LMFit for Python

Non-Linear Least-Squares Minimization and Curve-Fitting for Python

https://lmfit.github.io/lmfit-py/

Introduction

LMFIT will help you on the use of non-linear least-squares problems and apply these models to real data

$$\chi^2 = \sum_i^N \frac{[y_i^{\rm meas} - y_i^{\rm model}(\mathbf{v})]^2}{\epsilon_i^2}$$

This could be done with scipy:

```python
def residual(vars, x, data, eps_data):
    amp = vars[0]
    phaseshift = vars[1]
    freq = vars[2]
    decay = vars[3]

    model = amp * sin(x * freq  + phaseshift) * exp(-x*x*decay)

    return (data-model)/eps_data
```

```python
from scipy.optimize import leastsq
vars = [10.0, 0.2, 3.0, 0.007]
out = leastsq(residual, vars, args=(x, data, eps_data))
```

Introduction

The SciPy library is robust and easy to use and follows basically the same recipe as LMFIT, however:

- We need to track of the order of the variables, and their meaning.
- To fix a particular variable (not vary it in the fit), the residual function has to be altered to have fewer variables
- There is no simple, robust way to put bounds on values for the variables, or enforce mathematical relationships between the variables.

The LMFIT module overcomes these shortcomings by using:

objects - a core reason for working with Python.

LMFIT - Pros

The key concept for LMFIT is to use **Parameter** objects instead of plain floating point numbers as the variables for the fit.

By using Parameter objects (or the closely related Parameters – a dictionary of Parameter objects), one can:

- forget about the order of variables and refer to Parameters by meaningful names.
- place bounds on Parameters as attributes, without worrying about order.
- fix Parameters, without having to rewrite the objective function.
- place algebraic constraints on Parameters.

LMFIT – Parameter Object

```
from lmfit import minimize, Parameters

def residual(params, x, data, eps_data):
    amp = params['amp'].value
    pshift = params['phase'].value
    freq = params['frequency'].value
    decay = params['decay'].value

    model = amp * sin(x * freq  + pshift) * exp(-x*x*decay)

    return (data-model)/eps_data

params = Parameters()
params.add('amp', value=10)
params.add('decay', value=0.007)
params.add('phase', value=0.2)
params.add('frequency', value=3.0)

out = minimize(residual, params, args=(x, data, eps_data))
```

LMFIT – Parameter Object

```
from lmfit import minimize, Parameters

def residual(params, x, data, eps_data):
    amp = params['amp'].value
    pshift = params['phase'].value
    freq = params['frequency'].value
    decay = params['decay'].value

    model = amp * sin(x * freq  + pshift) * exp(-x*x*decay)

    return (data-model)/eps_data

params = Parameters()
params.add('amp', value=10, vary=False)
params.add('decay', value=0.007, min=0.0)
params.add('phase', value=0.2)
params.add('frequency', value=3.0, max=10)

out = minimize(residual, params, args=(x, data, eps_data))
```

```
params['amp'].vary = False
params['decay'].min = 0.10
```

LMFIT allows switching optimization methods without changing the objective function, provides tools for writing fitting reports, and provides better determination of Parameters confidence levels.

