

PyCFIT User Guide

Python Component Fitting System

Version 1.1.0

Table of Contents

- [PyCFIT User Guide](#)
 - [Table of Contents](#)
 - [Introduction](#)
 - [Purpose and Description](#)
 - [Inspiration](#)
 - [Key Features](#)
 - [Installation](#)
 - [System Requirements](#)
 - [Python Package Install](#)
 - [Platform-Specific Notes](#)
 - [MacOS 26 \(Tahoe\)](#)
 - [Quick Start Guide](#)
 - [Single Spectrum Fitting](#)
 - [Getting Started](#)
 - [Loading an Existing Model](#)
 - [Working with Components](#)
 - [Adding Components](#)
 - [Removing Components](#)
 - [Adjusting Component Parameters](#)
 - [Interactive Graph Adjustment](#)
 - [Parameter Dialog](#)
 - [Component Management Features](#)
 - [Sorting Components](#)
 - [Freeing Parameters](#)
 - [Expand/Collapse Tree](#)
 - [Naming Components](#)
 - [Finding the Best Fit](#)
 - [Model Output Options](#)
 - [Returning the Model](#)
 - [Exporting Models](#)
 - [Raster \(Grid\) Fitting](#)
 - [Creating a Grid Object](#)
 - [Grid Interface Overview](#)
 - [Navigating the Grid](#)

- Column Details
- Inspecting and Modifying Fits
 - Adjusting Color Scales
 - Masking Points
 - Refitting a Single Point
 - Refitting the Entire Grid
- Goodness of Fit Visualization
 - Residuals Mode (Default)
 - Degrees of Freedom Mode
 - Reduced Chi-Squared Mode
- Working with Results
 - The `get_results()` Method
 - Saving and Loading with ASDF
- Advanced Features
 - Working with Astropy Units
 - Custom Fitter Parameters
- Library API Usage
 - Single Spectrum API
 - Creating a FunctionFitter
 - Adding Components
 - Modifying Parameters
 - Parameter Identifiers
 - Deleting Components
 - Fitting
 - Retrieving Fit Information
 - Grid API
 - Creating a Grid
 - Fitting the Grid
 - Accessing Individual Points
 - Working with Results
 - Saving and Loading
- Tips and Best Practices
 - Supported Component Models
 - When to Use Each Component
 - Single Spectrum Fitting
 - Grid Fitting
 - Parameter Constraints
 - Performance
- Troubleshooting
 - Common Issues
 - GUI doesn't open on MacOS
 - Fit doesn't converge
 - Tied parameters create circular dependencies

- [Grid fitting is very slow](#)
 - [Memory error with large grids](#)
 - [Can't load saved ASDF file](#)
 - [Parameter tied expression not working](#)
 - [Contact Information](#)
 - [Software Maintenance](#)
 - [NASA Official](#)
 - [Documentation and Source](#)
 - [License](#)
-

Introduction

Purpose and Description

The **PyCFIT** (Python Component Fitting System) software facilitates the analysis of raster data from solar imaging spectrographs and comprises two main modules:

1. **Single Spectrum Fitting Interface**: Build complex models from simple component models to fit a single spectrum
2. **Grid Fitting Interface**: Extend initial models across entire data blocks efficiently

While PyCFIT can be used entirely from the library API in an interactive Python environment, its greatest value lies in its graphical components, which make finding and understanding fits easier and more intuitive.

Inspiration

PyCFIT was inspired by the longstanding [Component Fitting System \(CFIT\) for IDL](#) (Haugan, 1997), developed initially in support of the SoHO mission. As the solar physics community transitions from IDL to Python, we thank the IDL CFIT developers for their whole-hearted support and the lessons learned that informed our design.

Key Features

- **Interactive GUI** for single spectrum and raster fitting
 - **Six component model types**: Constant, Linear, Quadratic, Gaussian, Moffat, and Voigt profiles
 - **Flexible parameter constraints**: Fixed, free, bounded, or tied parameters
 - **Component sorting** by spectral location for better organization
 - **Goodness of fit metrics**: DOF (degrees of freedom) and reduced χ^2 visualization
 - **ASDF file format** support for storing and recovering complete Grid objects
 - **Parallel fitting** for grids using Dask
 - **Library API** for scripting and automation
-

Installation

System Requirements

- Windows, MacOS, or Linux system
- Python 3.11 or higher
- pip or conda package installer

Python Package Install

```
# Install with pip
pip install pycfit

# or Install with conda from conda-forge channel
conda install -c conda-forge pycfit
```

This will install all required dependencies, including:

- Astropy >= 7.1
- dask
- asdf-astropy >= 0.3
- PyQt5, qtpy, pyqtgraph, pyqtspinner
- ipython

Platform-Specific Notes

MacOS 26 (Tahoe)

Due to Qt framework issues on recent MacOS versions, pyCFIT GUI components **must** be run from within an `ipython` session with the `qt` GUI magic command:

Option 1: Start ipython with Qt GUI magic:

```
ipython --gui=qt
```

Option 2: Enable Qt GUI from within an ipython session:

```
In [1]: %gui qt
```

Quick Start Guide

Here's a minimal example to get you started with pyCFIT:

```
import pycfit

# Load example single spectrum data
wavelength, intensity, uncertainty = pycfit.data.get_example_single_spectrum()

# Open single spectrum GUI to create a model
model = pycfit.cfit_gui(wavelength, intensity, uncertainty=uncertainty)

# Load example grid data
wavelength, intensity, uncertainty, mask = pycfit.data.get_example_grid_spectra()

# Create and fit a grid using the model returned from single spectrum fitting
myGrid = pycfit.cfit_grid(model, wavelength, intensity,
                          uncertainty=uncertainty, mask=mask)
myGrid.fit()

# Open grid GUI to inspect and modify fits
myGrid = pycfit.cfit_grid_gui(myGrid)

# Retrieve results
results = myGrid.get_results()
```

Single Spectrum Fitting

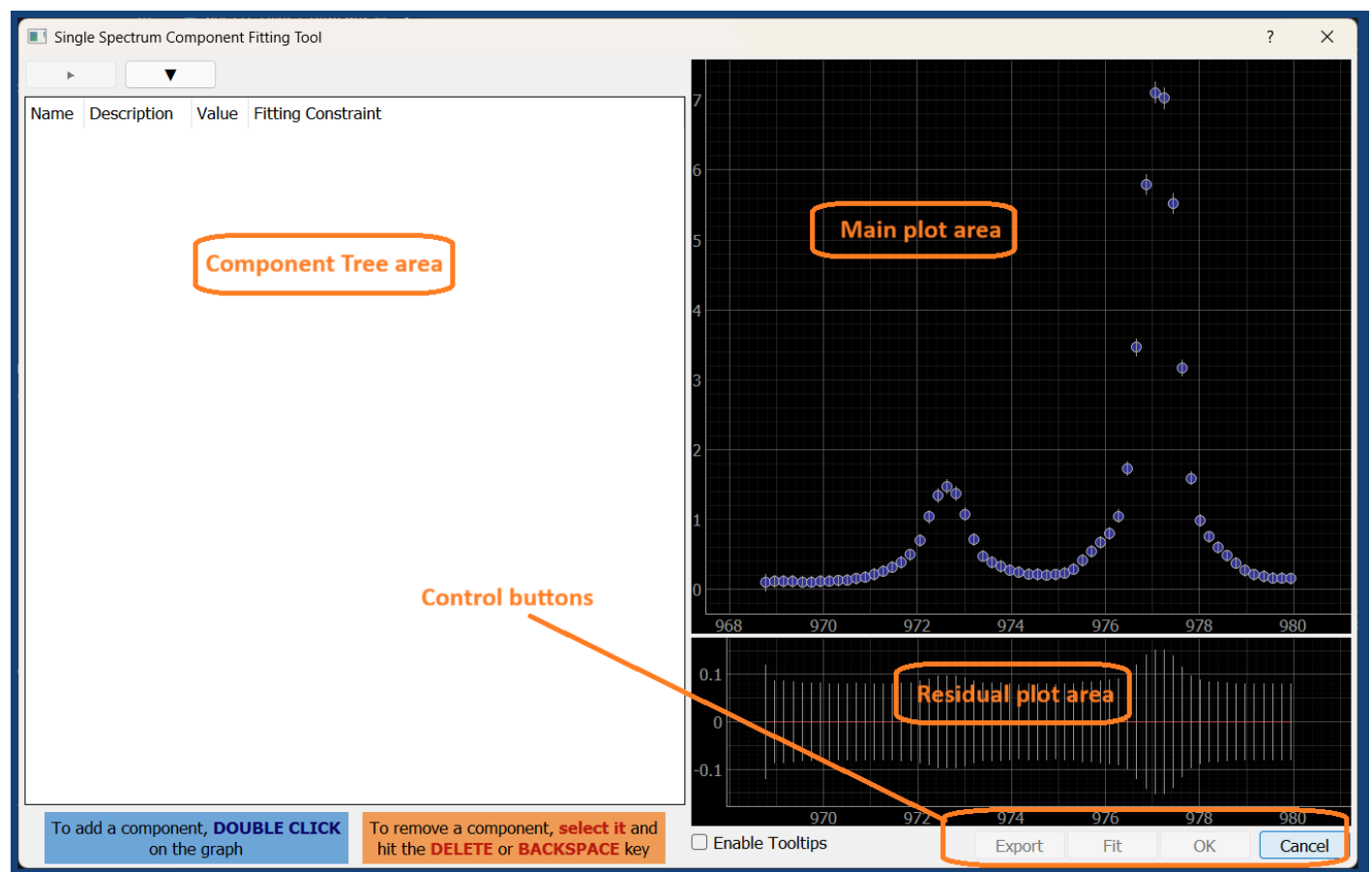
Getting Started

To initiate the graphical user interface for single spectrum fitting:

```
import pycfit

# Use sample data included with the package
wavelength,intensity,uncertainty = pycfit.data.get_example_single_spectrum()

# wavelength, intensity, uncertainty are 1D data arrays of the same shape
model = pycfit.cfit_gui(wavelength, intensity, uncertainty=uncertainty)
```



The GUI window displays:

- **Main plot area:** Intensity vs. wavelength with error bars
- **Residual plot** (below main plot): Shows fit residuals when a model is defined
- **Component Tree** (left panel): Lists all components and their parameters
- **Control buttons** (bottom right): Fit, Export, OK, Cancel

