

# <sup>1</sup> MultiMin: Mixture-of-Gaussians fitting and visualization for multivariate data and single-valued functions

<sup>4</sup> Jorge I. Zuluaga  

<sup>5</sup> 1 SEAP/FACoM, Instituto de Física - FCEN, Universidad de Antioquia, Colombia  Corresponding author

DOI: [10.xxxxxx/draft](https://doi.org/10.xxxxxx/draft)

## Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Editor: [Open Journals](#) 

## Reviewers:

- [@openjournals](#)

Submitted: 01 January 1970

Published: unpublished

## License

<sup>16</sup> Authors of papers retain copyright  
<sup>17</sup> and release the work under a  
<sup>18</sup> Creative Commons Attribution 4.0  
<sup>19</sup> International License ([CC BY 4.0](#)).  
<sup>20</sup>

## <sup>7</sup> Summary

<sup>8</sup> MultiMin is a Python package for fitting and analyzing *Mixtures of Gaussians* (MoGs, also  
<sup>9</sup> known as Gaussian mixture models) in one or many dimensions. The package provides (i) a  
<sup>10</sup> compact parameterization of multivariate Gaussian mixtures, (ii) numerical maximum-likelihood  
<sup>11</sup> fitting tools for data samples (including partially truncated domains), and (iii) publication-ready  
<sup>12</sup> visualizations through a unified corner-plot-like interface. In addition, MultiMin includes an  
<sup>13</sup> experimental workflow to fit *single-valued functions* (e.g., time-series, spectra, or model curves)  
<sup>14</sup> using MoG basis functions, returning interpretable diagnostics such as  $R^2$  and goodness-of-fit  
<sup>15</sup> measures.

## <sup>16</sup> Statement of need

Gaussian mixture models are a standard and versatile tool for density estimation, clustering, and probabilistic modeling because they can approximate complex continuous distributions through weighted sums of multivariate Gaussians ([Bishop, 2006](#); [McLachlan & Peel, 2000](#)). In research workflows, however, practitioners often need more than a black-box mixture estimator: they need a parameterization that is convenient for scientific interpretation, explicit control over covariance structure, support for bounded (truncated) variables, and a visualization layer that can compare samples and fitted models consistently across dimensions.

MultiMin addresses these needs by providing a research-oriented API centered on three core components:

- `MixtureOfGaussians`: definition, sampling, evaluation, and serialization of MoGs.
- `FitMoG` / `FitFunctionMoG`: numerical fitting tools for (i) multivariate samples via maximum-likelihood minimization and (ii) single-valued functions (e.g., spectra) via nonlinear least squares, both exposed through a consistent, data-first interface with many optional controls (domains, bounds, optimizer options, diagnostics).
- `MultiPlot`: a consistent grid visualization system for samples, PDFs, histograms, contours, and optional marginals (i.e., a unified alternative to piecemeal corner plotting).

The package is implemented in Python and integrates with the scientific Python ecosystem ([NumPy \(\[Harris et al., 2020\]\(#\)\)](#), [SciPy \(\[Virtanen et al., 2020\]\(#\)\)](#), and [Matplotlib \(\[Hunter, 2007\]\(#\)\)](#)). It is distributed as open source under the AGPLv3 license and includes extensive example notebooks that can be executed end-to-end.

## <sup>37</sup> State of the field

<sup>38</sup> Several mature libraries provide Gaussian mixture models, most prominently scikit-learn's  
<sup>39</sup> GaussianMixture estimator ([Pedregosa et al., 2011](#)). Such tools are excellent general-  
<sup>40</sup> purpose solutions for machine learning pipelines, but they typically emphasize predictive  
<sup>41</sup> usage and integration rather than scientific interpretability and publication-centric workflows.  
<sup>42</sup> In practice, many research users still need to build a layer around these estimators to (i) express  
<sup>43</sup> domain-specific constraints (e.g., partially bounded variables), (ii) extract explicit parametric  
<sup>44</sup> representations for reporting, and (iii) generate consistent multivariate diagnostic figures that  
<sup>45</sup> combine samples and fitted densities.