LMFIT - Minimize

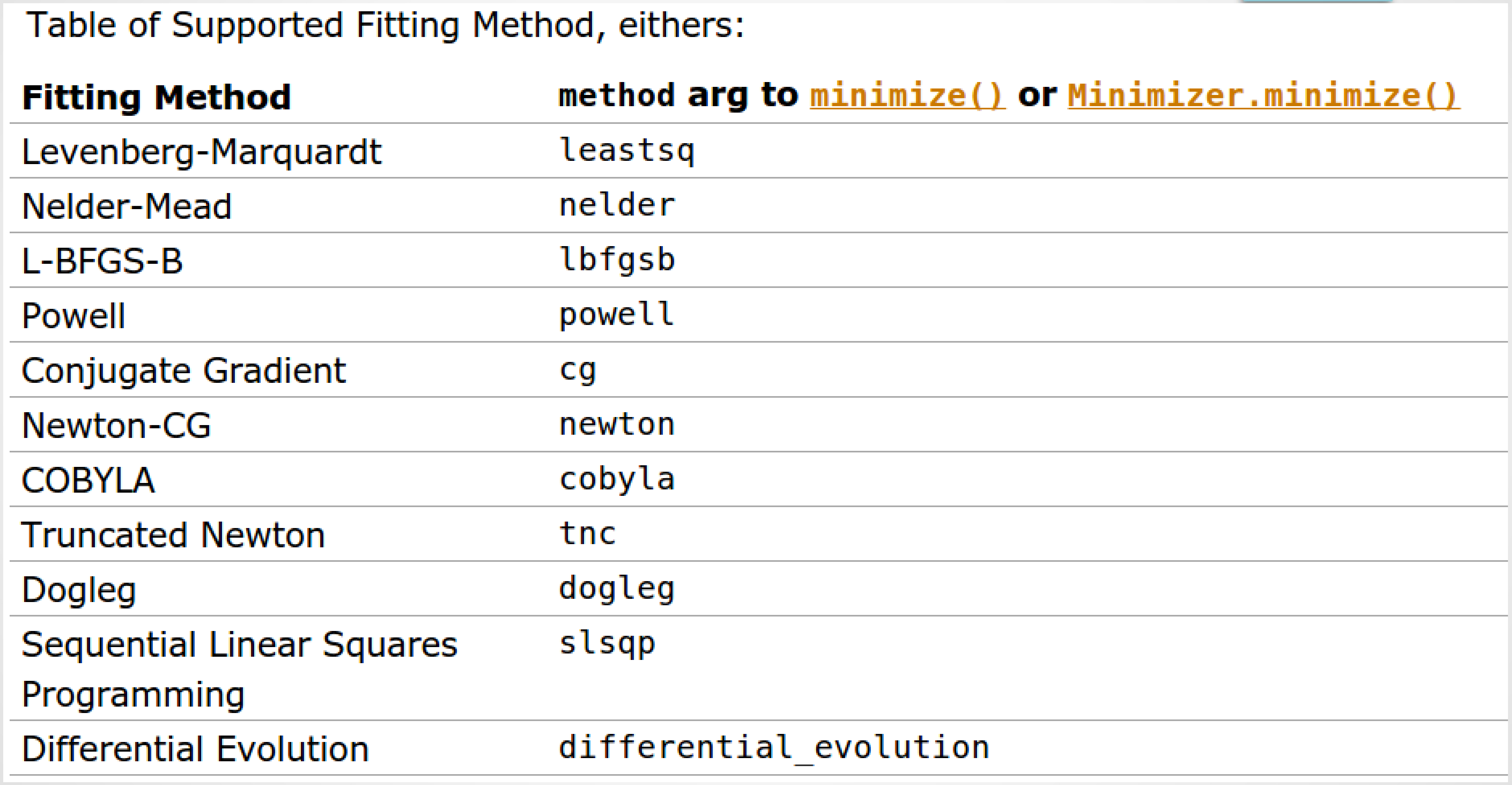


Table of Supported Fitting Method, eithers:

Fitting Method	**method arg to `minimize()` or `Minimizer.minimize()`**
Levenberg-Marquardt	`leastsq`
Nelder-Mead	`nelder`
L-BFGS-B	`lbfgsb`
Powell	`powell`
Conjugate Gradient	`cg`
Newton-CG	`newton`
COBYLA	`cobyla`
Truncated Newton	`tnc`
Dogleg	`dogleg`
Sequential Linear Squares Programming	`slsqp`
Differential Evolution	`differential_evolution`

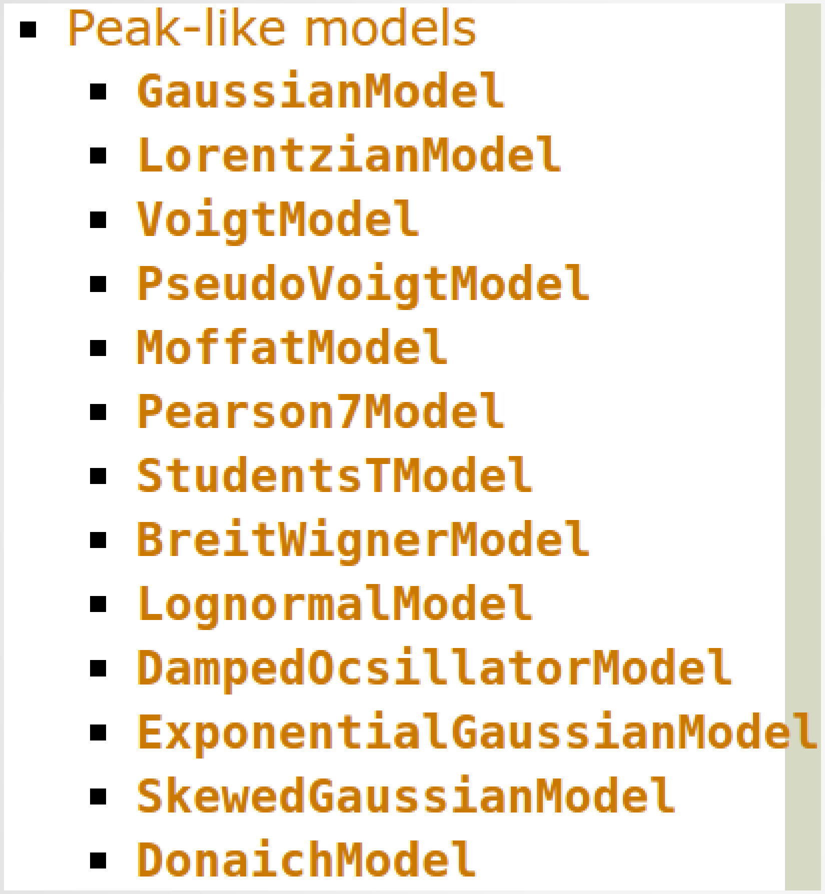
LMFIT - Model

Model allows us to wrap a model. This automatically generate the appropriate residual function, and determines the corresponding parameter names.