Loading an Existing Model

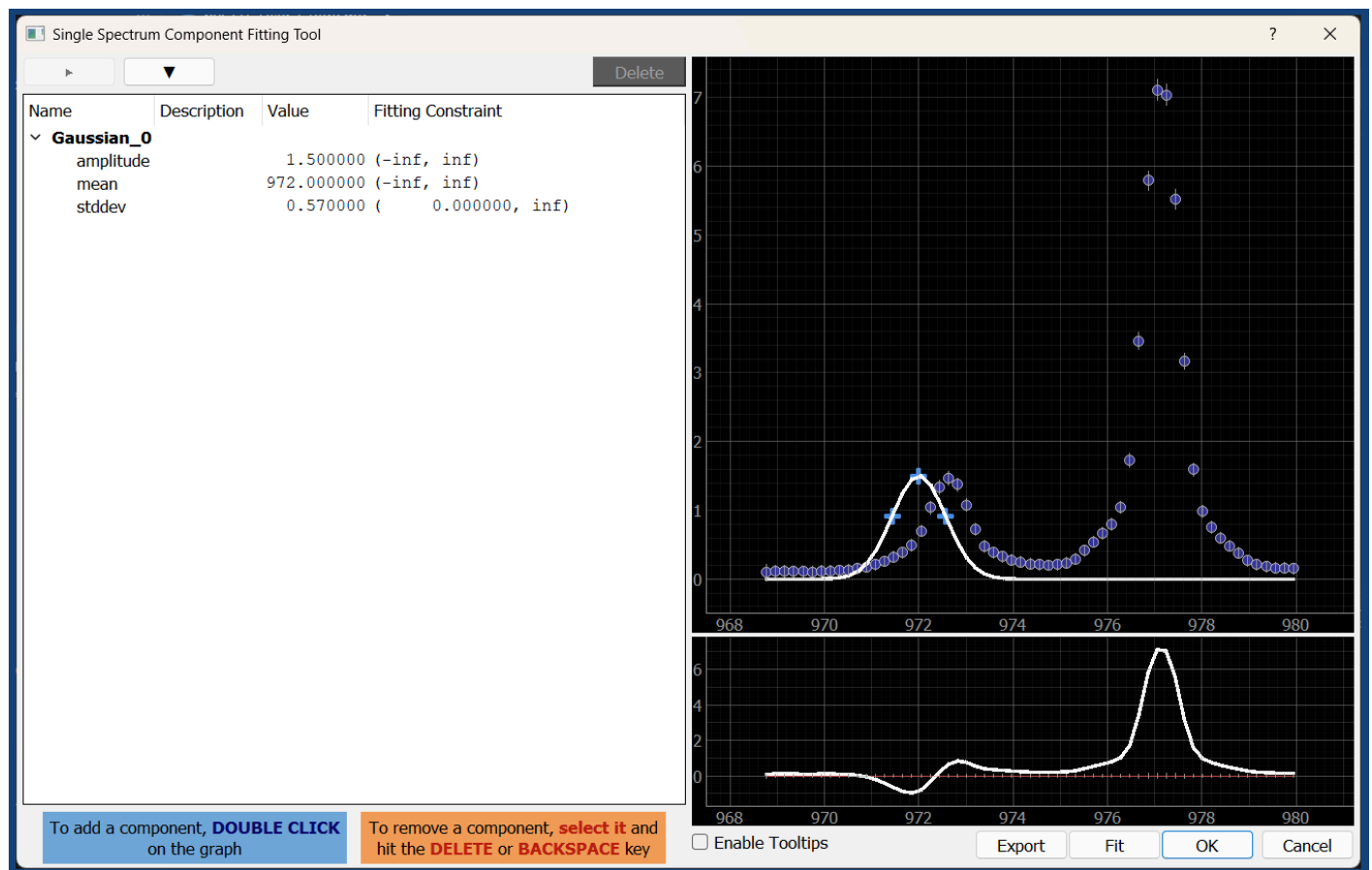
You can optionally load with an initial model:

```
from astropy.modeling.models import Gaussian1D

# Option 1: Pass an Astropy Model object
model_in = Gaussian1D(amplitude=1.5, mean=972, stddev=0.57)
model = pycfit.cfit_gui(wavelength, intensity,
                        uncertainty=uncertainty,
                        function=model_in)

# Option 2: Pass a path to a saved model file (.py or .pkl)
model = pycfit.cfit_gui(wavelength, intensity,
                        uncertainty=uncertainty,
                        function='saved_model.py')
```

The `function` keyword to `cfit_gui` will accept either an Astropy `Model` / `CompoundModel` object, a file path to a pickled Astropy model, or a file path to an appropriately defined Python model definition file. If providing a Python file, it should implement a `define_model()` function. Use `Export` as described in the [Exporting Models](#) section to see an example `.py` file.

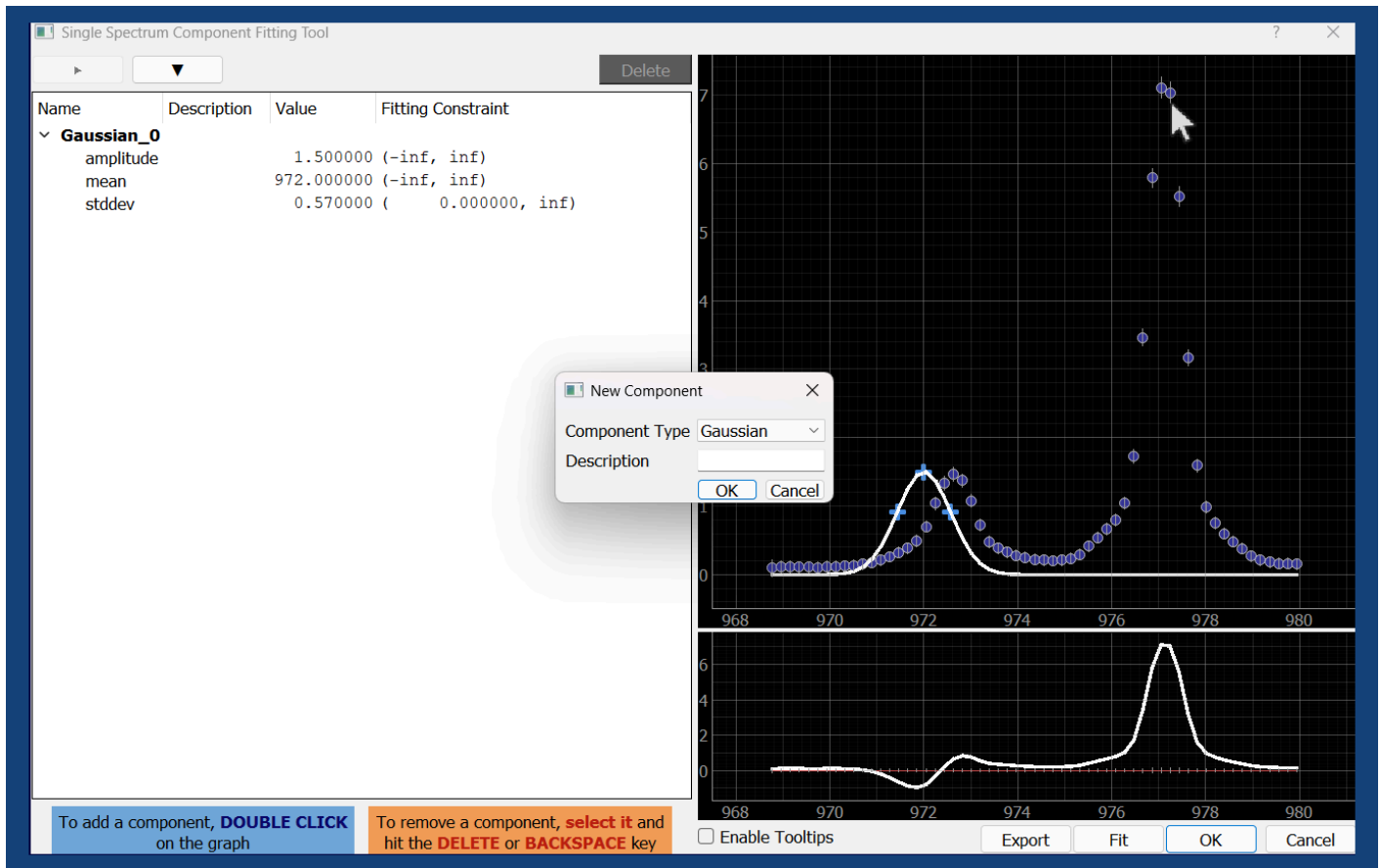


Working with Components

Adding Components

To add a new component:

1. **Double-click** anywhere on the graph window
2. Select the component type from the context menu:
 - Constant
 - Linear
 - Quadratic
 - Gaussian
 - Moffat
 - Voigt
3. Optionally, add descriptive text in the **Description** box



Default behaviors when adding components:

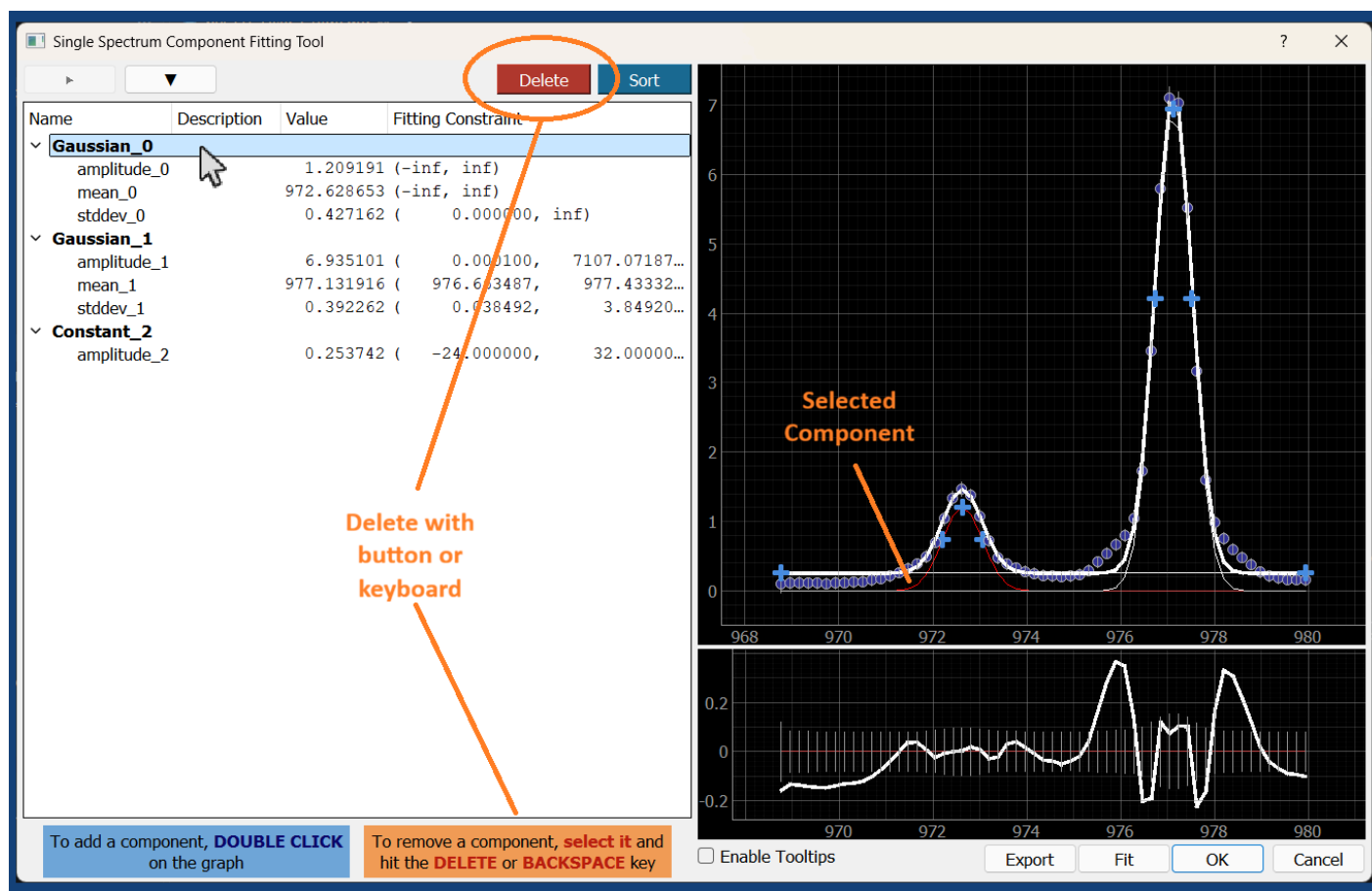
Component Type	Default Behavior
Constant	Line through clicked point with slope = 0
Linear	Line through clicked point with slope between first and last data point
Quadratic	2nd degree polynomial through endpoints and clicked point

Component Type	Default Behavior
Gaussian	Peak at clicked x-position, amplitude brings residual to zero at that point
Moffat	Peak at clicked x-position, amplitude brings residual to zero at that point
Voigt	Peak at clicked x-position, amplitude brings residual to zero at that point

Each Component added will appear in the plot window as a thin, white line and its name and Parameter information displayed in the Component Tree on the left side. A thicker white line in the main graph window displays the Compound Model and the lower graph panel displays the residuals between the data and the Compound Model. Selecting one of the elements in the Component Tree display with the mouse will highlight that Component Graph in red in the plot window.

