) – An array of the dependent values for each sample.
- **weights** (*array, shape = [n_samples], optional*) – An array of the weight of each sample. Often this is $1/\sigma$ where *sigma* is the standard deviation of the dependent value. If not given uniform weights are used.
- **user_args** (*kwargs*) – Keyword arguments passed to the model during evaluation.

eval(*params*, ***user_args*)
Evaluate the model against a dictionary of parameters

Parameters

- **params** (*dict, param_name -> float*) – A dictionary of parameters values.
- **user_args** (*kwargs*) – Keyword arguments passed to the model.

Return type dict, curve_name -> array, shape = [n_samples]

Returns The value of the model evaluated with the given parameters.

eval_packed(*params*, ***user_args*)
Evaluate the model against packed parameters values

Parameters

- **params** (*array, shape = [n_params]*) – A packed array of parameters.
- **user_args** (*kwargs*) – Keyword arguments passed to the model.

Return type dict, curve_name -> array, shape = [n_samples]

fit(*params0=None*, *report_progress=None*, ***user_args*)
Carry out the fit.

Parameters

- **params0** (*dict, param_name -> float*) – The initial parameter values.
- **report_progress** (*float or None*) – Period with which to report on fit progress (in seconds).
- **user_args** (*kwargs*) – Keyword arguments passed to the model.

Return type A `FitResults` object.

param(*name=None*, *initial=None*)
Create a new parameter.

Parameters

- **name** (*str, optional*) – The name of the new parameter. A name will be generated if this is omitted.
- **initial** (*float, optional*) – The initial value of the parameter. If not provided then a value must be provided when *fit()* is called.

Return type *parameter.FittedParam*

residuals(*params*, *weighted=True*, ***user_args*)
Evaluate the weighted model residuals against a dictionary of parameters values. See *eval()* for parameter descriptions.

residuals_packed(*params*, *weighted=True*, ***user_args*)
Compute the weighed residuals against packed paramater values. See *eval_packed()* for parameter descriptions.

2.2.2 The `FitResult` class

class squmfit.**FitResult**(*fit*, *params*, *covar_p=None*, *initial_result=None*)
This embodies a set of parameter values, possibly originating from a fit.

correl
The correlation coefficient between parameters or `None` if not available.

Return type dict, param_name -> param_name -> float or `None`

covar
The covariances between parameters, or `None` if not available (which may either be due to numerical trouble in calculation or simply not being provided when the *FitResult* was created).

Return type dict of dicts, param_name -> param_name -> float or `None`

curves
Results for particular curves.

Return type dict, curve_name -> *CurveResult*

eval(*expr*)
Evaluate the given *Expr* against the fitted parameter values.

Parameters **expr** (*Expr*) – The expression to evaluate.

Returns The evaluation of `expr`

fit
The *Fit* to which these parameter apply.

Return type *Fit*

initial
The *FitResult* used as the initial parameters for the *Fit* from which this result originated.

Return type *FitResult*

params
The fitted parameter values

Return type array, shape = [n_params]

stderr
The standard error of the parameter estimate or `None` if not available.

Return type dict, param_name -> float or `None`

total_chi_sqr
The sum of the individual curves' chi-squared values. This can be useful as a global goodness-of-fit metric.

Return type float

2.2.3 The `CurveResult` class

class `squmfit`.**CurveResult**(*fit_result*, *curve*)
This embodies a set of parameter values describing the goodness-of-fit with respect to a given curve.

chi_sqr
Chi-squared of the fit.

Return type float

curve
The curve described by this result

Return type `Curve`

degrees_of_freedom
The number of degrees-of-freedom of the fit. This is defined as the number of data points in the curve minus the number of free parameters in the fitting model.

Return type int

eval(***user_args*)
Evaluate the curve's model with overridden user arguments.

fit
The model evaluated with the parameter values.

Return type ndarray

fit_result
The *FitResult* this *CurveResult* was derived from.