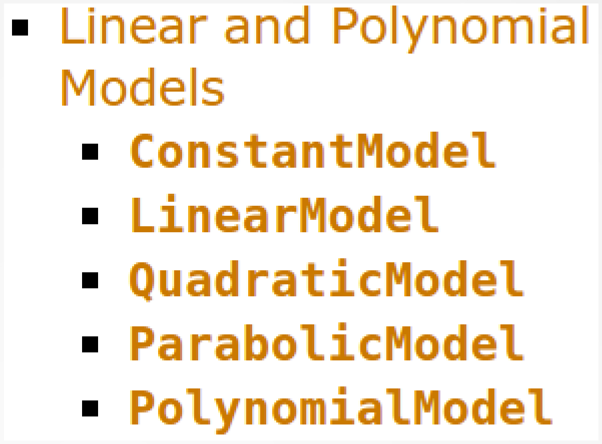
```python
def gaussian(x, amp, cen, wid):
    "1-d gaussian: gaussian(x, amp, cen, wid)"
    return (amp/(sqrt(2*pi)*wid)) * exp(-(x-cen)**2 /(2*wid**2))

gmod = Model(gaussian)
result = gmod.fit(y, x=x, amp=5, cen=5, wid=1)

print(result.fit_report())
```

This is a quite extensive object. You can define whatever model you wish with this model, and have access to a series of modules that are helpful to define the parameters and the fitting of the data

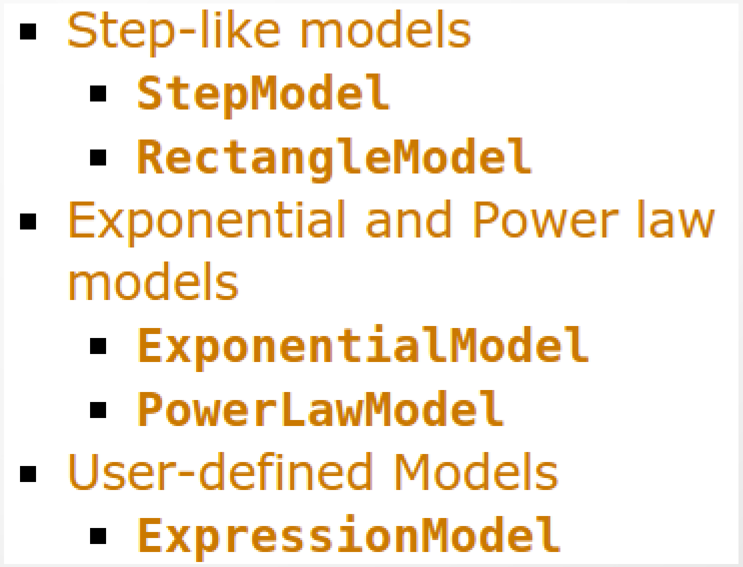
LMFIT – Built in Models



- Peak-like models
  - **`GaussianModel`**
  - **`LorentzianModel`**
  - **`VoigtModel`**
  - **`PseudoVoigtModel`**
  - **`MoffatModel`**
  - **`Pearson7Model`**
  - **`StudentsTModel`**
  - **`BreitWignerModel`**
  - **`LognormalModel`**
  - **`DampedOcsillatorModel`**
  - **`ExponentialGaussianModel`**
  - **`SkewedGaussianModel`**
  - **`DonaichModel`**



- Linear and Polynomial Models
  - **`ConstantModel`**
  - **`LinearModel`**
  - **`QuadraticModel`**
  - **`ParabolicModel`**
  - **`PolynomialModel`**



- Step-like models
  - **`StepModel`**
  - **`RectangleModel`**
- Exponential and Power law models
  - **`ExponentialModel`**
  - **`PowerLawModel`**
- User-defined Models
  - **`ExpressionModel`**

LMFIT – And more...

- Confidence Levels

The LMFIT confidence module allows you to explicitly calculate confidence intervals for variable parameters. For most models, it is not necessary: the estimation of the standard error from the estimated covariance matrix is normally quite good.

- Constraints

Being able to fix variables to a constant value or place upper and lower bounds on their values can greatly simplify modeling real data. These capabilities are key to LMFIT’s Parameters. In addition, it is sometimes highly desirable to place mathematical constraints on parameter values.