Removing Components

1. **Click** to select a component in the Component Tree
2. The selected component will be highlighted in **red** on the graph
3. Press **DELETE** or **BACKSPACE** to remove it, OR hit the **Delete** button at the top of the Tree view

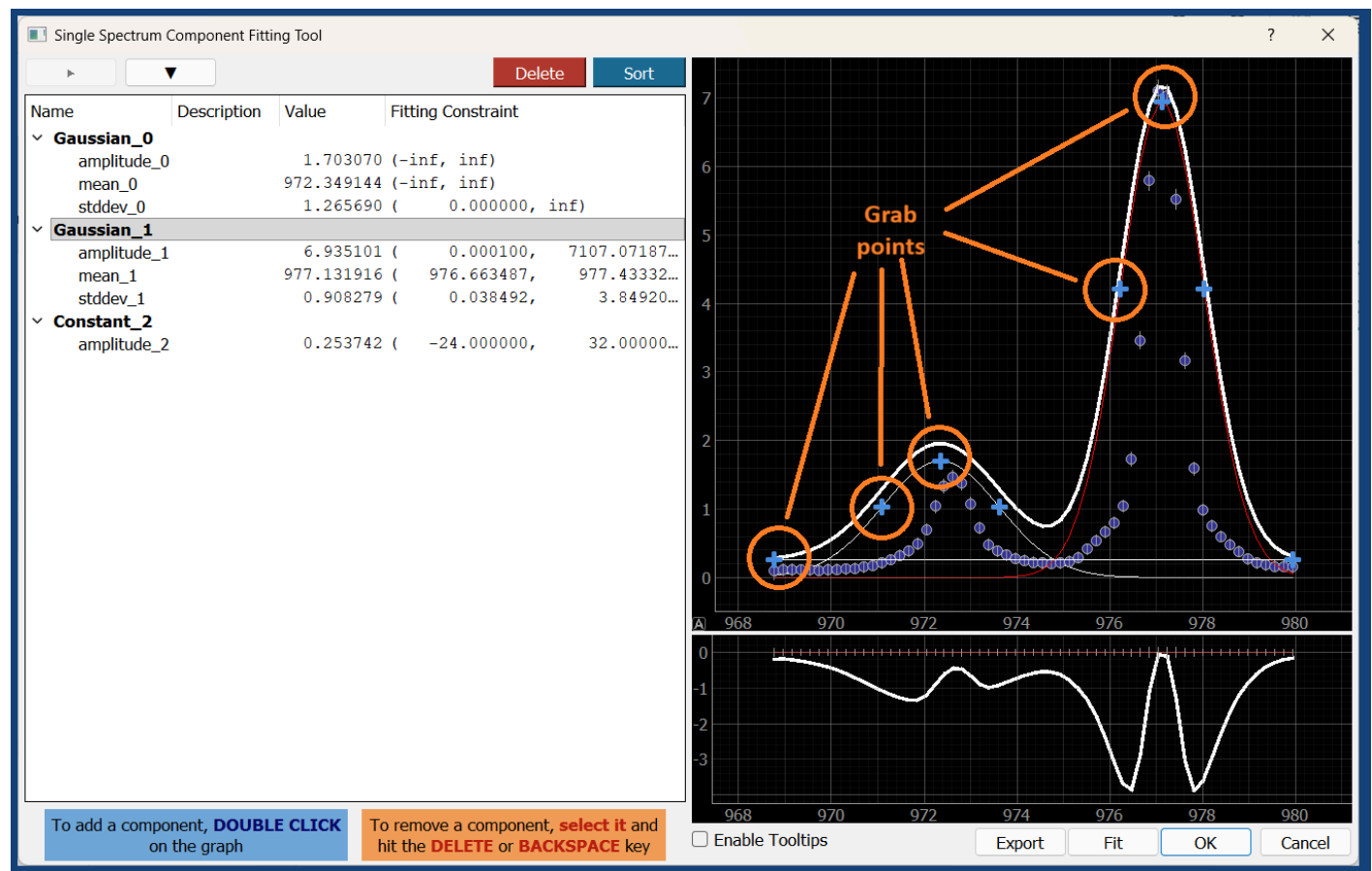


Adjusting Component Parameters

Interactive Graph Adjustment

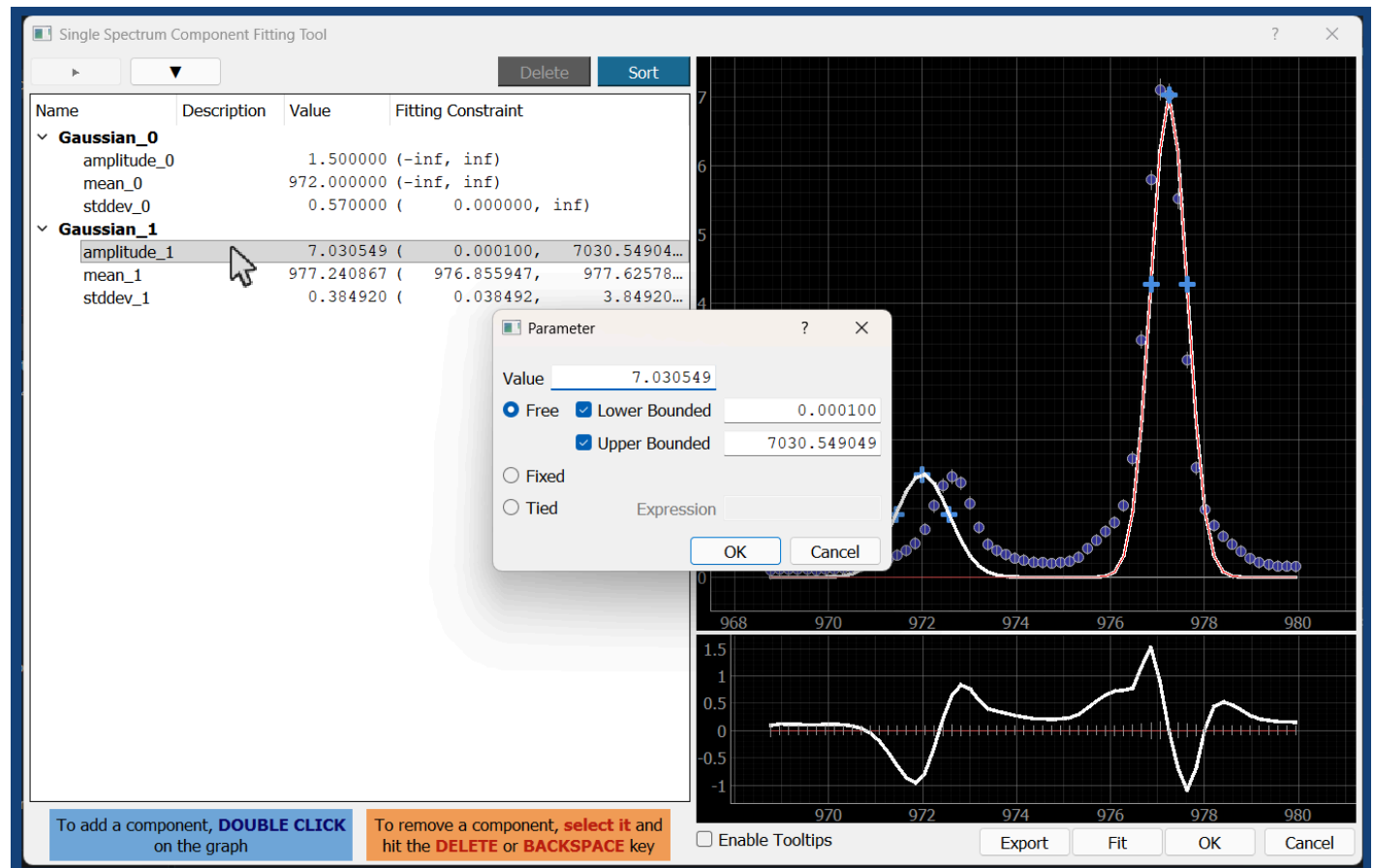
Each component has **plus-shaped (+) grab points** that can be dragged to reshape the component. The Voigt type Component also has **x-shaped (x) grab points** to differentiate the points affecting the Gaussian contribution from those affecting the Lorentzian contribution:

1. **Click and hold** a grab point
2. **Drag** to adjust the component shape
3. Related Parameter values update automatically and are visible in the Component Tree. When a drag event moves a free Parameter value outside set bounds, the affected bound is automatically adjusted to ensure that the new initial value falls within bounds. In this way, we avoid breaking the fit action before it begins.



Parameter Dialog

For precise parameter control, double-click a parameter in the Component Tree:



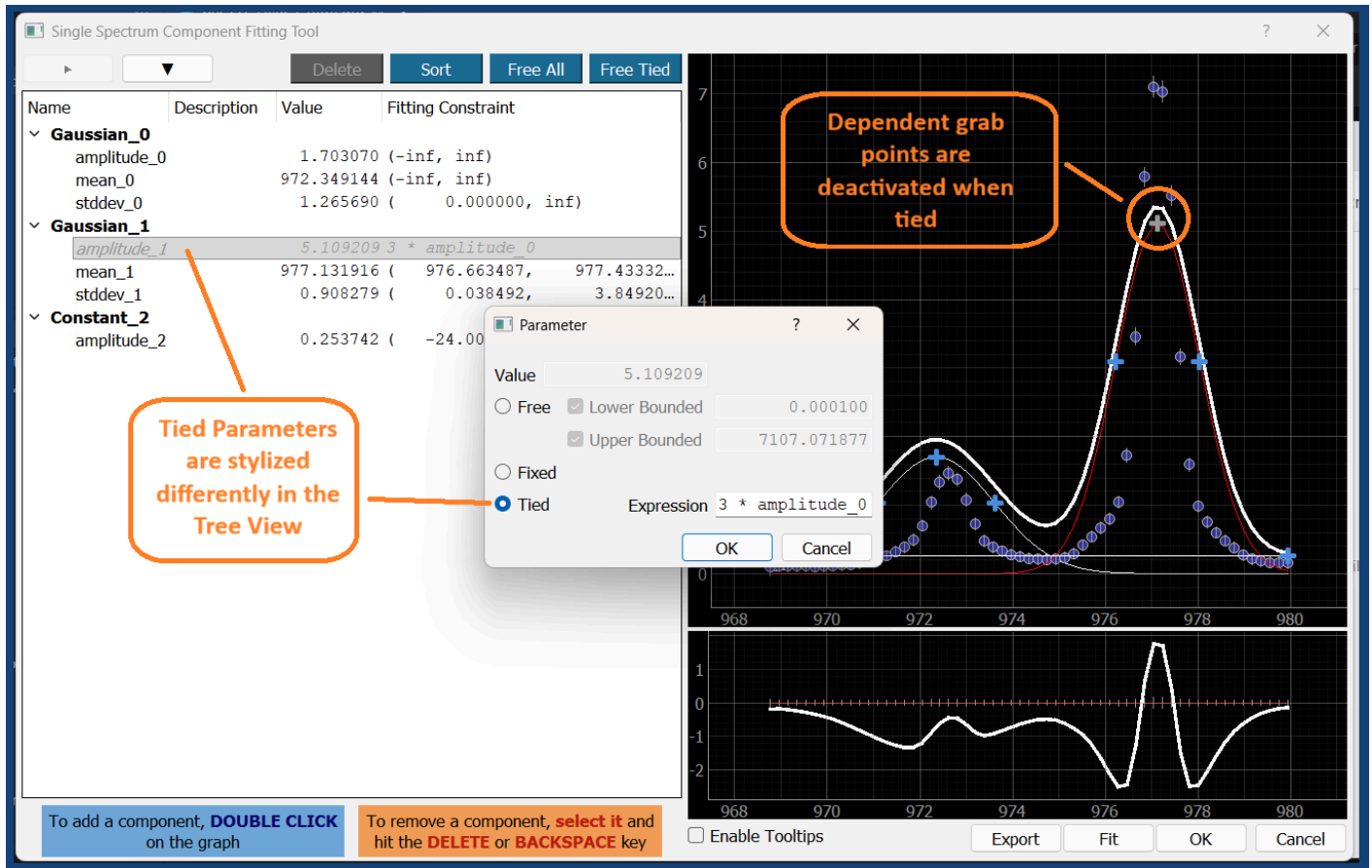
The Parameter Dialog allows you to:

1. **Set the parameter value** directly
2. **Choose constraint type:**
 - **Free:** Parameter varies during fitting (can optionally set bounds)
 - **Fixed:** Parameter held constant at the specified value
 - **Tied:** Parameter linked to other parameters via an expression

Example of tying parameters:

If you want $\text{amplitude}_1 = 0.3 * \text{amplitude}_2$:

- Select the tied option for amplitude_1
- Enter the expression: $0.3 * \text{amplitude}_2$



Parameter identifiers include a trailing numeric tag (e.g., `amplitude_1`, `mean_2`) corresponding to their component ID and are visible in the `Name` column of the Tree.

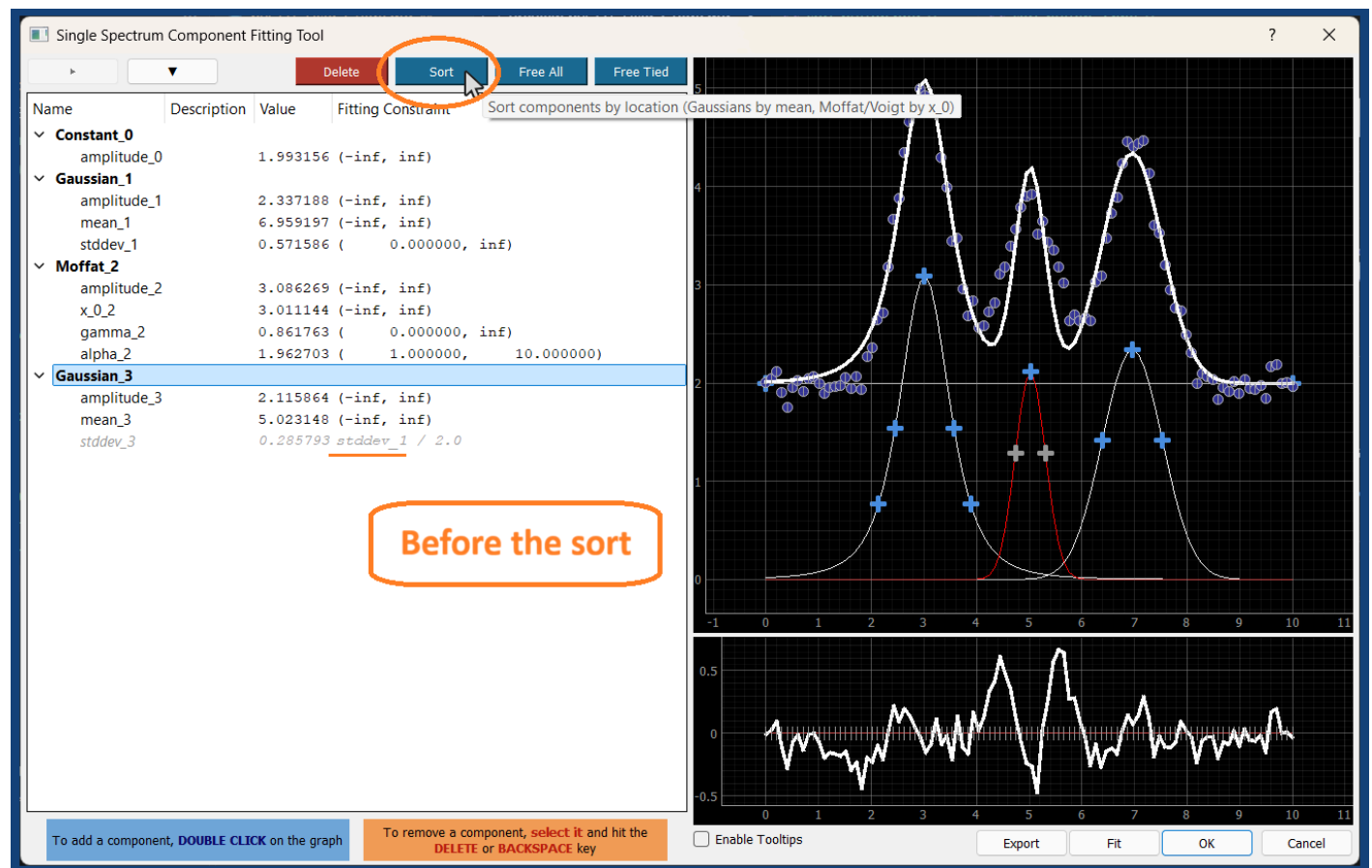
Component Management Features

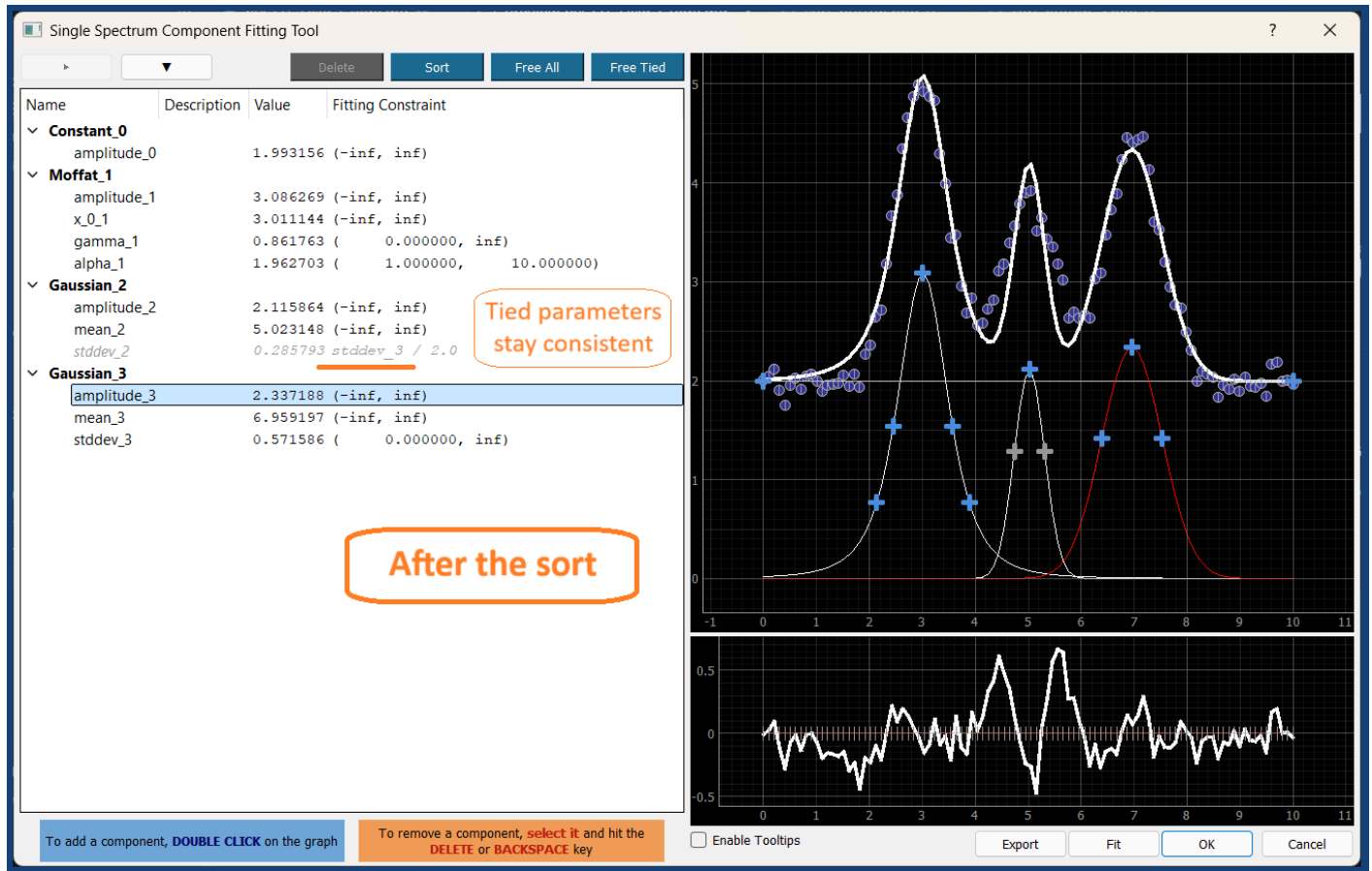
Sorting Components

Click the **Sort** button to automatically reorder components by spectral location:

1. **Unlocated components** (Constant, Linear, Quadratic) appear first
2. **Located components** (Gaussian, Moffat, Voigt) follow, sorted by their location parameter:
 - Gaussian: sorted by **mean**
 - Moffat and Voigt: sorted by **x_0**

This feature is particularly useful when you have multiple emission or absorption lines that need to be organized.