<sup>46</sup> MultiMin complements this landscape by focusing on scientific workflows:

- <sup>47</sup> **▪ Interpretability-first parameterization:** MoG components can be accessed and reported  
<sup>48</sup> as weights, means, standard deviations, correlations, and full covariance matrices.
- <sup>49</sup> **▪ Partially truncated domains:** fitting can be performed while enforcing bounded support  
<sup>50</sup> for selected variables (common in physical quantities constrained to intervals), with the  
<sup>51</sup> truncation explicitly represented in the fitted model.
- <sup>52</sup> **▪ Unified diagnostic plotting:** the same high-level plotting interface can overlay data,  
<sup>53</sup> mixture PDFs, contours, histograms, and marginals across dimensions, facilitating  
<sup>54</sup> reproducible figures for papers.

## <sup>55</sup> Theoretical background

<sup>56</sup> **Mixture representation.** A MoG with  $M$  components in  $k$  dimensions is represented as

$$f(\tilde{x}) = \sum_{i=1}^M w_i \mathcal{N}_k(\tilde{x}; \tilde{\mu}_i, \Sigma_i),$$

<sup>57</sup> with weights  $\sum_i w_i = 1$ . The package provides helpers to construct covariance matrices from  
<sup>58</sup> more interpretable quantities (e.g., per-dimension standard deviations and correlation/rotation  
<sup>59</sup> parameterizations), which is useful in scientific settings where covariances are rarely specified  
<sup>60</sup> directly.

<sup>61</sup> **Partially truncated multivariate Gaussians.** Many scientific variables are naturally bounded (e.g.,  
<sup>62</sup> ratios, angles, physical parameters constrained to an interval). MultiMin supports *partially*  
<sup>63</sup> *truncated* multivariate Gaussians by explicitly representing the truncation domain and using it  
<sup>64</sup> consistently in evaluation, sampling, and fitting. Starting from the unbounded multivariate  
<sup>65</sup> normal,

$$\mathcal{N}_k(\tilde{x}; \tilde{\mu}, \Sigma) = \frac{1}{\sqrt{(2\pi)^k \det \Sigma}} \exp \left[ -\frac{1}{2} (\tilde{x} - \tilde{\mu})^\top \Sigma^{-1} (\tilde{x} - \tilde{\mu}) \right],$$

<sup>66</sup> let  $T \subset \{1, \dots, k\}$  be the set of truncated coordinates and let  $a_i < b_i$  be the bounds for each  
<sup>67</sup>  $i \in T$ . The truncation region is

$$A_T = \{\tilde{x} \in \mathbb{R}^k : a_i \leq \tilde{x}_i \leq b_i \quad \forall i \in T\},$$

<sup>68</sup> with the remaining coordinates  $i \notin T$  unbounded. The partially truncated multivariate normal  
<sup>69</sup> is then defined as

$$\mathcal{T}\mathcal{N}_T(\tilde{x}; \tilde{\mu}, , \mathbf{a}_T, \mathbf{b}_T) = \frac{\mathcal{N}_k(\tilde{x}; \tilde{\mu}, ) \mathbf{1}_{A_T}(\tilde{x})}{Z_T(\tilde{\mu}, , \mathbf{a}_T, \mathbf{b}_T)},$$

<sup>70</sup> where  $\mathbf{1}_{A_T}$  is the indicator function of  $A_T$  and the normalization constant is

$$Z_T(\tilde{\mu}, , \mathbf{a}_T, \mathbf{b}_T) = \int_{A_T} \mathcal{N}_k(\tilde{x}; \tilde{\mu}, ) d\tilde{x} = \mathbb{P}_{\tilde{x} \sim \mathcal{N}_k(\tilde{\mu}, )} (\tilde{x} \in A_T).$$

71 MultiMin encodes bounds through a per-dimension domain argument (e.g., `domain=[[a_1,b_1],`  
 72 `..., [a_k,b_k]]`, with `None` for unbounded coordinates). A *truncated MoG* is obtained by  
 73 mixing truncated components with the same truncation region,

$$f_T(\tilde{x}) = \sum_{i=1}^M w_i \mathcal{T}\mathcal{N}_T(\tilde{x}; \tilde{\mu}_i, \mathbf{a}_T, \mathbf{b}_T).$$

