The Journal of Open Source Software

pyMultiFit: A Python library for fitting data with multiple models

Syed Ali Mohsin Bukhari [1,2¶], **Iqra Siddique** [3,4,5], **and Asad Ali** [1,2]

1 Department of Applied Mathematics and Statistics, Institute of Space Technology, Islamabad 44000, Pakistan **2** Space and Astrophysics Research Lab (SARL), National Centre of GIS and Space Applications (NCGSA), Islamabad 44000, Pakistan **3** Gran Sasso Science Institute (GSSI), Viale Crispi 7, I-67100 L'Aquila, Italy **4** Dipartimento di Fisica, Università di Trento, Via Sommarive 14 I-38123 Trento, Italy **5** INFN Laboratori Nazionali del Gran Sasso, Via Acitelli 22, I-67100 Assergi, L'Aquila, Italy ¶ Corresponding author

DOI: 10.21105/joss.09295

Software

- Review
- Repository
- Archive

Editor: Arfon Smith

Reviewers:

- @KandiSanjana
- @KristinaGagalova

Submitted: 16 October 2025
Published: 24 October 2025

Summary

`pyMultiFit` is an open-source Python library that streamlines multi-model curve fitting for scientific applications. It provides a uniform API for both statistical and non-statistical models, with built-in support for common distributions such as Gaussian, Laplace, SkewedNormal, LogNormal, and Exponential. The library emphasizes extensibility through a lightweight `BaseFitter` interface, enabling users to define custom models with minimal boilerplate. In addition to fitting, `pyMultiFit` includes utilities for synthetic data generation and integrated statistical tools for constructing synthetic datasets.

Together, these features make `pyMultiFit` especially well-suited for researchers working with **signals**, **spectra**, or **experimental datasets** where multiple overlapping components are common.

Statement of Need

Data fitting is a cornerstone of experimental and analytical workflows across the sciences. Yet, widely used scientific libraries such as `numpy` (Harris et al., 2020) and `scipy` (Virtanen et al., 2020) provide only limited functionality for fitting **multiple models or model mixtures** concurrently. Researchers working with multi-component fits—common in domains such as signal analysis, spectroscopy, and high-energy physics—are often required to implement complex and repetitive boilerplate code to manage these workflows.

Several modern libraries have been developed to extend beyond these limitations. For example, `zfit` (Eschle et al., 2020) leverages `tensorflow` (Abadi et al., 2016) to support high-performance statistical fitting, `pyAutoFit` (Nightingale et al., 2021) facilitates hierarchical model construction, and specialized `Mixture-Models` (Kasa et al., 2024) packages provide probabilistic mixture fitting. While powerful, these tools are either tightly bound to heavy backends, optimized for niche applications, or difficult to adapt for general multimodel curve-fitting tasks.

`pyMultiFit` addresses this gap by providing a lightweight and extensible framework dedicated to multi-model fitting. It offers built-in support for combining multiple models, generating synthetic datasets, and incorporating user-defined models with minimal effort.

By streamlining multi-component fitting workflows, `pyMultiFit` enables researchers to more rapidly explore, test, and refine fitting strategies without the overhead of custom infrastructure.

Modules

The pyMultiFit library is organized into three main modules:

1. **distributions**
   Provides classes for statistical distributions, including common implementations such as Gaussian and Laplace via the utilities_d script.
   - Custom distributions can be defined by subclassing the BaseDistribution class. At minimum, users must implement the distribution formula in the pdf method.
   - For convenience, statistical distributions can also be instantiated using the from_scipy_params class method, which accepts the same parameter names as scipy.stats.
2. **fitters**
   Provides classes for fitting models to data.
   - Custom fitters can be created by subclassing the BaseFitter class.
   - Fitters can handle multiple use cases of the same model.
   - The MixedDataFitter class is included for fitting multiple models to heterogeneous datasets.
3. **generators**
   Provides tools for generating synthetic datasets.
   - Generators can accept subclasses of BaseDistribution through a dictionary interface, enabling flexible model generation.
   - A key utility is the multiple_models function which enables heterogeneous model data generation using BaseDistribution subclasses.