Return type *FitResult*

reduced_chi_sqr
Reduced chi-squared of the fit. Namely, `chi_sqr / degrees_of_freedom`.

Return type float

residuals
The weighted residuals of the fit.

Return type ndarray

2.3 Rendering fit results

2.3.1 Text

`squmfit.pretty.`**`html_fit_result`**(*result*, *min_corr=0.5*)
Pretty-print a fit result as an HTML-formatted string

Parameters **min_corr** (*float*) – Don't show correlations smaller than this threshold

Return type str

`squmfit.pretty.`**`ipynb_fit_result`**(*result*, *min_corr=0.5*)
Produce a `HTML` summary of a `FitResult` suitable for rendering by IPython notebook.

Parameters **result** (`FitResult`) – The result to summarize.

`squmfit.pretty.`**`markdown_fit_result`**(*result*, *min_corr=0.5*)
Pretty-print a fit result as a Markdown-formatted string

Parameters **min_corr** (*float*) – Don't show correlations smaller than this threshold

Return type str

2.3.2 Plotting

`squmfit.plot.`**`plot_curve`**(*x*, *result*, *range=None*, *axes=None*, *npts=300*, *xscale='linear'*, *errorbars=True*, *label=None*, *color=None*, ***kwargs*)
Plot the result of a fit against a curve.

Parameters

- **x** – A key found in the *user_args* for the curve.
- **range** (*tuple of two floats*) – The limits of the X axis.
- **npts** (*int*) – Number of interpolation points for fit.
- **xscale** (*'linear', 'log', or 'symlog'*) – The scale to use for the X axis (see `pl.xscale()`).
- **label** – The label for the curve. Defaults to the curve's name.

`squmfit.plot.`**`plot_curve_residuals`**(*x*, *result*, *range=None*, *axes=None*, *xscale='linear'*, *abs_residuals=False*, *residual_range=None*, ***kwargs*)
Plot the residuals of a curve.

Parameters

- **x** – A key found in the *user_args* for the curve.

- **range** (`(xmin, xmax)`, optional) – The limits of the X axis
- **xscale** (*`'linear', 'log', or 'symlog'`*) – The scale to use for the X axis (see `pl.xscale()`).
- **axes** (`pl.Axes`) – Axes to plot on.
- **abs_residuals** (*`bool`*) – Whether to plot weighted (relative) or unweighted (absolute residuals).
- **residual_range** (`(ymin, ymax)`, optional) – Y range of residual plot
- **kwargs** – Keyword arguments to be passed to `Axes.scatter()`.

`squmfit.plot.`**plot_fit**(*x*, *result*, *range=None*, *xscale='linear'*, *errorbars=True*, *fig=None*, *with_residuals=True*, *abs_residuals=False*, *residual_range=None*, *legend_kwargs={}*, *marker=None*)

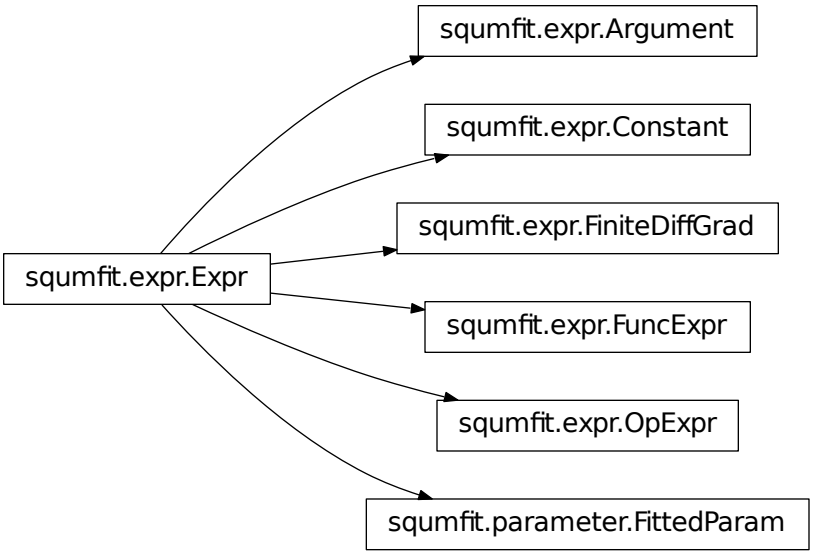
Plot the result of a fit.