74 **Maximum-likelihood fitting.** For a dataset  $\{\tilde{x}_n\}_{n=1}^N$  and a parametric MoG density  $f(\tilde{x} | \theta)$   
 75 (where  $\theta$  is a serialized list of parameters containing the components of  $\tilde{\mu}_i, \Sigma_i$ ), MultiMin  
 76 estimates parameters by classical maximum-likelihood. The likelihood and (negative) log-  
 77 likelihood objectives are

$$\mathcal{L}(\theta) = \prod_{n=1}^N f(\tilde{x}_n | \theta),$$

$$\text{NLL}(\theta) = - \sum_{n=1}^N \log f(\tilde{x}_n | \theta),$$

78 and the package minimizes NLL (or its normalized variant) with `scipy.optimize.minimize`  
 79 ([Virtanen et al., 2020](#)). We do not use an expectation–maximization (EM) clustering  
 80 formulation as a primary strategy because the typical use case here is not unsupervised  
 81 cluster discovery: instead, users select a predetermined number of components (e.g., for a  
 82 controlled parametric approximation or a physically motivated mixture), and the goal is a  
 83 direct parametric fit under constraints such as partial truncation.

## 85 Software design

86 MultiMin is organized as a compact, research-oriented library where the main public objects  
 87 are imported from the top-level namespace (`import multimin as mn`). Internally, functionality  
 88 is split into a few focused modules that mirror the conceptual workflow “define model → fit  
 89 → diagnose/visualize → export”:

- 90 **Modeling (mog):** `MixtureOfGaussians` implements parameterized multivariate mixtures,  
 91 including partially truncated domains via a per-dimension domain specification. The class  
 92 supports sampling, evaluation (PDF/log-PDF), parameter transformations convenient for  
 93 scientific reporting (e.g., standard-deviation/correlation forms and covariance constructors  
 94 in `Stats`), and serialization of a fitted model.
- 95 **Fitting (fitting):** `FitMoG` performs classical maximum-likelihood estimation by  
 96 minimizing the negative log-likelihood with `scipy.optimize.minimize`, using a flattened  
 97 parameter vector that is mapped to physical parameters (weights, means, covariance  
 98 structure) and optionally constrained through bounds. Truncated variables are handled  
 99 consistently through the same domain definition used by `MixtureOfGaussians`. For  
 100 single-valued functions, `FitFunctionMoG` fits a MoG basis to  $(X, F(X))$  pairs via  
 101 nonlinear least squares, using an analytically updated global normalization at each  
 102 objective evaluation and offering higher-level modes (e.g., adaptive routines guided by  
 103 peak finding and smoothing utilities from SciPy).
- 104 **Visualization (plotting):** `MultiPlot` provides a consistent grid layout for  
 105 scatter/histogram/PDF/contour overlays, with a single “properties” specification  
 106 controlling labels and ranges across panels. The plotting API is intentionally “thin” and  
 107 Pythonic: most methods accept standard Matplotlib-style keyword arguments (plus  
 108 structured dictionaries such as `sargs`, `hargs`, `dargs`, and `margs`) so users can tune  
 109 aesthetics without rewriting plotting logic. Optional marginal panels can be enabled  
 110 when needed.
- 111 **Utilities (util, base):** small helpers provide robust I/O (e.g., `Util.get_data` for  
 112 packaged example datasets), numerical/statistical helpers (`Stats`), and lightweight  
 113 base functionality shared by the main classes.

```

114     ▪ Optional acceleration (cmog): when the compiled library is available, MixtureOfGaussians.get_func
115         (cmog=True) generates callables that route evaluation through ctypes wrappers around
116         optimized batch evaluators, enabling fast likelihood/PDF evaluation on large grids
117         without changing the high-level user code.

118     Across the API, defaults are chosen to keep common workflows short (a few lines from
119     raw arrays to a fitted model and a figure), while exposing many optional parameters to
120     support controlled scientific use cases (fixed domains, constrained covariances, custom
121     initialization and bounds, detailed optimizer options, and fully customizable plotting). A
122     complete documentation site with narrative examples and the full API reference is provided at
123     https://multimin.readthedocs.io.

