Package: opGMMassessment (via r-universe)

May 17, 2026

Type Package

Title Optimized Automated Gaussian Mixture Assessment

Version 0.4.5

Maintainer Jorn Lotsch <j.lotsch@em.uni-frankfurt.de>

Description Necessary functions for optimized automated evaluation of the number and parameters of Gaussian mixtures in one-dimensional data. Various methods are available for parameter estimation and for determining the number of modes in the mixture. A detailed description of the methods ca ben found in Lotsch, J., Malkusch, S. and A. Ultsch. (2022) <doi:10.1016/j.imu.2022.101113>.

URL https://github.com/JornLotsch/opGMMassessment

Depends R (>= 3.5.0)

License GPL-3

Encoding UTF-8

LazyData true

LinkingTo Rcpp

Imports AdaptGauss, DataVisualizations, DistributionOptimization, cluster, mixtools, grDevices, methods, foreach, stats, utils, rlang, ggplot2, parallel, caTools, dplyr, mclust, mixAK, multimode, NbClust, ClusterR, doParallel, Rcpp (>= 1.0.0)

Config/pak/sysreqs cmake libgmp3-dev make libicu-dev libuv1-dev libssl-dev zlib1g-dev

Repository https://jornlotsch.r-universe.dev

Date/Publication 2026-04-17 10:31:45 UTC

RemoteUrl https://github.com/jornlotsch/opgmmassessment

RemoteRef HEAD

RemoteSha dcf9080b136a7ab55a3e95c1789533305f598efd

Contents

`Chromatogram` *Example data of lysophosphatidic acids, LPA.*

Description

Data set containing times of detector hits after chromatographic separation of five different lysophosphatidic acids (Classes CLs = LPA 16:0, 18:0, 18:3, 20:0, and 20:4).

Usage

```
data("Chromatogram")
```

Details

Size 1166 x 3 , stored in `Chromatogram$[Cls, Time, Lipids]`

Examples

```
data(Chromatogram)
str(Chromatogram)
```

`get_seed` *Random Seed Recovery from RNG State*

Description

Functions for recovering the original seed value that produced the current random number generator state. Provides both R and C++ implementations with the C++ version offering significantly improved performance for large search spaces.

Usage

```
get_seed(range = NULL, fallback_seed = 42, max_search = 2147483647,
  step_size = 50000, use_cpp = TRUE, ...)

get_seed_cpp(range = NULL, fallback_seed = 42, max_search = 2147483647,
  step_size = 50000, batch_size = 10000, verbose = TRUE)
```

Arguments

range	Optional integer vector of specific seed values to search. If provided, only these seeds will be tested instead of systematic range searching.
fallback_seed	Integer seed value to return if no matching seed is found during the search process (default: 42).
max_search	Maximum seed value to search up to when performing systematic range searching. Must be a positive integer within the valid range for R's random number generator (default: 2147483647).
step_size	Step size for systematic range searching when no specific range is provided. Larger values speed up search but may miss the target seed if it falls between steps (default: 50000).
use_cpp	Logical; if TRUE, uses the fast C++ implementation via get_seed_cpp(). If FALSE, falls back to the slower R implementation with a warning (default: TRUE).
batch_size	Integer specifying the number of seeds to process in each C++ batch operation. Larger batches are more memory efficient but require more RAM. Only used in get_seed_cpp() (default: 10000).
verbose	Logical; if TRUE, prints progress information during the search process including batch progress and timing information. Only used in get_seed_cpp() (default: TRUE).
...	Additional arguments passed to get_seed_cpp() when use_cpp = TRUE.

Details

The functions work by systematically testing seed values to find one that reproduces the current RNG state stored in .Random.seed. The search process:

- Tests each candidate seed by setting it and comparing the resulting RNG state
- Uses efficient C++ implementation for faster processing of large search spaces
- Supports both targeted searching (via range parameter) and systematic range searching
- Employs batched processing to optimize memory usage and performance

Performance Considerations:

The C++ implementation (get_seed_cpp()) provides significant performance improvements:

- Batch processing reduces overhead for large search spaces
- Optimized memory management prevents excessive RAM usage
- Native C++ random number generation matching R's implementation
- Progress reporting for long-running searches

Search Strategy:

- If range is provided: Tests only the specified seed values
- If range is NULL: Performs systematic search from 1 to max_search in steps of step_size
- Search terminates immediately when a matching seed is found

- Returns `fallback_seed` if no match is found within the search parameters

Memory Management:

The C++ implementation uses batched processing controlled by `batch_size` to:

- Process large search ranges without excessive memory allocation
- Provide regular progress updates during long searches
- Allow interruption of long-running operations

Value

Returns an integer representing the seed value that reproduces the current random number generator state. If no matching seed is found within the search parameters, returns the `fallback_seed` value.