Parameters

- **x** – A key found in *user_args* for each curve in the fit
- **range** (`(xmin, xmax)`, optional) – Range of abscissa
- **xscale** (*`'linear', 'log', or 'symlog'`*) – The scale to use for the X axis (see `pl.xscale()`).
- **errorbars** (*`bool`*) – Plot errorbars on points.
- **fig** (`pl.Figure`, optional) – The figure on which to plot.
- **with_residuals** (*`bool`*) – Plot residuals alongside fit
- **abs_residuals** (*`bool`*) – Whether to plot weighted (relative) or unweighted (absolute residuals).
- **residual_range** (`(ymin, ymax)`, optional) – Y range of residual plot
- **marker** – Set the marker with which to mark points
- **legend_kwargs** – Keyword arguments passed to `pl.legend()`.

2.4 Helper classes

These classes represent the various nodes of the expression tree. It is generally not necessary to use these explicitly but they have been included for completeness.



2.4.1 The `Constant` class

class squmfit.expr.**Constant**(*value*)
An `Expr` containing a constant value

__init__(*value*)
Create a constant-valued `Expr`

Parameters **value** – The value

evaluate(*params*, ***user_args*)
Evaluate the model with the given parameter values.

Parameters

- **params** (*array, shape = [n_params]*) – Packed parameters vector
- **user_args** (*kwargs*) – Keyword arguments from user

parameters()
Return the set of fitted parameters used by this expression.

Return type `set` of `FittedParam`.

2.4.2 The `FiniteDiffGrad` class

class squmfit.expr.**FiniteDiffGrad**(*expr*)
The gradient of an expression computed by finite difference

__init__(*expr*)
Create an expression representing the gradient of an expression.

Parameters **expr** (Expr) – The expression to differentiate.

evaluate(*params*, ***user_args*)
Evaluate the model with the given parameter values.

Parameters

- **params** (*array, shape = [n_params]*) – Packed parameters vector
- **user_args** (*kwargs*) – Keyword arguments from user

parameters()
Return the set of fitted parameters used by this expression.

Return type set of FittedParam.

CHAPTER 3

Overview

squmfit is a general-purpose library for non-linear least-squared curve fitting. The library is designed to enable modular models which can be easily composed to describe complex data sets. Moreover, the library supports the ability to simultaneously fit multiple data sets, each with its own model, over a common set of parameters.

In contrast to lmfit, *squmfit* treats the free parameters of the the fit as first-class objects. This allows models to be built up, added, multiplied, and generally treated as standard Python expressions. Moreover, no assumptions are imposed regarding how parameters interact, allowing fitting of a common set of parameters over several data sets.

3.1 A simple example

Let's say that we have a exponential decay with Poissonian noise,

```
>>> import numpy as np
>>> noise_std = 0.1
>>> xs = np.arange(4000)
>>> ys = np.random.poisson(400 * np.exp(-xs / 400.))
```

We first define the functional form which we believe describes our data,

```
>>> import squmfit
>>> def exponential_model(t, amplitude, rate):
>>>     return amplitude * np.exp(-t * rate)
```

We then create a *Fit* object which we will use to define the parameters and objective of our fit,

```
>>> fit = squmfit.Fit()
```

Say we want to fit this model to our generated `ys`, allowing both the `amplitude` and `rate` parameters to vary. We first need to tell `squmfit` about these parameters,

```
>>> amp = fit.param('amp', initial=100)
>>> tau = fit.param('tau', initial=100)
```

Now we can use `amp` and `tau` as normal Python variables,

```
>>> model = exponential_model(xs, amp, 1. / tau)
```

Alternatively we could simply do away with the function altogether,

```
>>> model = amp * np.exp(-t / tau)
```

Note how we can write expressions involving parameters with the usual Python arithmetic operations, such as `1. / tau`, greatly simplifying model composition.