```

## 124     Code examples

```

125     Visualization as a first-class feature. Multivariate visual diagnostics are handled by MultiPlot,
126     which uses a consistent grid layout to render scatter plots, histograms, PDF heatmaps, and
127     contour overlays. This design is conceptually similar to corner-plot workflows (Foreman-Mackey, 2016),
128     but MultiMin couples the visualization directly to MoG objects and fitted
129     results, ensuring that “data vs model” comparisons reuse the same ranges, labels, and aesthetics.

130     The typical end-to-end workflow (data generation, visualization, fitting, and plotting) is a few
131     lines of code, mirroring the tutorial notebooks shipped with the project:

```

```

import numpy as np
import multimin as mn

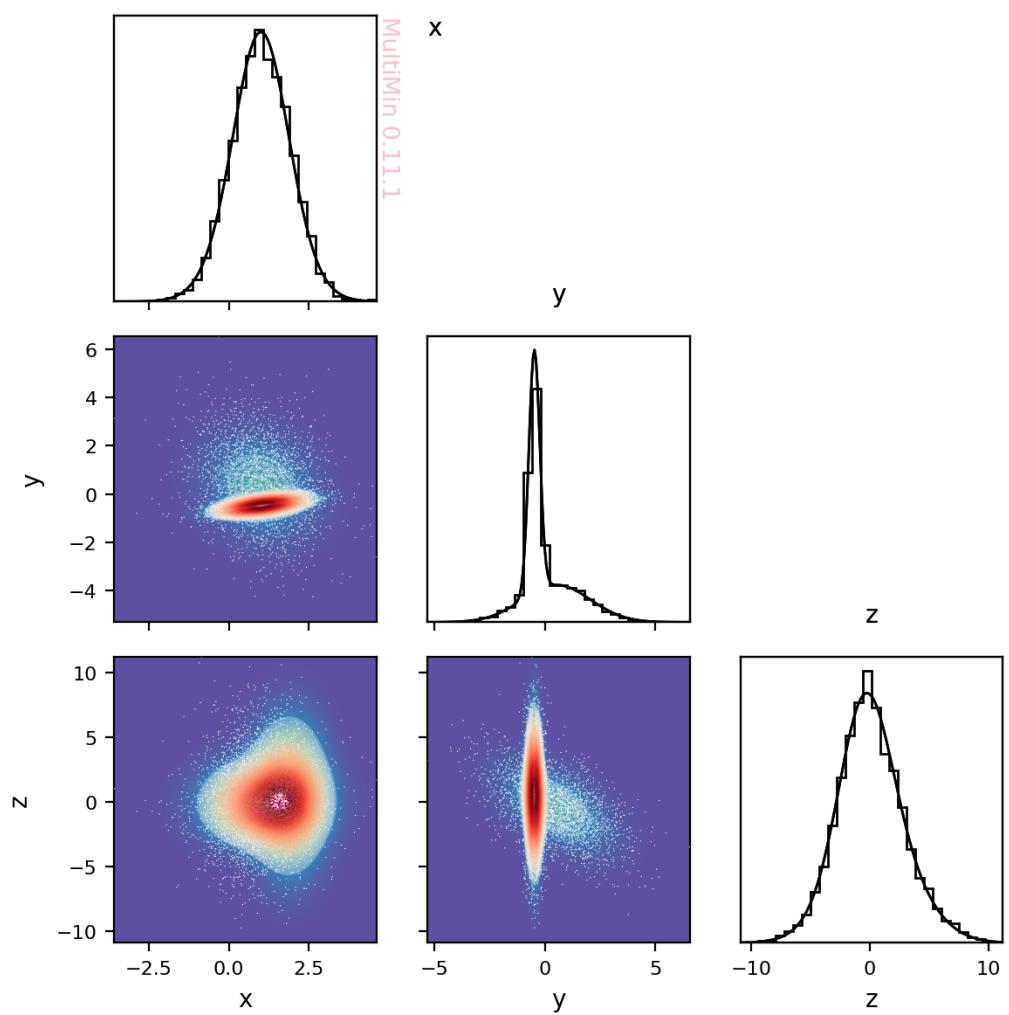
deg = np.pi / 180

# Synthetic sample from a 3D, 2-component MoG
weights = [0.5, 0.5]
mus = [[1.0, 0.5, -0.5], [1.0, -0.5, +0.5]]
sigmas = [[1, 1.2, 2.3], [0.8, 0.2, 3.3]]
angles = [[10 * deg, 30 * deg, 20 * deg], [-20 * deg, 0.0, 30 * deg]]
Sigmas = mn.Stats.calc_covariance_from_rotation(sigmas, angles)
mog_true = mn.MixtureOfGaussians(mus=mus, weights=weights, Sigmas=Sigmas)

np.random.seed(1)
data = mog_true.rvs(5000)

# Fit and plot
F = mn.FitMoG(data=data, ngauss=2)
F.fit_data()
G=F.plot_fit(
    properties=["x", "y", "z"],
    pargs=dict(cmap='Spectral_r'),
    sargs=dict(s=0.2,edgecolor='None',color='w',nbins=30),
    cargs=dict(levels=50,zorder=+300,alpha=0.3),
    figsize=3,
    marginals=True
)

