MLFittingRoutine.py

Marquardt-Levenberg Fitting

Parameter Optimization for Atomic Spectra

ElecSus Library Documentation

Abstract

This document provides comprehensive documentation for `MLFittingRoutine.py`, which implements spectral fitting using the lmfit library. The module enables extraction of physical parameters (temperature, magnetic field, cell length, etc.) from experimental atomic spectra through nonlinear least-squares optimization.

Contents

1 Theoretical Foundation **2**
1.1 Nonlinear Least Squares 2
1.2 Parameter Estimation 2

2 Physical Model Parameters **2**
2.1 Fitting Parameters 2

3 Line-by-Line Code Analysis **3**
3.1 Module Imports 3
3.2 Fit Function Wrapper 3
3.3 Main Fitting Function: ML_fit 3
3.4 Parameter Control 4
3.5 Fitting Execution 4

4 Available Fitting Methods **4**

5 Usage Example **5**

6 Summary **5**

1 Theoretical Foundation

1.1 Nonlinear Least Squares

Axiom 1 (Optimization Principle). *The best-fit parameters minimize the sum of squared residuals:*

$$\chi^2(\vec{p}) = \sum_{i=1}^{N} [y_i - f(x_i; \vec{p})]^2 \tag{1}$$

where $\vec{p}$ is the parameter vector and f is the model function.

Definition 1 (Levenberg-Marquardt Algorithm). *An iterative method that interpolates between gradient descent and Gauss-Newton:*

$$\vec{p}_{k+1} = \vec{p}_k - (J^T J + \lambda\, diag(J^T J))^{-1} J^T \vec{r} \tag{2}$$

where J is the Jacobian matrix, $\vec{r}$ is the residual vector, and λ is the damping parameter.

Theorem 1 (Convergence Properties). *The L-M algorithm converges to a local minimum. For well-posed problems with good initial guesses, convergence is typically quadratic near the minimum.*

1.2 Parameter Estimation

Definition 2 (Covariance Matrix). *The parameter covariance matrix at the minimum:*

$$Cov(\vec{p}) \approx \sigma^2 (J^T J)^{-1} \tag{3}$$

where $\sigma^2 = \chi^2/(N - M)$ is the reduced chi-squared.

Theorem 2 (Parameter Uncertainties). *The standard error of parameter p_i is:*

$$\sigma_{p_i} = \sqrt{Cov(\vec{p})_{ii}} \tag{4}$$

2 Physical Model Parameters

2.1 Fitting Parameters

Parameter	Symbol	Physical Meaning
`T`	T	Temperature (K)
`lcell`	L	Cell length (m)
`Bfield`	B	Magnetic field (G)
`Btheta`	θ_B	Field polar angle (rad)
`Bphi`	ϕ_B	Field azimuthal angle (rad)
`GammaBuf`	Γ_{buf}	Buffer gas broadening (MHz)
`shift`	δ	Frequency offset (MHz)
`DoppTemp`	T_D	Doppler temperature (K)
`E_x, E_y`	E_x, E_y	Electric field components
`E_phase`	ϕ_E	Relative phase (rad)
`rb85frac`	f_{85}	^{85}Rb fraction (%)

Table 1: Physical parameters in the fitting model

3 Line-by-Line Code Analysis

3.1 Module Imports

```
import matplotlib.pyplot as plt
import numpy as np
import lmfit as lm
from spectra import get_spectra
```

Import plotting, numerical, fitting libraries, and the spectrum calculator.

3.2 Fit Function Wrapper

```
def fit_function(x,E_x,E_y,E_phase,T,lcell,Bfield,Btheta,Bphi,
                 GammaBuf,shift,DoppTemp=20,rb85frac=72.17,
                 K40frac=0.01,K41frac=6.73,Elem='Rb',Dline='D2',
                 Constrain=True,output='S0', verbose=False):
```

Wrapper function with explicit parameter arguments for lmfit compatibility.

```
    # Ex / Ey separated to allow for fitting polarisation
    E_in = np.array([E_x,E_y*np.exp(1.j*E_phase),0.])
```

$$\vec{E}_{in} = (E_x, E_y e^{i\phi_E}, 0) \tag{5}$$

Construct complex electric field vector. Phase allows elliptical polarization.

```
    #reconstruct parameter dictionary from arguments
    p_dict = {'Elem':Elem,'Dline':Dline,'T':T,'lcell':lcell,
              'Bfield':Bfield,'Btheta':Btheta,'Bphi':Bphi,
              'GammaBuf':GammaBuf,'shift':shift,'DoppTemp':DoppTemp,
              'Constrain':Constrain,'rb85frac':rb85frac,
              'K40frac':K40frac,'K41frac':K41frac}

    outputs = [output]
    y_out = get_spectra(x,E_in,p_dict,outputs)[0].real
    return y_out
```

Call the spectrum generator and return real-valued output for fitting.