Freeing Parameters

Use the parameter management buttons above the Component Tree:

- **Free All:** Remove constraints from all parameters (makes them free) and reinstates the most recently set bounds, if any.
- **Free Tied:** Remove only tied constraints, leaving fixed parameters unchanged

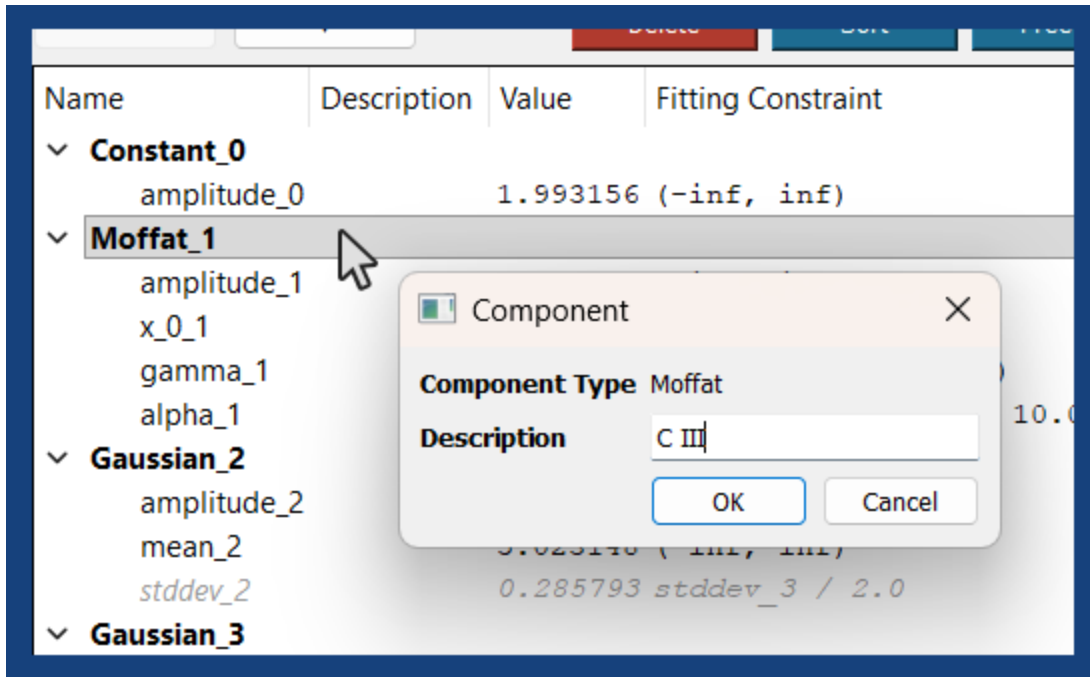
Expand/Collapse Tree

- ► button: Expand all components to show parameters
- ▼ button: Collapse all components

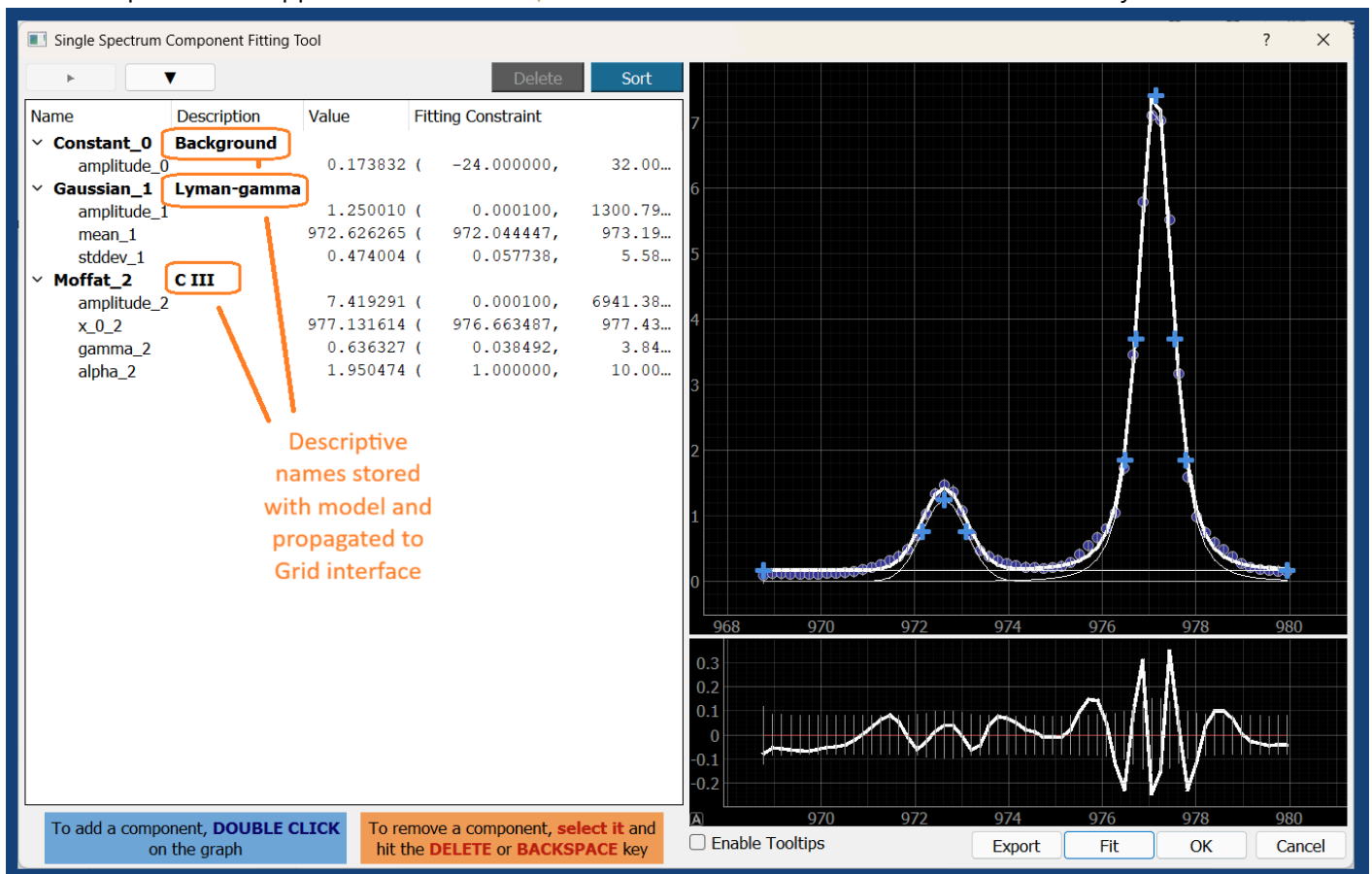
Naming Components

Give descriptive names to your components:

1. **Double-click** the component name in the Component Tree
2. Enter a descriptive name (e.g., "O III emission", "Background")
3. Click **OK**

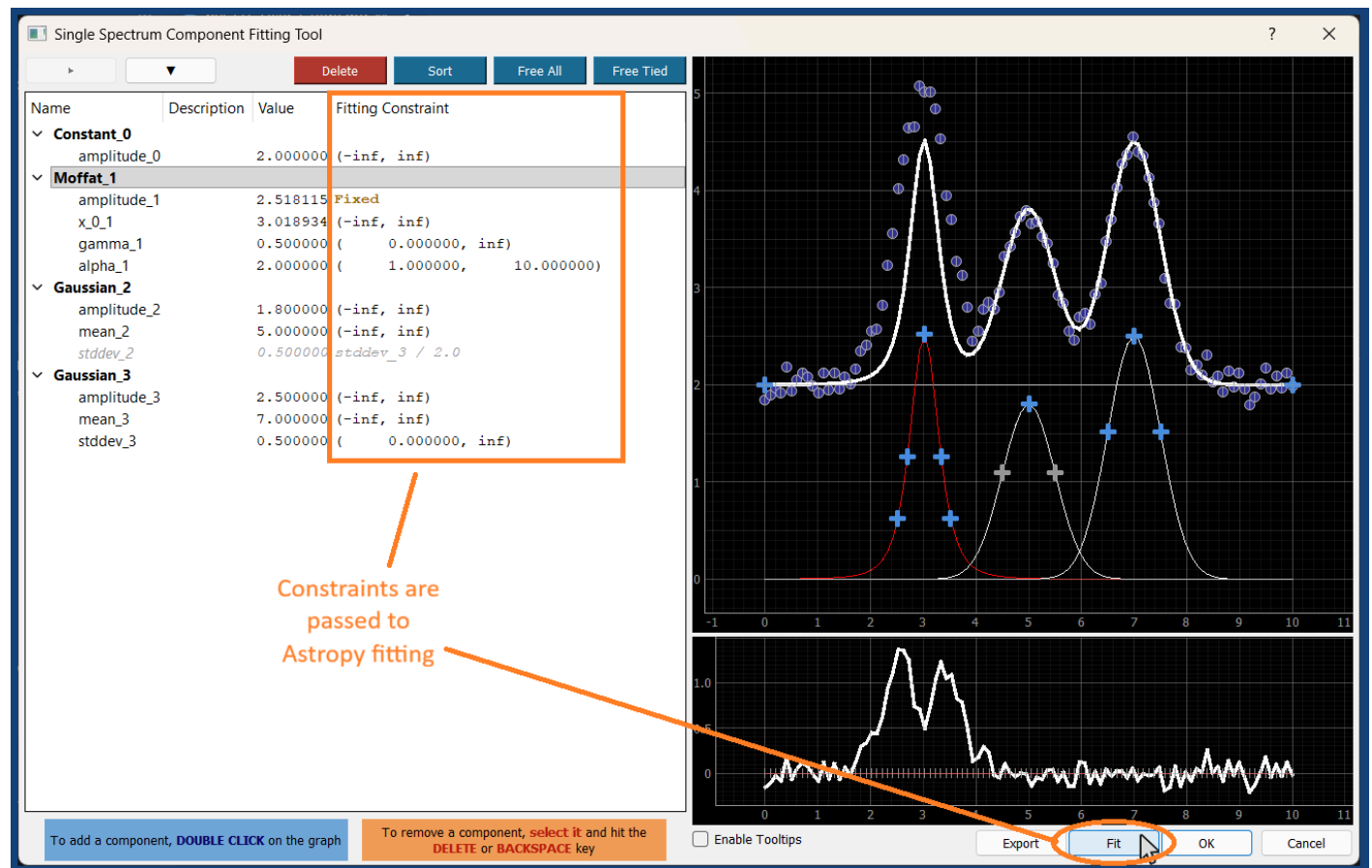


The descriptive name appears in the **Description** column in the Tree view and is saved with your model.



Finding the Best Fit

Click the **Fit** button to perform least-squares fitting using Astropy's Trust Region Reflective algorithm (TRFLSQFitter).



The fitter respects all parameter constraints:

- Free parameters within their bounds
- Fixed parameters at their set values
- Tied parameters maintaining their relationships

Model Output Options

Returning the Model

When you click **OK**, the current model is returned as an Astropy Model object:

```
model = pycfit.cfit_gui(wavelength, intensity, uncertainty=uncertainty)
# model is now an Astropy CompoundModel object
```

If you click **Cancel**:

- If an initial model was provided, it's returned unchanged
- Otherwise, **None** is returned

Exporting Models

Click the **Export** button to save your model:



Export options:

1. **Python file (.py)**: Creates a human-readable Python module with a `define_model()` function

```
import mymodel
model = mymodel.define_model()
```

2. **Pickle file (.pkl)**: Binary format for quick loading

```
import pickle
with open('mymodel.pkl', 'rb') as f:
    model = pickle.load(f)
```

The Python file export includes:

- All component parameters and their values
- Parameter constraints (fixed, bounds, tied expressions)
- Descriptive component names

Example exported Python file structure:

```
from astropy.modeling.models import Gaussian1D, Const1D

def define_model():
    """
    Model created with pyCFIT
    """
    # Component 0: Constant (Background)
    comp_0 = Const1D(amplitude=0.35)
    comp_0.amplitude.fixed = False

    # Component 1: Gaussian (O III emission)
    comp_1 = Gaussian1D(amplitude=7.15, mean=972.5, stddev=0.8)
    comp_1.amplitude.fixed = False
    comp_1.mean.bounds = (970, 975)

    return comp_0 + comp_1
```

Raster (Grid) Fitting

Creating a Grid Object

After developing a model with single spectrum fitting, extend it across a raster (2D grid) of spectra:

```
import pycfit

# model: Astropy Model
# wavelength: 1D array of wavelength values, shape (N,)
# intensity: 3D array, shape (N, X, Y)
# uncertainty: 3D array (optional), shape (N, X, Y)
# mask: 3D boolean array (optional), shape (N, X, Y)
# - marks bad data points
# - use driven by bad data or other data filters
# user_mask: 2D boolean array (optional), shape (X, Y)
# - marks points to skip
# - use driven by User not wanting to fit the spectrum at a given location

# Option 1: Create Grid without GUI, fit manually
GM = pycfit.cfit_grid(model, wavelength, intensity,
                      uncertainty=uncertainty, mask=mask)
GM.fit() # Fit all points
results = GM.get_results()

# Option 2: Create Grid with GUI (auto-fits)
GM = pycfit.cfit_grid_gui(model, wavelength, intensity,
                          uncertainty=uncertainty,
                          mask=mask, user_mask=user_mask)
```