```



**Figure 1:** Example of a fitted multivariate MoG visualized with `MultiPlot`.

<sup>132</sup> Fit quality diagnostics for multivariate fits are available via a Kolmogorov–Smirnov (K–S)  
<sup>133</sup> distance computed on  $S = -2 \log p(\tilde{x})$  (observed vs. synthetic samples) and an optional Q–Q  
<sup>134</sup> plot:

```

stats = F.quality_of_fit(data=data, n_sim=5000, plot_qq=True, figsize=5)
ks_dist = stats["ks_dist"]
r2_identity = stats["r2_identity"]
  
```

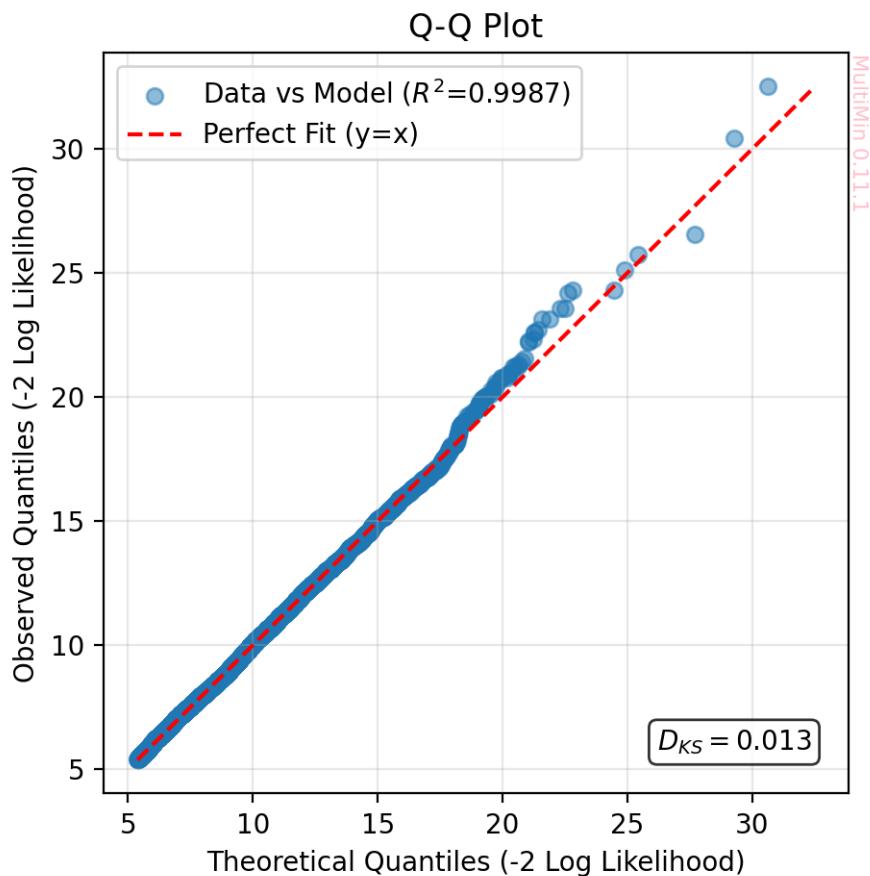


Figure 2: Q-Q plot and K-S distance for a multivariate MoG fit.

135 In addition to fitting samples, MultiMin includes an experimental workflow to fit *single-valued*  
 136 *functions* (univariate or multivariate) by matching a MoG-based model to  $(X, F(X))$  pairs.  
 137 In the univariate function-fitting setting this is formulated as a nonlinear least-squares problem  
 138 (minimizing squared residuals between the observed function values and the model prediction  
 139 on the mesh), and the package reports diagnostics such as  $R^2$  that are useful for model  
 140 selection and automated fitting loops.