Highlights

pyMultiFit provides a consistent API for custom distributions, multi-model data generation, and heterogeneous fitting workflows.
This section highlights key features of the library with illustrative examples and figures.

Custom and scipy-compatible distributions

Users can define distributions by subclassing BaseDistribution or use the familiar scipy.stats parameterization. Figure 1 shows the comparison between a custom Gaussian distribution implemented in pyMultiFit and the equivalent scipy.stats distribution.

```python
import numpy as np
import scipy.stats as ss
from matplotlib import pyplot as plt

from pymultifit.distributions import GaussianDistribution

x = np.linspace(-15, 15, 100)

custom_distribution = GaussianDistribution(mu=-1, std=4, normalize=True).pdf(x)

scipy_distribution = ss.norm(loc=-1, scale=4).pdf(x)
custom_with_scipy = GaussianDistribution.from_scipy_params(loc=-1,
                                                            scale=4).pdf(x)

f, ax = plt.subplots(1, 2, figsize=(15, 5))
ax[0].plot(x, custom_distribution,
           label='custom distribution\nw/o\nscipy parametrization')
ax[0].plot(x, scipy_distribution, label='scipy distribution')
```

```
ax[1].plot(x, custom_with_scipy,
           label='custom distribution\nw\nscipy parametrization')
ax[1].plot(x, scipy_distribution, label='scipy distribution')

[i.set_xlabel('x') for i in ax]
[i.set_ylabel('pdf') for i in ax]
[i.set_title('Gaussian Distribution') for i in ax]
[i.grid(True, alpha=0.5, ls='--') for i in ax]
[i.legend(loc='best') for i in ax]

plt.tight_layout()
plt.show()
```

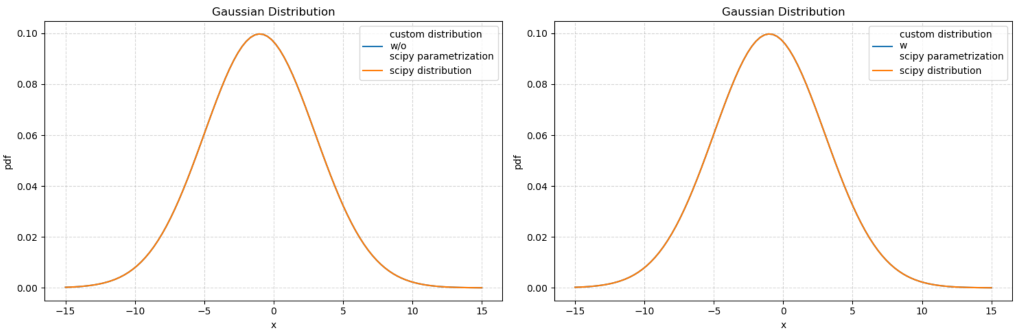

Figure 1: Comparison of the pdf of a Gaussian distribution with and without scipy parametrization

Flexible tools for generating synthetic datasets and fitting multiple models

Flexible tools allow researchers to generate synthetic datasets with multiple components and fit them with minimal boilerplate. Figure 2 demonstrates the fitting of a 3-component Gaussian mixture using `pyMultiFit`, with both individual components and the composite model shown.

```
import matplotlib.pyplot as plt
import numpy as np

from pymultifit.fitters import GaussianFitter
from pymultifit.generators import multi_gaussian

x = np.linspace(-10, 10, 10_000)

amp = np.array([4, 2, 6])
mu = np.array([-3, 0, 6])
std = np.array([1, 1, 0.3])
params = np.column_stack([amp, mu, std])

mg_data = multi_gaussian(x, params, noise_level=0.2, normalize=False)

# guess for the parameters
# note that the order of the parameters must match the order of the
# parameters in the distribution class
amp_guess = np.array([3, 1, 4])
mu_guess = np.array([-2, 0, 5])
std_guess = np.array([1, 0.5, 0.5])
```

```
params_guess = np.column_stack([amp_guess, mu_guess, std_guess])
mg_fitter = GaussianFitter(x, mg_data)
mg_fitter.fit(params_guess)
mg_fitter.plot_fit(show_individuals=True)
plt.show()
```