Example with provided data and model:

```
# Get a stored model
import mymodel
model = mymodel.define_model()

# Get example grid data (small 20x20 subset)
wavelength, intensity, uncertainty, mask = pycfit.data.get_example_grid_spectra()

# Create and open GUI
GM = pycfit.cfit_grid_gui(model, wavelength, intensity,
                          uncertainty=uncertainty, mask=mask)
```

Loading from saved ASDF file:

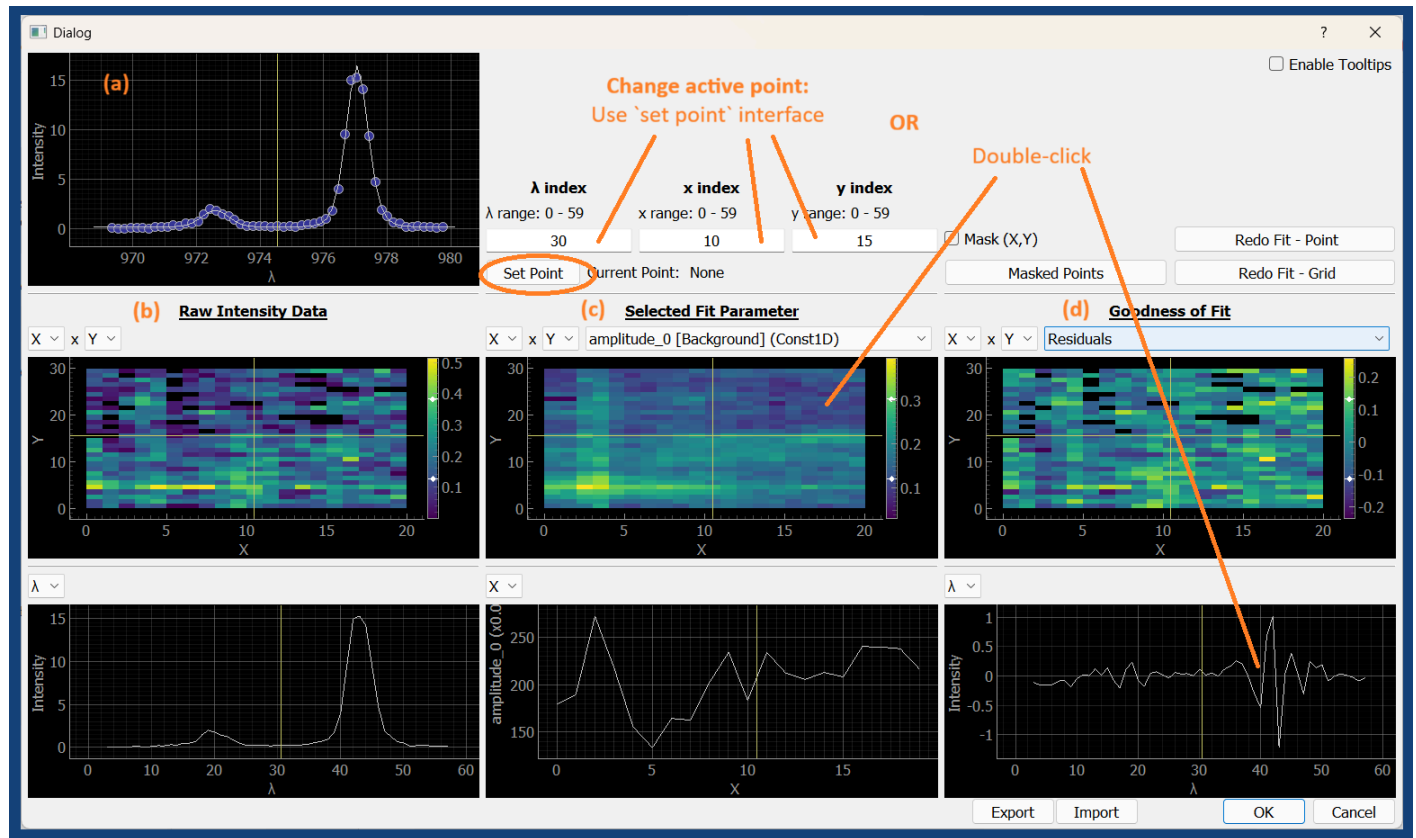
```
# Load a previously saved Grid (includes model)
GM = pycfit.cfit_grid_gui(file='my_grid_fit.asdf')
```

Loading from Instantiated Grid Object:

```
# Visualize an existing Grid (auto-fits if not already fitted)
GM = pycfit.cfit_grid_gui(GM)
```

Grid Interface Overview

The Grid GUI has a multi-panel layout for comprehensive data exploration:



Panel layout:

Panel	Description
(a) Top left	Spectrum overview: wavelength vs. intensity at selected (X, Y) with fit overlay
(b) Left column	Intensity data exploration across the cube
(c) Center column	Model parameters visualization across the grid
(d) Right column	Goodness of Fit visualization (residuals, DOF, χ^2)

Each column has:

- **Top panel:** 2D slice (spectrogram or map)
- **Bottom panel:** 1D line plot
- **Axis selectors:** Dropdown menus to choose which dimension to display

Navigating the Grid

Change the active point (X, Y, λ):

1. **Double-click** in any 2D or 1D panel
2. **Or** enter indices manually and click **Set Point**

The active point is marked with **crosshairs** in all panels and updates all views simultaneously.

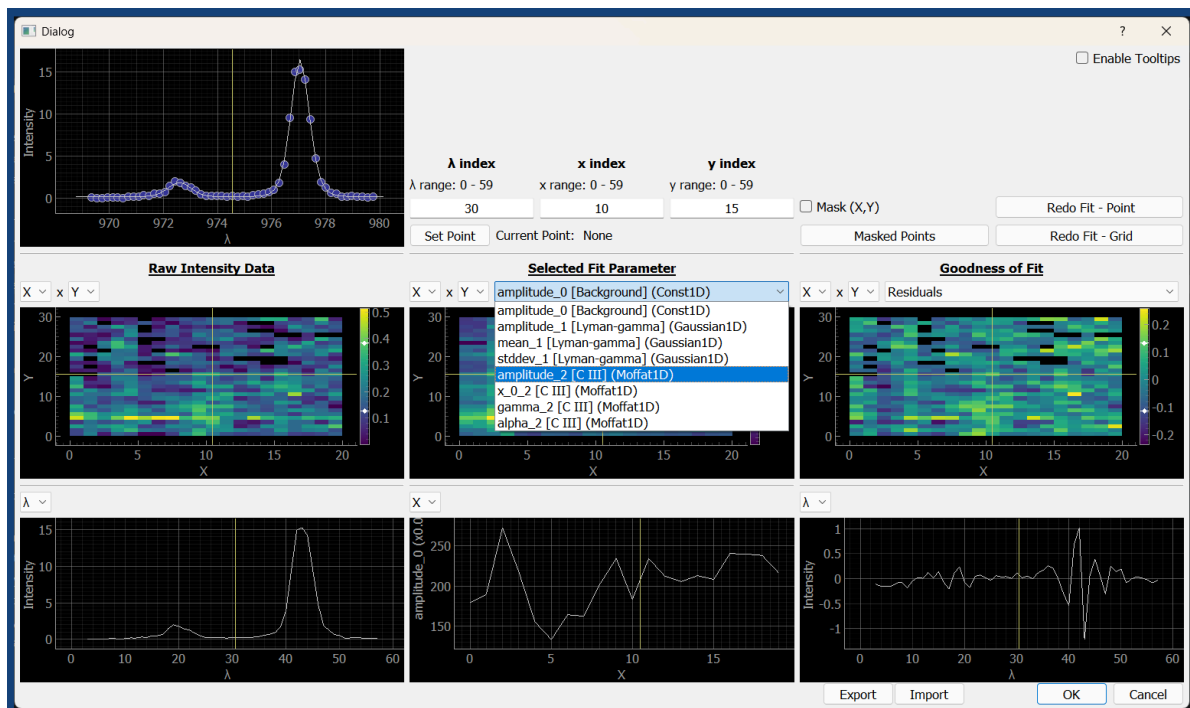
Column Details

Left Column: Intensity

- Displays measured intensity values
- (b1) 2D slice: Intensity at selected wavelength across X, Y
- (b2) 1D plot: Intensity spectrum at selected X, Y

Center Column: Model Parameters

- Use the dropdown menu to select which parameter to visualize
- (c1) 2D map: Parameter values across the grid
- (c2) 1D plot: Parameter value vs. wavelength (fixed), X, or Y



Each entry shows:

- Parameter name (e.g., **amplitude_1**)
- Component description [in brackets]
- Component type (Gaussian1D, Moffat1D, etc.)

Right Column: Goodness of Fit

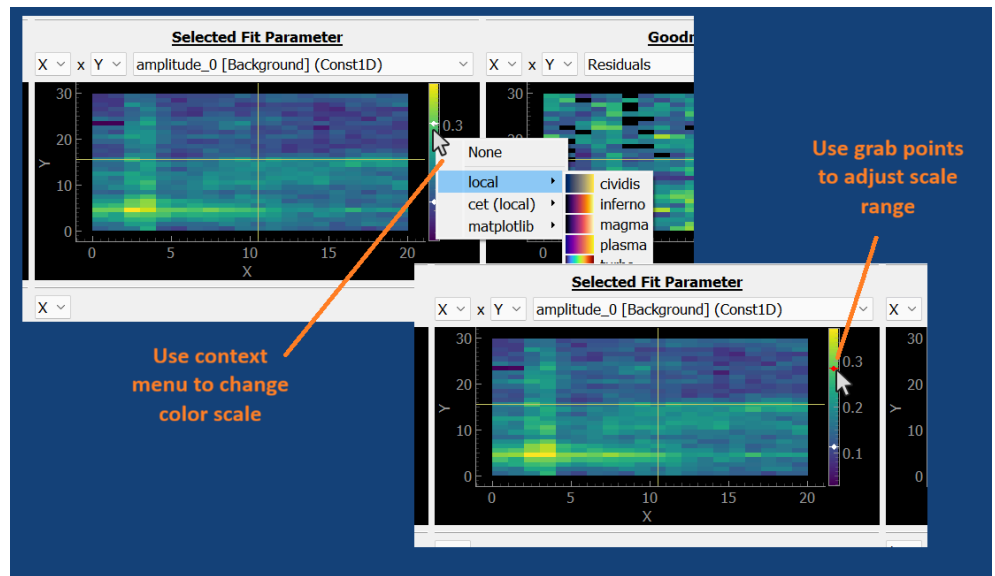
- Use the dropdown to select visualization mode:
 - **Residuals** (default)
 - **DOF** (Degrees of Freedom)
 - **Reduced Chi-Sq** (χ^2/DOF)

Inspecting and Modifying Fits

Adjusting Color Scales

Each 2D spectrogram has an interactive **color bar**:

1. **Right-click** the color bar to select a color table
2. **Click and drag** the white handles to adjust the range



Masking Points

To exclude a point from results:

1. Navigate to the point (double-click or Set Point)
2. Check the **Mask (X,Y)** checkbox
3. The point will be excluded from returned results