3.3 Main Fitting Function: ML_fit

```
def ML_fit(data,E_in,p_dict,p_dict_bools,data_type='S0',
           p_dict_bounds=None,method='leastsq',verbose=False):
    """Main fitting method."""

    x = np.array(data[0])
    y = np.array(data[1])
```

Extract x (detuning) and y (signal) from data.

```
    # Non-numeric arguments to pass to fitting function
    kwargz = {'Elem':p_dict['Elem'],'Dline':p_dict['Dline']}
    if 'Constrain' in list(p_dict.keys()):
        kwargz['Constrain'] = p_dict['Constrain']
    else:
        kwargz['Constrain'] = True
    kwargz['output'] = data_type
```

Set up keyword arguments for non-fitted parameters.

```
    model = lm.Model(fit_function)
    params = model.make_params(**p_dict)
```

Create lmfit Model object and initialize parameters.

```
    params['E_x'].value = E_in[0]
    params['E_y'].value = E_in[1][0]
    params['E_phase'].value = E_in[1][1]
    params['E_phase'].min = 0
    params['E_phase'].max = np.pi-1e-4
```

Set polarization parameters with bounds on phase.

3.4 Parameter Control

```
    # Turn off all parameters varying by default
    allkeys = params.valuesdict()
    for key in allkeys:
        params[key].vary = False

    # Turn on fitting parameters as specified in p_dict_bools
    for key in p_dict_bools:
        params[key].vary = p_dict_bools[key]

        if p_dict_bounds is not None:
            if key in p_dict_bounds:
                params[key].min = p_dict_bounds[key][0]
                params[key].max = p_dict_bounds[key][1]
```

Selective parameter variation: only fit parameters marked True in `p_dict_bools`.

3.5 Fitting Execution

```
    result = model.fit(y, x=x, params=params, method=method, **kwargz)
    return result.best_values, result
```

Execute fit and return best-fit values plus full result object.

4 Available Fitting Methods

Method String	Algorithm
`'leastsq'`	Levenberg-Marquardt (default)
`'least_squares'`	Trust Region Reflective
`'differential_evolution'`	Global optimization
`'nelder'`	Nelder-Mead simplex
`'powell'`	Powell's method
`'cobyla'`	COBYLA
`'slsqp'`	Sequential Least Squares

Table 2: Supported optimization algorithms

5 Usage Example

```python
# Experimental data
data = [detuning_array, transmission_array]

# Initial parameters
p_dict = {'Elem':'Rb', 'Dline':'D2', 'T':350, 'lcell':75e-3,
          'Bfield':100, 'Btheta':0, 'Bphi':0, 'GammaBuf':0,
          'shift':0, 'DoppTemp':350, 'rb85frac':72.17}

# Which parameters to fit
p_dict_bools = {'T':True, 'Bfield':True, 'shift':True}

# Parameter bounds
p_dict_bounds = {'T':(300, 500), 'Bfield':(0, 1000)}

# Input polarization [E_x, (E_y_amplitude, phase)]
E_in = [1, (0, 0)]

# Run fit
best_values, result = ML_fit(data, E_in, p_dict, p_dict_bools,
                             p_dict_bounds=p_dict_bounds)

print(result.fit_report())
```

6 Summary

The `MLFittingRoutine.py` module provides:

1. `fit_function`: lmfit-compatible wrapper for spectrum calculation
2. `ML_fit`: Main fitting interface with flexible parameter control

Key features:

- Selective parameter fitting via Boolean dictionary
- Parameter bounds support
- Multiple optimization algorithms
- Returns full lmfit result object with uncertainties, correlations
- Supports all Stokes parameters as fit targets

The module enables precision determination of:

- Vapor temperature from Doppler width
- Magnetic field from Zeeman splitting
- Cell length from absorption depth
- Isotope ratios from line intensities