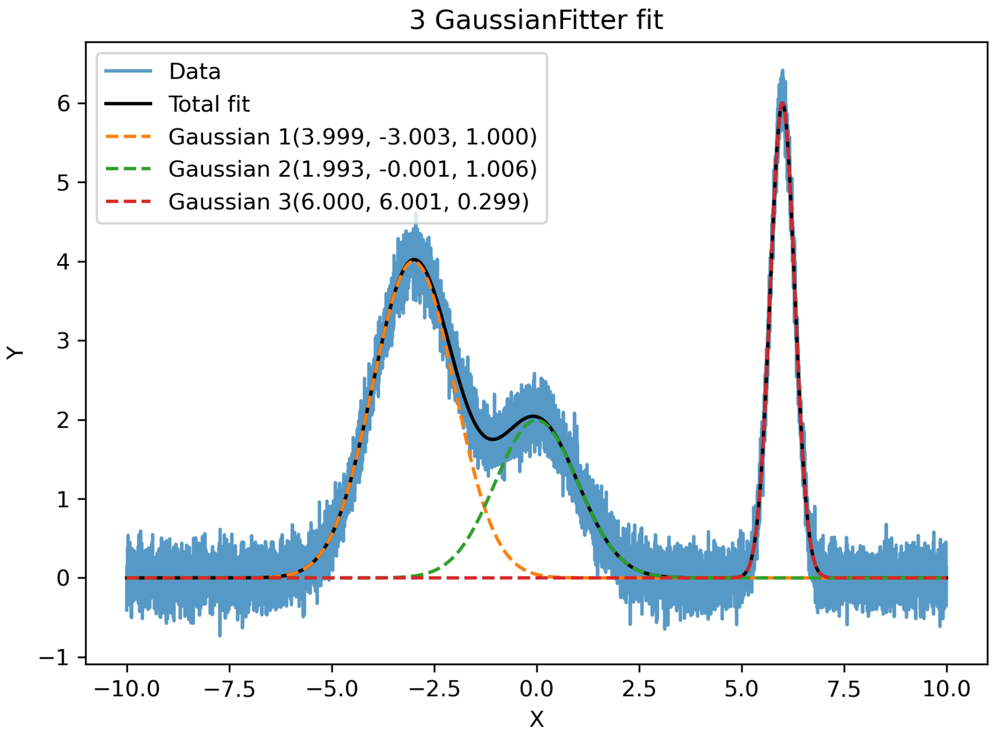

Figure 2: Example of a 3-component Gaussian model fit using the `GaussianFitter`

Flexible API for generating and fitting heterogeneous datasets.

The MixedDataFitter enables simultaneous fitting of heterogeneous models (e.g., Gaussian + SkewNormal + Linear), useful in complex workflows such as spectral modeling. Figure 3 illustrates a combined Gaussian, SkewNormal, and Linear model fit.

```
import matplotlib.pyplot as plt
import numpy as np

from pymultifit.distributions.utilities_d import (gaussian_pdf_,
                                                  skew_normal_pdf_, line)
from pymultifit.fitters import (MixedDataFitter, GaussianFitter,
                                SkewNormalFitter, LineFitter)
from pymultifit.generators import multiple_models

x = np.linspace(-10, 10, 10_000)

gauss = (10, -1, 0.2)
skewNorm = (3, 5, 0.2, 3)
lineParams = (-0.2, -0.3)
params = [gauss, skewNorm, lineParams]
```

```python
mixed_data = multiple_models(x, params,
                             model_list=['G', 'Sk', 'L'],
                             mapping_dict={'G': gaussian_pdf_,
                                           'Sk': skew_normal_pdf_,
                                           'L': line},
                             noise_level=0.2)

guess = [(8, 0, 1), (8, 3, 0, 2), (1, 0)]
mixed_fitter = MixedDataFitter(x, mixed_data, model_list=['G', 'Sk', 'L'],
                               model_dictionary={'G': GaussianFitter,
                                                 'Sk': SkewNormalFitter,
                                                 'L': LineFitter})
mixed_fitter.fit(guess)
mixed_fitter.plot_fit(show_individuals=True)
plt.show()
```