#### 141 Univariate spectral-line fitting

142 A practical example of the function-fitting workflow is the decomposition of a complex univariate  
 143 spectral line profile into a small number of Gaussian components. The spectrum used in this  
 144 example is shipped with MultiMin (complex-line.txt) and is based on the multi-component  
 145 synthetic spectra used in the context of GaussPy+ (Riener et al., 2019). It can be loaded via  
 146 `mn.Util.get_data` and `numpy.loadtxt` and then fitted as:

```
import numpy as np
import multimin as mn

path = mn.Util.get_data("complex-line.txt")
chan, spectrum = np.loadtxt(path, unpack=True, comments="#")

Ff = mn.FitFunctionMoG(data=(chan, spectrum), ngauss=3)
Ff.fit_data(mode="adaptive", advance=10)
fig = Ff.plot_fit(dargs=dict())
```

```
stats_f = Ff.quality_of_fit()
```

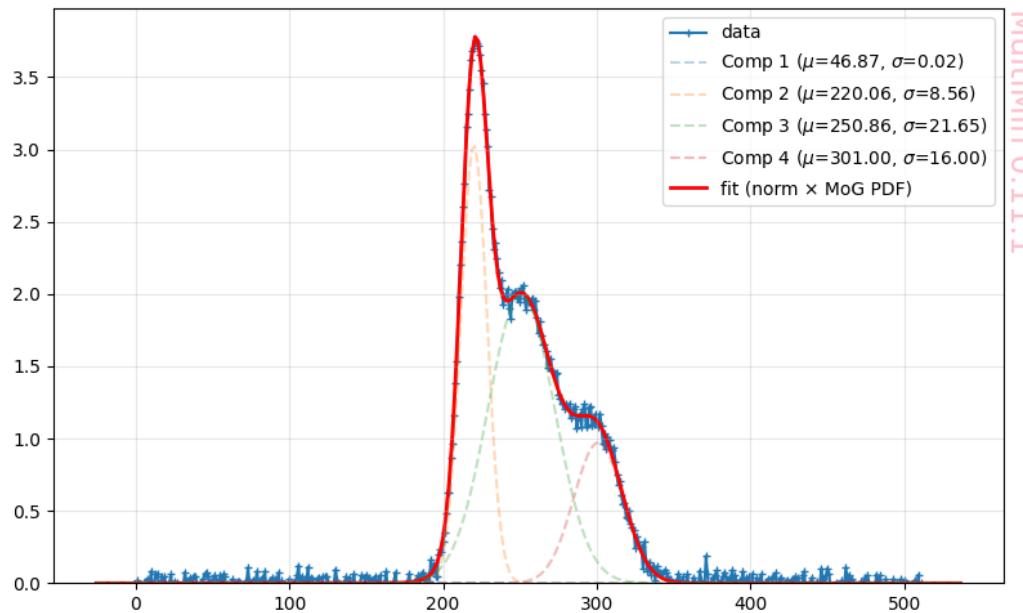


Figure 3: Example of a complex spectral line and a MoG-based fit using FitFunctionMoG.

### 147 Semi-analytical descriptions and accelerated evaluators

148 Beyond fitting and plotting, MultiMin can export fitted distributions as semi-analytical artifacts  
149 suitable for papers and high-throughput evaluation. In particular, fitted MoGs can be rendered  
150 as LaTeX (explicit parameters and matrices) and as self-contained Python callables. For  
151 example, the LaTeX export can be generated as follows:

```
latex_str, _ = F.mog.get_function(type="latex", print_code=False, decimals=4)
print(latex_str)
```

152 The latex\_str output is formatted as follows:

$$f(\mathbf{x}) = w_1 \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_1, \Sigma_1) + w_2 \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_2, \Sigma_2)$$

153 where

$$w_1 = 0.4909$$

$$\boldsymbol{\mu}_1 = \begin{pmatrix} 0.9577 \\ 0.5176 \\ -0.4634 \end{pmatrix}$$

$$\Sigma_1 = \begin{pmatrix} 1.0805 & -0.3353 & 0.2666 \\ -0.3353 & 2.3655 & -1.716 \\ 0.2666 & -1.716 & 4.4798 \end{pmatrix}$$

$$w_2 = 0.5091$$

$$\boldsymbol{\mu}_2 = \begin{pmatrix} 1.0192 \\ -0.481 \\ 0.6188 \end{pmatrix}$$

157

$$_2 = \begin{pmatrix} 0.6319 & 0.1054 & -0.0236 \\ 0.1054 & 0.0604 & -0.0145 \\ -0.0236 & -0.0145 & 11.0725 \end{pmatrix}$$

158 Here the normal distribution is defined as:

$$\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \Sigma) = \frac{1}{\sqrt{(2\pi)^k \det \Sigma}} \exp \left[ -\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu})^\top \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu}) \right]$$

159 These expressions can be incorporated into scientific publications to provide a semi-analytical  
160 representation of the fitted distribution.