Next we add our curve to our `Fit`, specifying the model to which we wish to fit along with some weights (taking care to avoid division-by-zero, of course),

```
>>> weights = np.zeros_like(ys, dtype='f')
>>> weights[ys > 0] = 1 / np.sqrt(ys[ys > 0])
>>> fit.add_curve('curve1', model, ys, weights=weights)
```

Finally we can run our fit and poke around at the results,

```
>>> res = fit.fit()
>>> print res.params
{'amp': 403.01725751512635, 'tau': 393.19866908823133}
>>> print res.curves['curve1'].reduced_chi_sqr
0.949579885697
```

`res` is a *`FitResult`* object, which contains a variety of information about the fit, including the residuals, covariances, and fit values.

`squmfit` has a variety of options for presenting the results of a fit. *`squmfit.pretty`* has several utilities for producing a quantitative summary of the fit. For instance, *`squmfit.pretty.markdown_fit_result()`* will produce a Markdown document describing the fit parameters and various goodness-of-fit metrics. If you use IPython Notebook, *`squmfit.pretty.ipynb_fit_result()`* can be used to generate a presentation that can be rendered in rich HTML within the notebook.

Finally, *`squmfit.plot.plot_fit()`* can be used to plot fits and residuals.

3.2 How does it work?

Expressions in `squmfit` are represented by the *`Expr`* class. An `Expr` captures an abstract syntax tree that describes *how* a result can be computed. The elementary expressions could represent a fitted parameter (e.g. `amp` in the example above), or an expression depending upon a fitted parameter. The `Expr` class implements basic arithmetic operations (e.g. `__add__`) and numpy's ufuncs (e.g. `sqrt`), allowing it to be treated as a scalar or an array.

Of course, some functions require more structure beyond the operations supported by `Expr` evaluate their result. In this case, you can tell `squmfit` that you want the function arguments to be evaluated before they are provided to your function with the *`model()`* decorator,

```
>>> @squmfit.model
>>> def sum_odds(vec):
>>>     return vec[1::2].sum()
```

In this case, we could invoke `sum_odds` with an *`Expr`*, which `squmfit` would automatically evaluate. It would then evaluate `sum_odds` with the value of the expression, and pack the result back into an *`Expr`*.

squmfit.**model**(*func*)

Transforms a function to ensure that all of its parameters are evaluated. Calls to the transformed function will result in an *Expr* when any of its parameters are Exprs.

CHAPTER 4

Fitting

Fitting begins with the *Fit* class which is used to define the data sets and models to be fit and ultimately

class squmfit.**Fit**
This represents a fit configuration.

CHAPTER 5

Inspecting fit results

class squmfit.**FitResult**(*fit*, *params*, *covar_p=None*, *initial_result=None*)

This embodies a set of parameter values, possibly originating from a fit.

CHAPTER 6

Functional interpretation

The design of `squmfit` is largely inspired by patterns introduced to the author by the functional programming community. If one is so inclined it can be useful to think of *`Expr`* as an reader applicative having access to an environment of packed parameter values. Composition is standard function composition. Pure values can be lifted with the *`squmfit.expr.Constant`* class although this is rarely explicitly needed as pure values are automatically lifted by the operations of *`Expr`*. The *`Expr.map()`* function provides functorial map. Functions can also be lifted with the *`model()`* decorator.

Python Module Index

S

Index

Symbols

A

C

D

E

F

G

H

I

L

M

N

P

R

S

T