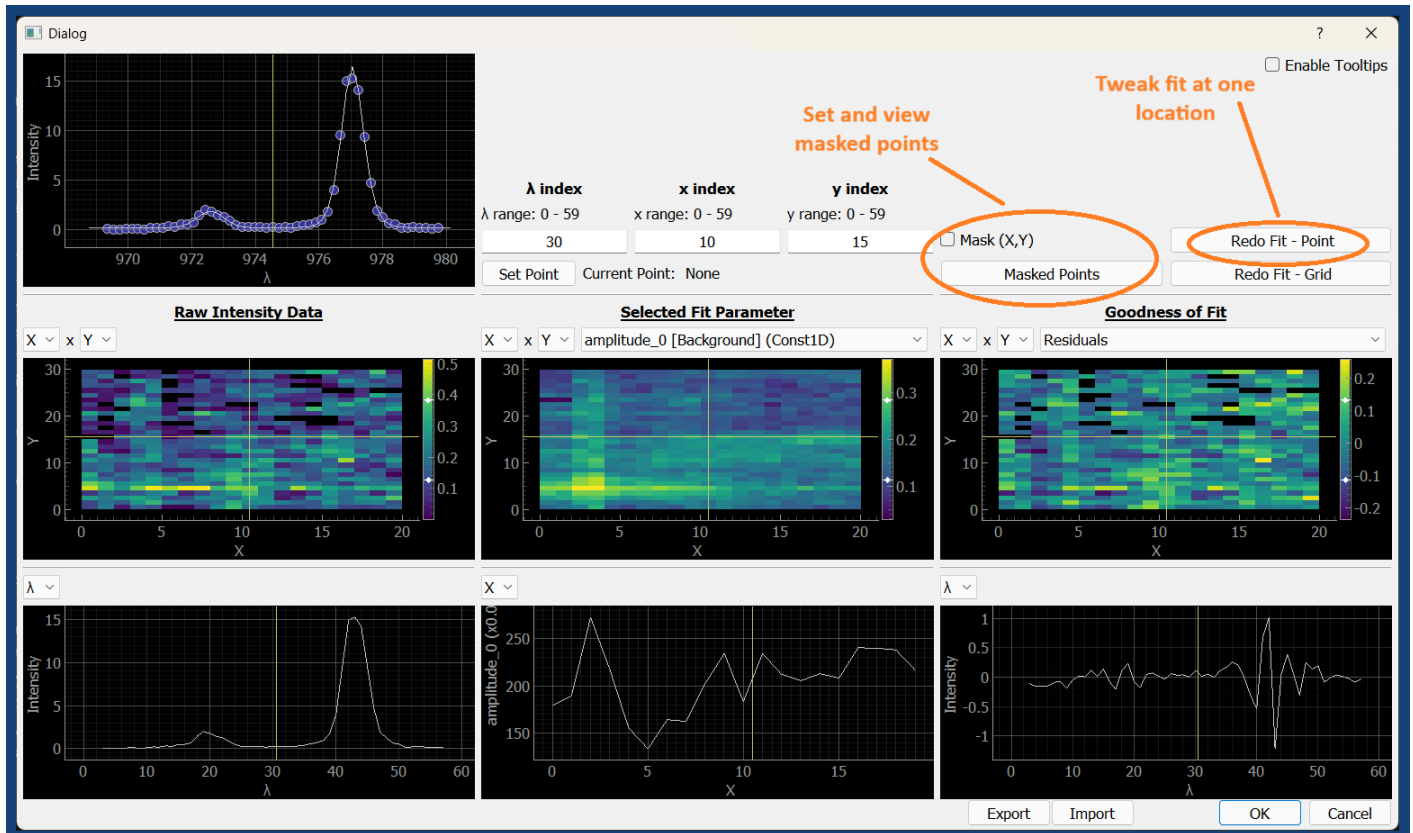
To view all masked points:

- Click **Show Masked Points**

Refitting a Single Point

If a fit at a particular location needs adjustment:

1. Navigate to that point
2. Click **Redo Fit – Point**



This opens a limited single spectrum fitting dialog where you can:

- Adjust initial parameter values
- Change bounds or constraints
- Refit with new settings

Note:

You cannot add, remove or sort Components (to keep the grid model consistent), but you can effectively disable a Component by fixing all its Parameters to zero.

These newly fitted Parameter values are included with the results. Changes to the *constraints*, however, do not persist outside of this interaction. If you re-fit the entire grid or re-fit this specific point again, the constraints will revert to those in the base model.

Refitting the Entire Grid

Click **Redo Fit – Grid** to refit all unmasked points with current Parameters as initial values.

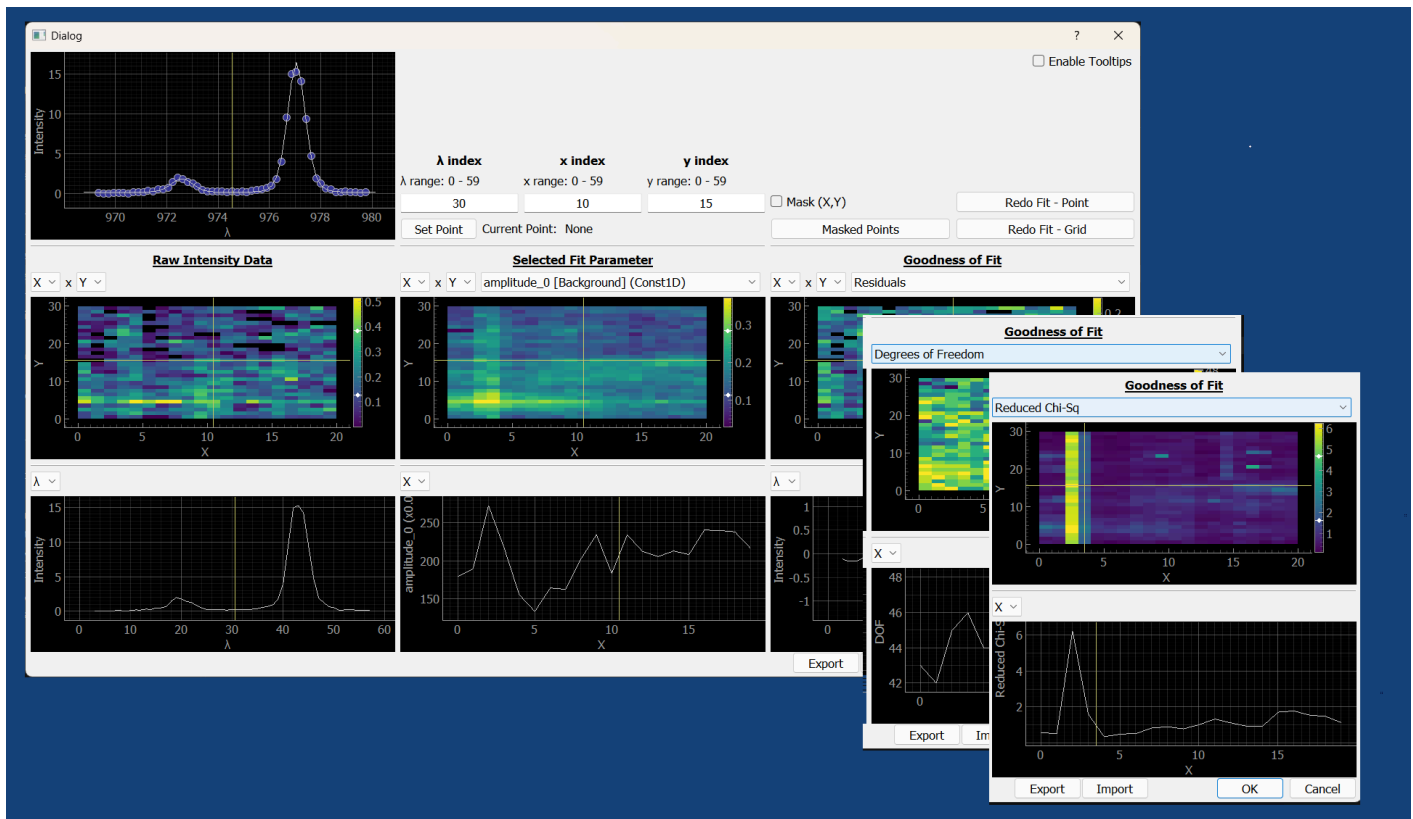
Goodness of Fit Visualization

The right column provides three visualization modes for assessing fit quality:

Residuals Mode (Default)

Displays the 3D residual cube (data - model):

- (d1) 2D slice: Residuals at selected wavelength
- (d2) 1D plot: Residual spectrum at selected point
- Axis selectors: Choose X, Y, or λ for slicing



Degrees of Freedom Mode

Displays a 2D map of degrees of freedom for each fit:

- Shows how many free parameters were used at each location
- Useful for identifying points where the fit may be over/under-constrained

Reduced Chi-Squared Mode

Displays a 2D map of reduced χ^2 (χ^2/DOF) values where:

$$\chi^2 = \sum_{i=1}^N \left(\frac{\text{Observed}_i - \text{Fit}_i}{\text{uncertainty}_i} \right)^2$$

- Values near 1.0 indicate good fits
- Values $\gg 1.0$ suggest poor fits or underestimated uncertainties
- Values $\ll 1.0$ suggest overestimated uncertainties

Note: In Degrees of Freedom and Reduced Chi-Sq modes, the wavelength (λ) axis selector is hidden since these are 2D quantities, not 3D.

Working with Results

The `get_results()` Method

Retrieve fit results as a dictionary:

```
results = GM.get_results()
print(results.keys())
```

Returned dictionary keys:

Key	Type	Shape	Description
<code>base_model</code>	Astropy Model	-	The model used for fitting
<code>mask</code>	boolean array	(X, Y)	User-defined masked points
<code>fit_info</code>	FitInfoArrayContainer	(X, Y)	Covariance and fit information from Astropy
<code>std_err</code>	dict of float arrays	(X, Y) per parameter	Standard error for each fitted parameter
<code>dof</code>	int array	(X, Y)	Degrees of freedom
<code>red_chi_sq</code>	float array	(X, Y)	Reduced χ^2 at each point
<code><parameter_name></code>	float array	(X, Y)	Fitted value for each parameter (e.g., <code>amplitude_1</code> , <code>mean_2</code>)

Example usage:

```
results = GM.get_results()

# Get Gaussian amplitude across the grid
amplitude_map = results['amplitude_1'] # Shape: (X, Y)

# Get standard errors
amplitude_errors = results['std_err']['amplitude_1']

# Get reduced chi-squared
chi_squared_map = results['red_chi_sq']

# Get masked points
masked_locations = results['mask']
```

Saving and Loading with ASDF

Saving:

1. Click **Export** in the Grid GUI
2. Choose a filename and location
3. The entire Grid object is saved in ASDF format

Or programmatically:

```
GM.save_asdf('my_grid_fit.asdf')
```

The ASDF file stores:

- Complete Grid object state
- All fitted parameters at every point
- Masks, uncertainties, and fit statistics
- Base model with all constraints



Loading:

1. Click **Import** in the Grid GUI
2. Choose an **.asdf** filename and location
3. The entire Grid object is loaded and the GUI refreshed

Or programmatically:

```
import pycfit

# Load Grid object from file
GM = pycfit.load_asdf('my_grid_fit.asdf')

# Open in GUI
GM = pycfit.cfit_grid_gui(GM)

# Or work with it programmatically
results = GM.get_results()
```

Advanced Features

Working with Astropy Units

PyCFIT accepts Astropy Quantities with physical units:

```
import astropy.units as u

# Get example data with units
wavelength, intensity, uncertainty =
pycfit.data.get_example_single_spectrum(units=True)

# wavelength is now an Astropy Quantity
print(wavelength.unit) # e.g., Angstrom

# Use with single spectra CFIT
model = pycfit.cfit(wavelength, intensity, uncertainty=uncertainty)
model = pycfit.cfit_gui(wavelength, intensity, uncertainty=uncertainty)

# Grid data with units
wavelength, intensity, uncertainty, mask =
pycfit.data.get_example_grid_spectra(units=True)
GM = pycfit.cfit_grid(model, wavelength, intensity,
                      uncertainty=uncertainty, mask=mask)
```

Custom Fitter Parameters

Pass additional parameters to the underlying Astropy fitter:

```
# Single spectrum fitting
FF = pycfit.cfit(wavelength, intensity, uncertainty=uncertainty, function=model)
FF.fit(maxiter=500, acc=1e-10)

# Grid fitting
GM = pycfit.cfit_grid(model, wavelength, intensity,
                      uncertainty=uncertainty, mask=mask)
GM.fit(maxiter=200, acc=1e-8)
```

Common parameters:

- `maxiter`: Maximum number of iterations (default: 100)
- `acc`: Accuracy for convergence (default: 1e-7)

See [Astropy TRFLSQFitter documentation](#) for more options.

