Package: evmissing (via r-universe)

May 23, 2026

Type Package

Title Extreme Value Analyses with Missing Data

Version 1.0.2

Date 2026-03-05

Maintainer Paul J. Northrop <p.northrop@ucl.ac.uk>

Description Performs likelihood-based extreme value inferences with adjustment for the presence of missing values based on Simpson and Northrop (2026) <doi:10.1002/env.70075>. A Generalised Extreme Value distribution is fitted to block maxima using maximum likelihood estimation, with the location and scale parameters reflecting the numbers of non-missing raw values in each block. A Bayesian version is also provided. For the purposes of comparison, there are options to make no adjustment for missing values or to discard any block maximum for which greater than a percentage of the underlying raw values are missing. Example datasets containing missing values are provided.

Imports gamlssx, graphics, itp, nieve, revdbayes, rust, stats

License GPL (>= 3)

LazyData TRUE

Encoding UTF-8

Depends R (>= 4.1.0)

Suggests testthat (>= 3.0.0)

URL https://paulnorthrop.github.io/evmissing/, https://github.com/paulnorthrop/evmissing

BugReports https://github.com/paulnorthrop/evmissing/issues

Config/testthat/edition 3

RoxygenNote 7.3.3

Roxygen list(markdown = TRUE)

Config/pak/sysreqs libicu-dev

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

Date/Publication 2026-04-23 17:57:10 UTC

RemoteUrl https://github.com/paulnorthrop/evmissing

RemoteRef HEAD

RemoteSha 9ac7ee6ac388485a9f702f832e31e8276446c0e8

Contents

`evmissing-package` *evmissing: Extreme Value Analyses with Missing Data*

Description

Performs likelihood-based extreme value inferences with adjustment for the presence of missing values. A Generalised Extreme Value (GEV) distribution is fitted to block maxima using maximum likelihood estimation, with the GEV location and scale parameters reflecting the numbers of non-missing raw values in each block. A Bayesian version is also provided. For the purposes of comparison, there are options to make no adjustment for missing values or to discard any block maximum for which greater than a percentage of the underlying raw values are missing.

The `evmissing` package was created to accompany the research paper Simpson, E. S. and Northrop, P. J. (2026) Accounting for missing data when modelling block maxima, *Environmetrics*, **37**(2): e70075 doi:10.1002/env.70075.

Details

The main functions are

- `gev_mle`: maximum likelihood inference for block maxima based on a GEV distribution, with `S3 methods` including `confint`.
- `gev_bayes`: Bayesian inference for block maxima based on a GEV distribution.

For objects returned by `gev_mle`, inferences about return levels are performed by `gev_return`, with with `S3 methods` including `confint`.

The function `gev_influence` quantifies the influence that individual extreme (small or large) block maxima have on the maximum likelihood estimators of GEV parameters.

The following example datasets are provided.

- `BloomsburyOzoneMaxima`: Annual maxima ozone levels at Bloomsbury, London, UK, 1992-2024.
- `PlymouthOzoneMaxima`: Annual maxima ozone levels at Plymouth, Devon, UK, 1998-2024.
- `BrestSurgeMaxima`: Annual maxima surge heights at Brest, France, 1846-2007.

Author(s)

Maintainer: Paul J. Northrop <`p.northrop@ucl.ac.uk`> [copyright holder]

Authors:

- Emma S. Simpson [copyright holder]

References

Simpson, E. S. and Northrop, P. J. (2026) Accounting for Missing Data When Modelling Block Maxima, *Environmetrics* **37**(2): e70075. doi:10.1002/env.70075.

See Also

Useful links:

- `https://paulnorthrop.github.io/evmissing/`
- `https://github.com/paulnorthrop/evmissing`
- Report bugs at `https://github.com/paulnorthrop/evmissing/issues`

`block_maxima` *Block Maxima and Missing Information*

Description

Extracts disjoint block maxima from a time series of raw data. Can also provide information about the effect of missing value patterns on block maxima via pseudo-maxima created by applying block-wise missing value patterns in partially-observed (partial) blocks to fully-observed (full) blocks.