### 161 Accelerated evaluators

162 Optionally, the generated Python code can target C-optimized batch evaluators exposed  
163 through ctypes for fast evaluation on large grids:

```
164 code_str_c, f_fast_c = F.mog.get_function(type="python", cmog=True, print_code=False)
```

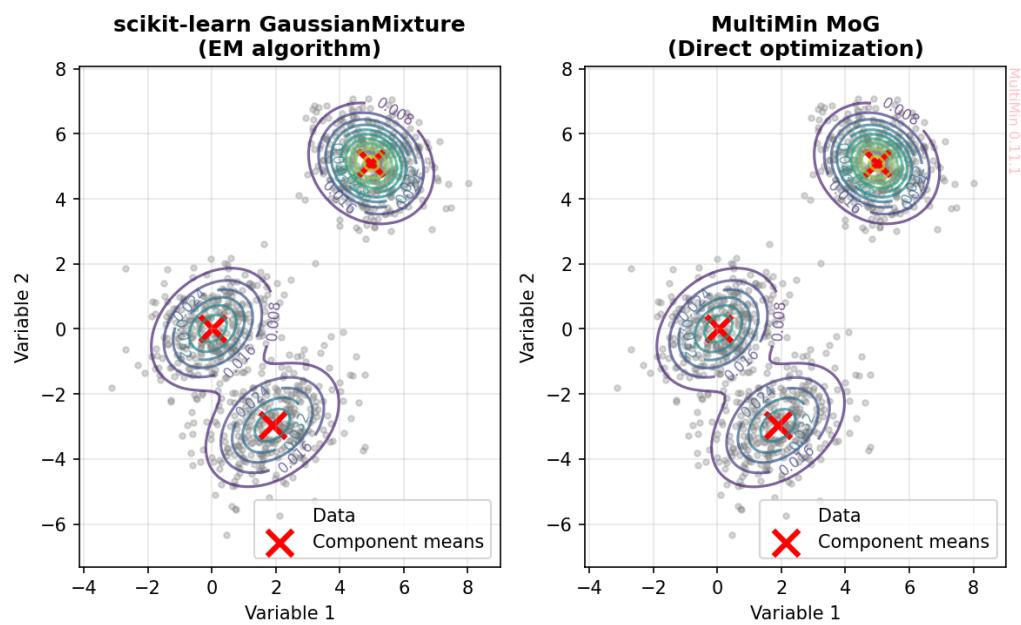
165 When the optional C backend is available, cmog=True routes evaluation through ctypes  
166 wrappers around compiled batch evaluators. In examples/multimin\_cmog.ipynb, a  
167 representative micro-benchmark shows a speedup for point-wise evaluation (same fitted MoG,  
same machine/kernel):

- 168   ■ Single point: Python callable  $\sim 60 \mu\text{s}$  per loop vs cmog=True  $12 \mu\text{s}$  per loop. Speed-up  
169    $\sim \times 5$ .
- 170   ■ Batch (10,000 points): Python callable  $445 \mu\text{s}$  per loop vs cmog=True  $280 \mu\text{s}$  per loop.  
171   Speed-up  $\sim \times 2$ .

## 172 Comparison with other similar tools

173 MultiMin overlaps with existing Gaussian-mixture tooling, but targets a different point in  
174 the design space: it prioritizes scientific interpretability, bounded (truncated) domains, and  
175 reproducible diagnostics/figures as first-class outputs.

176 **scikit-learn.** scikit-learn provides a widely used GaussianMixture implementation oriented  
177 toward machine-learning workflows (e.g., clustering and density estimation pipelines)  
178 (Pedregosa et al., 2011). MultiMin can be used for similar density-estimation tasks,  
179 but emphasizes an explicit, report-friendly parameterization (weights, means, standard  
180 deviations/correlations/covariances), publication-ready multivariate visualization via MultiPlot,  
181 and domain-aware (partially truncated) likelihoods. The repository includes a notebook-level  
182 comparison between MultiMin and scikit-learn GMM (see Figure 4).