Note

- Requires an active RNG state (i.e., `.Random.seed` must exist)
- Large search ranges may take considerable time even with C++ optimization
- The search is deterministic but computationally intensive
- Consider using smaller `step_size` values if the initial search fails

See Also

`set.seed`, `.Random.seed`

Examples

```
## Basic seed recovery after generating random numbers
set.seed(123)
recovered_seed <- get_seed()
print(recovered_seed)
```

GMMplotGG *Plot of Gaussian mixtures*

Description

The function plots the components of a Gaussian mixture and superimposes them on a histogram of the data.

Usage

```
GMMplotGG(Data, Means, SDs, Weights, BayesBoundaries,
 SingleGausses = TRUE, Hist = FALSE, Bounds = TRUE, SumModes = TRUE, PDE = TRUE)
```

Arguments

Data	the data as a vector.
Means	a list of mean values for a Gaussian mixture.
SDs	a list of standard deviations for a Gaussian mixture.
Weights	a list of weights for a Gaussian mixture.
BayesBoundaries	a list of Bayesian boundaries for a Gaussian mixture.
SingleGausses	whether to plot the single Gaussian components as separate lines.
Hist	whether to plot a histgram of the original data.
Bounds	whether to plot the Bayesian boundaries for a Gaussian mixture as vertical lines.
SumModes	whether to plot the summed-up mixes.
PDE	whether to use the Pareto density estimation instead of the standard R density function.

Value

Returns a ggplot2 object.

p1	the plot of Gaussian mixtures.

Author(s)

Jorn Lotsch and Sebastian Malkusch

References

Lotsch, J., Malkusch S. (2021): opGMMassessment – an R Package for automated Guassian mixture modeling.

Examples

```
## example 1
data(iris)
Means0 <- tapply(X = as.vector(iris[,3]), INDEX =  as.integer(iris$Species), FUN = mean)
SDs0 <- tapply(X = as.vector(iris[,3]), INDEX =  as.integer(iris$Species), FUN = sd)
Weights0 <- c(1/3, 1/3, 1/3)
GMM.Sepal.Length <- GMMplotGG(Data = as.vector(iris[3]),
 Means = Means0,
 SDs = SDs0,
 Weights = Weights0,
 Hist = TRUE)
```

Mixture3 *Example Gaussian mixture data.*

Description

Data set containing 1000 instances distributed according to a Gaussian mixture with m = [-10, 0, 10], s = [1, 2, 3], w = [0.07, 0.05, 0.88].

Usage

```
data("Mixture3")
```

Details

Size 1000 x 1

Examples

```
data(Mixture3)
str(Mixture3)
```

opGMMassessment *Gaussian mixture assessment*

Description

The package provides the necessary functions for optimized automated evaluation of the number and parameters of Gaussian mixtures in one-dimensional data. It provides various methods for parameter estimation and for determining the number of modes in the mixture.

Usage

```
opGMMassessment(Data, FitAlg = "MCMC", Criterion = "LR",
   MaxModes = 8, MaxCores = getOption("mc.cores", 2L),
   PlotIt = FALSE, KS = TRUE, Seed = "simple")
```

Arguments

Data	the data as a numerical vector (1D).
FitAlg	which fit algorithm to use: "ClusterRGMM" = GMM from ClusterR, "densityMclust" from mclust, "DO" from DistributionOptimization (slow), "MCMC" = NMixMCMC from mixAK, or "normalmixEM" from mixtools.
Criterion	which criterion should be used to establish the number of modes from the best GMM fit: "AIC", "BIC", "FM", "GAP", "LR" (likelihood ratio test), "NbClust" (from NbClust), "SI" (Silverman).

MaxModes	the maximum number of modes to be tried.
MaxCores	the maximum number of processor cores used under Unix.
PlotIt	whether to plot the fit directly (plot will be stored nevertheless).
KS	perform a Kolmogorow-Smirnow test of the fit versus original distribution.
Seed	Seed value for reproducibility. Options: "auto" (complex seed recovery, slow), "simple" (fast reproducible seed, default), or integer (exact control, recommended). Use integers for systematic testing with different seeds, "simple" for general use, or "auto" when you need to maintain exact RNG state continuity.

Value

Returns a list of Gaussian modes.

Cls	the classes to which the cases are assigned according to the Gaussian mode membership.
Means	means of the Gaussian modes.
SDs	standard deviations of the Gaussian modes.
Weights	weights of the Gaussian modes.
Boundaries	Bayesian boundaries between the Gaussian modes.
Plot	Plot of the obtained mixture.
KS	Results of the Kolmogorov-Smirnov test.

Author(s)

Jorn Lotsch and Sebastian Malkusch

References

Lotsch J, Malkusch S, Ultsch A. Comparative assessment of automated algorithms for the separation of one-dimensional Gaussian mixtures. Informatics in Medicine Unlocked, Volume 34, 2022, https://doi.org/10.1016/j.imu.2022.101113.

Examples

```
## example 1
data(iris)
opGMMassessment(Data = iris$Petal.Length,
  FitAlg = "normalmixEM",
  Criterion = "BIC",
  PlotIt = TRUE,
  MaxModes = 5,
  MaxCores = 1,
  Seed = 42)
```

Index