Usage

```
block_maxima(
  data,
  block_length,
  block,
  pseudo = FALSE,
  sliding = FALSE,
  season = FALSE
)
```

Arguments

`data`	A numeric vector containing a time series of raw data. If `sliding = FALSE` then `data` must contain at least one full disjoint block. If `sliding = TRUE` then `data` must contain at least one full sliding block.
`block_length`	A numeric scalar. Used calculate the maxima of disjoint blocks of `block_length` contiguous values in the vector `data`. If `length(data)` is not an integer multiple of `block_length` then the values at the end of `data` that do not constitute a complete block of length `block_length` are discarded, without warning.
`block`	A numeric vector with the same length as `data`. The value of `block[i]` indicates the block into which `data[i]` falls. For example, `block` could provide the year in which observation `i` was observed. The block lengths implied by `block` should have similar values, for example, 366 for leap years and 365 for other years.
`pseudo`	A logical scalar. If `pseudo = TRUE` then pseudo-maxima are calculated as the block maxima obtained by applying the missing value patterns from partial blocks to all full blocks.
`sliding`	A logical scalar. Only relevant if `pseudo = TRUE`. If `sliding = TRUE` then pseudo-maxima are calculated for **all** full blocks of the relevant length, rather than only a set of disjoint blocks. See **Details**.
`season`	A logical scalar. Only relevant if `pseudo = TRUE` and `sliding = TRUE`. If `season = TRUE` then the way in which the pseudo-maxima are calculated respects the seasonality that may be exhibited over the duration of a block. If, for example, a block covers a single year, then the missing values applied to a full block occur at the same time of year as in the originating partial block. If `season = FALSE` then the missing values applied have the same positions in the partial and full blocks with respect to the start of each block.

Details

Exactly one of the arguments block_length or block must be supplied.

If block_length is supplied and sliding = TRUE then the maxima are calculated for **all** blocks of length block_length present in data, starting with the first block data[1:block_length] and sliding the block repeatedly by one observation until reaching the final block data[(length(data) - block_length + 1):length(data)].

Also explain for block

Value

A list, with class c("list", "block_maxima", "disjoint", "evmissing"), containing the following numeric vectors:

- maxima: the block maxima.
- notNA: the numbers of non-missing observations in each block.
- n: the maximal block length, that is, the largest number of values that could have been observed in each block.

If pseudo = TRUE then the returned list also contains the following:

- whereNA: a named list containing, for each block, the positions of any missing values in the block. For example, if (only) the first and fifth observations in block 3 are missing then the third component (named block3) of whereNA is c(1, 5). If a block has no missing values then its component in whereNA is integer(0).
- pseudo_maxima: a numeric matrix containing block maxima created by applying the missing value patterns from partial blocks to all full blocks. Each column contains the pseudo-maxima resulting from a particular partial block. The columns are labelled by the number of the partial block and the columns by the number of the full block. If a partial block contains all missing values then its entry in pseudo_maxima is NA. If there are no full blocks or no partial blocks then pseudo_maxima is NA.
- full_maxima: a numeric vector of maxima from full blocks.
- partial_maxima: a numeric vector of maxima from partial blocks.

The input arguments pseudo and sliding are also included.

If a block contains only missing values then its value of maxima is NA and its value of notNA is 0.

If block is supplied then these vectors are named using the values in block. Otherwise, these vectors do not have names.

See Also

Plot method plot.block_maxima.