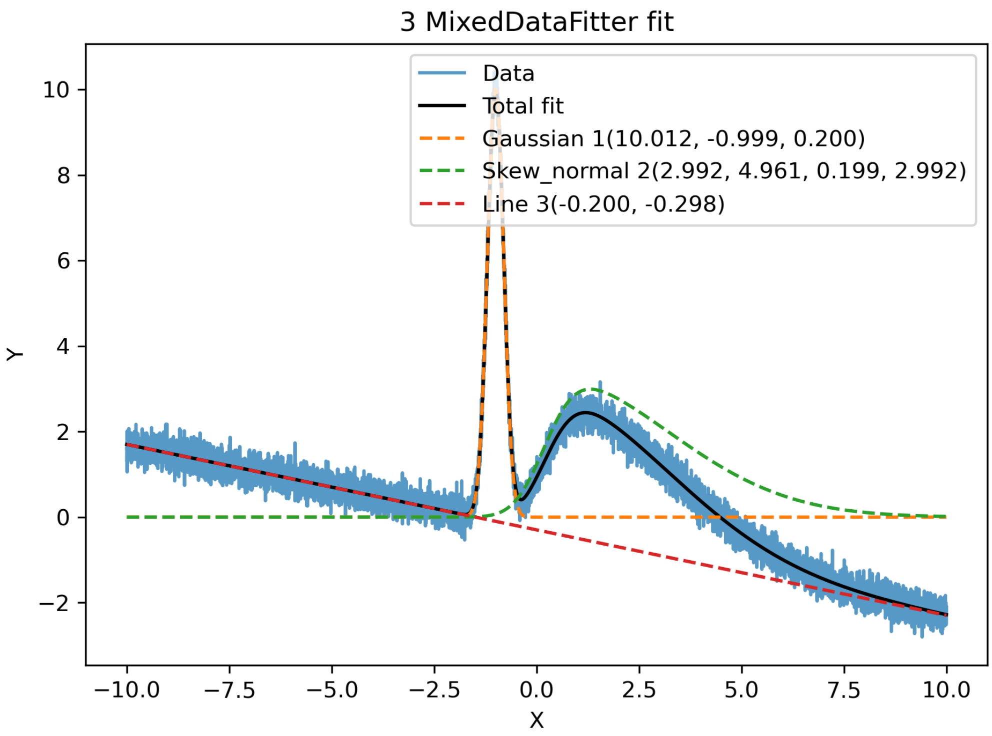

Figure 3: Demonstration of the MixedDataFitter for fitting heterogeneous models

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., … Zheng, X. (2016). TensorFlow: A system for large-scale machine learning. *arXiv e-Prints*, arXiv:1605.08695. https://doi.org/10.48550/arXiv.1605.08695

Eschle, J., Puig Navarro, A., Silva Coutinho, R., & Serra, N. (2020). zfit: Scalable pythonic fitting. *SoftwareX*, *11*, 100508. https://doi.org/10.1016/j.softx.2020.100508

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., … Oliphant, T. E. (2020). Array programming with NumPy. *585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Kasa, S. R., Yijie, H., Kasa, S. K., & Rajan, V. (2024). Mixture-Models: a one-stop Python Library for Model-based Clustering using various Mixture Models. *arXiv e-Prints*, arXiv:2402.10229. https://doi.org/10.48550/arXiv.2402.10229

Nightingale, James., Hayes, R., & Griffiths, M. (2021). PyAutoFit: A Classy Probabilistic Programming Language for Model Composition and Fitting. *The Journal of Open Source Software*, *6*(58), 2550. https://doi.org/10.21105/joss.02550

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., … SciPy 1. 0 Contributors. (2020). SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature Methods*, *17*, 261–272. https://doi.org/10.1038/s41592-019-0686-2