Library API Usage

PyCFIT follows a Model-View design pattern. The GUI is a view on underlying data objects (`FunctionFitter` and `Grid`) that can be used independently for scripting and automation.

Single Spectrum API

Creating a FunctionFitter

```
import pycfit

# Create FunctionFitter object with only data
FF = pycfit.cfit(wavelength, intensity, uncertainty=uncertainty)

# Or with an initial model included
FF = pycfit.cfit(wavelength, intensity, uncertainty=uncertainty, function=model)
```

Adding Components

```
# Add components sequentially
FF.add_component('Constant')
FF.add_component('Gaussian')
FF.add_component('Moffat')
```

Supported component strings: 'Constant', 'Linear', 'Quadratic', 'Gaussian', 'Moffat', 'Voigt'

Modifying Parameters

Use `get_model()` and `update_model()` methods to retrieve and modify parameters.

```
# Get the compound model
M = FF.get_model()
print(M)

# Modify parameter values
M.amplitude_0 = 0.5 # Constant amplitude
M.mean_1 = 976      # Gaussian mean
M.stddev_1 = 0.8    # Gaussian stddev

# Update the model
FF.update_model(M)
```

```
# Verify changes
print(FF.get_model())
```

Parameter Identifiers

Each parameter has a trailing numeric tag corresponding to its component ID:

Component Added	ID	Parameters
Constant (1st)	0	<code>amplitude_0</code>
Gaussian (2nd)	1	<code>amplitude_1, mean_1, stddev_1</code>
Moffat (3rd)	2	<code>amplitude_2, x_0_2, gamma_2, alpha_2</code>

Deleting Components

```
# Delete component by ID
FF.delete_component(1) # Deletes the Gaussian

# IDs are reassigned to be contiguous
# The Moffat (formerly ID=2) is now ID=1
```

Fitting

```
# Perform the fit
FF.fit()

# Get the fitted model
fitted_model = FF.get_model()
print(fitted_model)
```

Retrieving Fit Information

```
# Chi-squared value
chi_sq = FF.get_chi_sq()
print(f"Chi-squared: {chi_sq}")

# Detailed fit information from Astropy
fit_info = FF.get_fit_info()
print(fit_info)
```

The `fit_info` contains:

- `message`: Termination message
- `success`: Boolean indicating convergence
- `cost`: Final cost function value
- `x`: Final parameter values
- `jac`: Jacobian matrix
- `param_cov`: Parameter covariance matrix

See the Astropy documentation for more information about the [TRFLSQFitter](#) and the related SciPy documentation for further description of the `fit_info` returned.

Grid API

Creating a Grid

```
import pycfit

# Check data shapes
print(f"Wavelength shape: {wavelength.shape}") # (N,)
print(f"Intensity shape: {intensity.shape}")    # (N, X, Y)

# Create Grid object (doesn't fit by default)
GM = pycfit.cfit_grid(model, wavelength, intensity,
                      uncertainty=uncertainty, mask=mask)

# Check grid shape
print(f"Grid shape: {GM.shape}") # (X, Y)
```

Fitting the Grid

```
# Fit all points
GM.fit()

# Check if fitted
print(GM.is_fitted()) # True

# Clear all fits
GM.clear_fit()
print(GM.is_fitted()) # False

# Refit
GM.fit()
```

Accessing Individual Points

Use 2-tuple indexing (X, Y) to access/modify parameters at specific locations:

```
# View model at a specific point
key = (4, 2)
print(GM[key])

# View parameter names
print(GM.param_names)
# ['amplitude_0', 'amplitude_1', 'mean_1', 'stddev_1', ...]

# Get parameter value at a point
print(GM.amplitude_0.value[key])

# Modify initial condition
GM.amplitude_0.value[key] = 0.5

# Refit just that point
GM.fit(key)
```

Working with Results

```
# Get results dictionary
results = GM.get_results()

# Access fitted parameters
amplitude_map = results['amplitude_1'] # (X, Y) array
mean_map = results['mean_1']

# Access goodness of fit metrics
reduced_chi_sq = results['red_chi_sq'] # (X, Y)
dof = results['dof'] # (X, Y)

# Access uncertainties
amplitude_errors = results['std_err']['amplitude_1'] # (X, Y)

# Get the base model
base_model = results['base_model']

# Get mask
user_mask = results['mask'] # (X, Y) boolean array
```

Saving and Loading

```
# Save to ASDF file
GM.save_asdf('my_grid_fit.asdf')

# Load from ASDF file
GM = pycfit.load_asdf('my_grid_fit.asdf')

# Continue working with loaded Grid
if not GM.is_fitted():
    GM.fit()

results = GM.get_results()
```

Tips and Best Practices

Supported Component Models

PyCFIT supports six component model types from Astropy:

Component	Astropy Model	Parameters	Description
Constant	Const1D	amplitude	Constant offset
Linear	Linear1D	slope, intercept	Linear baseline
Quadratic	Polynomial1D (degree=2)	c0, c1, c2	Quadratic polynomial
Gaussian	Gaussian1D	amplitude, mean, stddev	Gaussian profile
Moffat	Moffat1D	amplitude, x_0, gamma, alpha	Moffat profile (heavy-tailed)
Voigt	Voigt1D	x_0, amplitude_L, fwhm_L, fwhm_G	Voigt profile (convolution of Gaussian and Lorentzian)

When to Use Each Component

Use Case	Recommended Components
Continuum background	Constant, Linear, or Quadratic
Emission lines	Gaussian (thermal broadening) or Voigt (pressure + thermal)
Absorption lines	Gaussian or Voigt
Broad wings / PSF effects	Moffat
Multiple overlapping lines	Combination of Gaussian/Voigt components

Single Spectrum Fitting

1. **Start with averaged data:** If preparing a model for grid fitting, use an average spectrum from your raster data
2. **Add components in logical order:** Background components first, then emission/absorption features
3. **Use the Sort button:** After adding multiple emission lines, sort them by location for easier management
4. **Set reasonable bounds:** Prevent unphysical parameter values with bounds
5. **Use tied parameters** for physical constraints (e.g., same width for isotope lines)
6. **Name your components:** Descriptive names make complex models easier to understand
7. **Test your model:** After fitting, visually inspect residuals for systematic features

Grid Fitting

1. **Start with a good single-spectrum model:** Time spent optimizing the initial model saves time later
2. **Use masks liberally:** Exclude bad pixels with `mask`, exclude spatial regions with `user_mask`
3. **Check goodness of fit maps:** Use Degrees of Freedom and χ^2 visualizations to identify problematic regions
4. **Refit suspicious points:** Use "Redo Fit – Point" for locations with poor fits
5. **Save intermediate results:** Export to ASDF regularly to preserve your work

Parameter Constraints

1. **Free parameters by default:** Let the fitter find the best values
2. **Add bounds when needed:** Prevent negative widths, constrain line positions
3. **Fix known values:** If you know a parameter (e.g., instrumental resolution), fix it
4. **Tie related parameters:** Physical relationships (e.g., doublet ratios) should be enforced

Performance

1. **Reduce grid size during development:** Use a small subset for testing
 2. **Use the example data:** `pycfit.data.get_example_grid_spectra()` provides quick test data
 3. **Close the GUI between major operations:** Large grids benefit from fresh starts
 4. **Monitor memory usage:** Very large grids (>1000×1000) may require chunking
-

Troubleshooting

Common Issues

GUI doesn't open on MacOS

Problem: GUI window doesn't appear or crashes immediately

Solution: Run pyCFIT from within an ipython session with Qt GUI magic:

```
ipython --gui=qt
```

Then:

```
import pycfit  
model = pycfit.cfit_gui(wavelength, intensity, uncertainty=uncertainty)
```

Fit doesn't converge

Problem: Fitter returns "maxiter reached" or produces nonsensical results

Solutions:

1. Check initial parameter values are reasonable
2. Increase `maxiter`: `FF.fit(maxiter=500)`
3. Ensure data isn't pathological (all zeros, all NaN, etc.)
4. Try different bounds on parameters
5. Simplify the model (fewer components)

Memory error with large grids

Problem: Python crashes or reports memory error

Solutions:

1. Process the grid in spatial chunks
2. Reduce the uncertainty array size (downsample or use constant uncertainty)
3. Close other applications
4. Use a machine with more RAM

Tied parameters create circular dependencies

Problem: Error message about circular ties (A tied to B, B tied to A)

Solution: Review tied expressions to ensure no circular references:

```
# Bad: Circular dependency
# amplitude_1 tied to "amplitude_2"
# amplitude_2 tied to "amplitude_1"

# Good: One-way dependency
# amplitude_1 tied to "0.5 * amplitude_2"
# amplitude_2 is free
```

Parameter tied expression not working

Problem: Tied parameter doesn't update as expected

Solution: Ensure expression is syntactically correct:

```
# Correct expressions:
"amplitude_2"           # Simple reference
"2.0 * amplitude_2"     # Multiplication
"amplitude_2 / 3.14"    # Division
"amplitude_2 + mean_1"  # Addition

# Invalid expressions:
"amplitude_2 * "        # Incomplete expression
"= amplitude_2 - 5"     # Don't incorporate assignment expression (=)
```

Grid fitting is very slow

Problem: Fitting a large grid takes too long

Solutions:

1. Reduce the number of free parameters (fix or tie some)
2. Reduce `maxiter` for initial exploration
3. Use a smaller subset for testing

Can't load saved ASDF file

Problem: `load_asdf()` fails with an error

Solutions:

1. Ensure pyCFIT version matches (ASDF format may change between versions)
2. Check that file is named with an `.asdf` suffix
3. Check file isn't corrupted (try a recent backup)

Contact Information

Software Maintenance

Ayris Narock

Email: ayris.a.narock@nasa.gov

NASA Official

Therese Kucera

Email: therese.a.kucera@nasa.gov

Documentation and Source

- **Source Code:** [NASA GitLab - PyCFIT](#)
- **PyPI Package:** <https://pypi.org/project/pycfits/>

License

This project is Copyright © National Aeronautics and Space Administration and licensed under the terms of the Apache Software License 2.0. This package is based upon the OpenAstronomy packaging guide which is licensed under the BSD 3-clause licence.

This user guide reflects pyCFIT version 1.1 and later. For older versions, some features may not be available.