Examples

```
## Simulate example data
set.seed(7032025)
data <- rexp(15)

# Create some missing values
data[c(5, 7:8)] <- NA
# 5 blocks (columns), each with 3 observations
matrix(data, ncol = 5)

# Supplying block_length, disjoint maxima
block_length <- 3
block_maxima(data, block_length = block_length)
# Supplying block_length, sliding maxima
block_maxima(data, block_length = block_length, sliding = TRUE)

# Supplying block
block <- rep(1:5, each = 3)
block_maxima(data, block = block)

## Data with a partially-observed block
data <- c(data, 1:2)

# Supplying block_length (the extra 2 observations are ignored)
block_length <- 3
block_maxima(data, block_length = block_length)
# Supplying block (with an extra group indicator)
block <- c(block, 7, 7)
block_maxima(data, block = block)
```

---

block_maxima_ts *Block Maxima for a Time Series*

Description

Extracts block maxima and missing value information for each block. Works like `block_maxima` but returns extra components, including: `whereNA`, the positions of the missing values within each block, and `pseudo_maxima`, the maxima created by applying blockwise missing value patterns in incomplete blocks to full blocks, that is, blocks without any missing values. To be useful, the input data, `data`, must contain at least one full block.

Usage

```
block_maxima_ts(data, block_length, block)
```

Arguments

data	A numeric vector containing a time series of raw data.
block_length	A numeric scalar. Used calculate the maxima of disjoint blocks of block_length contiguous values in the vector data. If length(data) is not an integer multiple of block_length then the values at the end of data that do not constitute a complete block of length block_length are discarded, without warning.
block	A numeric vector with the same length as data. The value of block[i] indicates the block into which data[i] falls. For example, block could provide the year in which observation i was observed.

Details

Exactly one of the arguments block_length or block must be supplied. If the block sizes implied by block are unequal then an incomplete block and a full block may have different lengths. If this occurs when pseudo_maxima are calculated, then the longer block is trimmed, by discarding trailing values, so that the lengths match.

Value

A list, with class c("list", "block_maxima", "evmissing"), containing the following components:

- maxima: the block maxima.
- notNA: the numbers of non-missing observations in each block.
- n: the maximal block length, that is, the largest number of values that could have been observed in each block.
- whereNA: a named list containing, for each block, the positions of any missing values in the block. For example, if (only) the first and fifth observations in block 3 are missing then the third component (named block3) of whereNA is c(1, 5). If a block has no missing values then its component in whereNA is integer(0).
- pseudo_maxima: a numeric matrix containing (pseudo) block maxima created by applying the missing value patterns from incomplete blocks to all full blocks, that is, blocks without any missing values. Each column contains the pseudo-maxima resulting from a particular incomplete block. The columns are labelled by the number of the incomplete block and the columns by the number of the full block. If an incomplete block contains all missing values then its entry in pseudo_maxima is NA. If there are no full blocks or no incomplete blocks then pseudo_maxima is NA.
- full_maxima: a numeric vector of maxima from full blocks.
- partial_maxima: a numeric vector of maxima from partial blocks.

If a block contains only missing values then its value of maxima is NA, its value of notNA is 0 and whereNA contains the positions of all the observations in the block.

If block is supplied then these vectors are named using the values in block. Otherwise, these vectors do not have names.

Examples

```
## Simulate example data
set.seed(7032025)
data <- rexp(15)

# Create some missing values
data[c(5, 7:8)] <- NA
# 5 blocks (columns), each with 3 observations
matrix(data, ncol = 5)
# Supplying block_length
block_length <- 3
block_maxima_ts(data, block_length = block_length)
# Supplying block
block <- rep(1:5, each = 3)
block_maxima_ts(data, block = block)

## Data with an incomplete block
data <- c(data, 1:2)

# Supplying block_length (the extra 2 observations are ignored)
block_length <- 3
block_maxima_ts(data, block_length = block_length)
# Supplying block (with an extra group indicator)
block <- c(block, 7, 7)
block_maxima_ts(data, block = block)
```

BloomsburyOzone *Ozone Levels at Bloomsbury, UK*

Description

Daily maximum ozone levels at Bloomsbury in London (UK) for the years 1992-2024 inclusive.