**Figure 4:** Comparison figure produced by the MultiMin notebooks, contrasting mixture-model workflows.

183 **GaussPy+.** GaussPy+ is a specialized automated Gaussian decomposition package for emission-  
 184 line spectra, with substantial algorithmic machinery for noise estimation, quality control, and  
 185 (optionally) spatially coherent refitting in large surveys (Riener et al., 2019). MultiMin is  
 186 not a drop-in replacement for that end-to-end pipeline; however, for the core task of fitting  
 187 a spectrum with multiple Gaussian components, the FitFunctionMoG workflow produces  
 188 comparable decompositions in representative cases (e.g., the complex-line example included  
 189 with this package and shown in Figure 3), while providing a unified API that also extends to  
 190 multivariate MoG fitting with shared diagnostics (e.g., K-S / Q-Q tests).

## 191 Research impact statement

192 MultiMin was developed to support research workflows where multivariate distributions must  
 193 be fitted, interpreted, and communicated with consistent diagnostics. The repository includes  
 194 reproducible notebooks demonstrating applications such as Near-Earth asteroid datasets and  
 195 controlled comparisons against scikit-learn's GMM implementation, along with gallery figures  
 196 that can be directly reused in publications and reports. By integrating fitting, diagnostics, and  
 197 visualization in a single package, MultiMin reduces the glue code typically required to turn a  
 198 fitted mixture model into reproducible, publication-quality results.

## 199 AI usage disclosure

200 Generative AI tools were used during the development and documentation of MultiMin,  
 201 including the preparation of this manuscript. Specifically, AI assistance was used for drafting  
 202 text, refactoring portions of code, and scaffolding tests and examples. All AI-assisted outputs  
 203 were reviewed, edited, and validated by the human author(s), who made the core technical  
 204 and design decisions and verified correctness via automated tests and executable notebooks.

## 205 Acknowledgements

206 The author thanks the SEAP/FACom community at Universidad de Antioquia for feedback  
207 during development, and Juanita Agudelo-Quiceno (Universidad de Antioquia) for beta-testing  
208 and for using the library in the context of Near-Earth Asteroid (NEA) research. The author  
209 also acknowledges the broader scientific Python ecosystem for foundational libraries used by  
210 this project.

## 211 References

- 212 Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer.
- 213 Foreman-Mackey, D. (2016). Corner.py: Scatterplot matrices in python. *The Journal of Open*  
214 *Source Software*, 1(2), 24. <https://doi.org/10.21105/joss.00024>
- 215 Harris, C. R., Millman, K. J., Walt, S. J. van der, Gommers, R., Virtanen, P., Cournapeau, D.,  
216 Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., Kerkwijk,  
217 M. H. van, Brett, M., Haldane, A., Río, J. F. del, Wiebe, M., Peterson, P., ... Oliphant,  
218 T. E. (2020). Array programming with NumPy. *Nature*, 585(7825), 357–362. <https://doi.org/10.1038/s41586-020-2649-2>
- 219 Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science &*  
220 *Engineering*, 9(3), 90–95. <https://doi.org/10.1109/MCSE.2007.55>
- 221 McLachlan, G., & Peel, D. (2000). *Finite mixture models*. Wiley.
- 222 Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel,  
223 M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau,  
224 D., Brucher, M., Perrot, M., & Duchesnay, É. (2011). Scikit-learn: Machine learning in  
225 python. *Journal of Machine Learning Research*, 12, 2825–2830. <http://jmlr.org/papers/v12/pedregosa11a.html>
- 226 Riener, M., Kainulainen, J., Henshaw, J. D., Orkisz, J. H., Murray, C. E., & Beuther, H. (2019).  
227 GaußPy+: A fully automated Gaussian decomposition package for emission line spectra.  
228 *Astronomy & Astrophysics*, 628, A78. <https://doi.org/10.1051/0004-6361/201935519>
- 229 Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D.,  
230 Burovski, E., Peterson, P., Weckesser, W., Bright, J., Walt, S. J. van der, Brett, M.,  
231 Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E.,  
232 ... Mulbregt, P. van. (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing  
233 in Python. *Nature Methods*, 17(3), 261–272. <https://doi.org/10.1038/s41592-019-0686-2>
- 234
- 235