Usage

```
BloomsburyOzone
```

Format

`BloomsburyOzone` is a data frame with 12054 rows and the 3 variables:

- `Date`: with class `"Date"` in the format `YYYY-MM-DD`.
- `Year`: Values in 1992-2024.
- `Ozone`: daily maximum ozone level in μg/m^3.

Source

The Department for Environment Food and Rural Affair (DEFRA). The London Bloomsbury monitoring site at the UK-AIR database Data Selector.

See Also

`BloomsburyOzoneMaxima` for the annual maxima and numbers of missing values per year.

Examples

```
head(BloomsburyOzone)

# Time series plot of annual maxima ozone levels
plot(BloomsburyOzone$Date, BloomsburyOzone$Ozone, xlab = "year",
     ylab = "ozone (micrograms / metre cubed)", pch = 16)
```

`BloomsburyOzoneMaxima` *Annual Maxima Ozone Levels at Bloomsbury, UK*

Description

Annual maxima of daily maximum ozone levels at Bloomsbury in London (UK) for the years 1992-2024 inclusive.

Usage

```
BloomsburyOzoneMaxima
```

Format

`BloomsburyOzoneMaxima` is a data frame with 33 rows (years 1992 to 2024) and the 4 variables:

- `maxima`: annual maximum ozone level in μg/m^3.
- `notNA` : the number of days of the year for which raw data were available.
- `n` : the number of days in the year (365 or 366).
- `block` : a block number of 1 for year 1992 through to 33 for year 2024.

The row names of `BloomsburyOzoneMaxima` are the years `1992:2024`. The raw data are missing for approximately 5% of the days.

Source

The Department for Environment Food and Rural Affair (DEFRA). The London Bloomsbury monitoring site at the UK-AIR database Data Selector.

See Also

`BloomsburyOzone` for the raw time series.

Examples

```
head(BloomsburyOzoneMaxima)

# Time series plot of annual maxima ozone levels
plot(rownames(BloomsburyOzoneMaxima), BloomsburyOzoneMaxima$maxima,
     ylab = "ozone (micrograms / metre cubed)", xlab = "year", pch = 16)

# Time series plot of proportion of non-missing days
plot(rownames(BloomsburyOzoneMaxima),
     BloomsburyOzoneMaxima$notNA / BloomsburyOzoneMaxima$n,
     ylab = "proportion of non-missing days", xlab = "year", pch = 16)

# Plot ozone levels against the proportion of non-missing days
plot(BloomsburyOzoneMaxima$notNA / BloomsburyOzoneMaxima$n,
     BloomsburyOzoneMaxima$maxima,
     ylab = "ozone (micrograms / metre cubed)",
     xlab = "proportion of non-missing days", pch = 16)
```

BrestSurgeDays *Number of Days per Month in 1846-2007*

Description

Number of days in each month relevant to the Brest sea surge heights data BrestSurgeMaxima.

Usage

```
BrestSurgeDays
```

Format

BrestSurgeDays is a data frame with 162 rows (years 1846 to 2007) and the 12 variables (one for each month of the year). Each value in the data frame gives the number of days in the month in question.

The row names of BrestSurgeMaxima are the years 1946:2007 and the column names are the abbreviated names of the months.

See Also

- BrestSurgeMaxima: Annual maxima surge heights at Brest, France.
- BrestSurgeMissing: numbers of missing values in each month.

Examples

```
head(BrestSurgeDays)
```

`BrestSurgeMaxima` *Annual Maxima Sea Surge Heights at Brest, France*

Description

Annual maxima of sea surge heights near high tide at Brest tide gauge station (France) for the years 1846-2007 inclusive.

Usage

```
BrestSurgeMaxima
```

Format

`BrestSurgeMaxima` is a data frame with 162 rows (years 1846 to 2007) and the 4 variables:

- `maxima`: annual maximum surge height at high tide in cm.
- `notNA` : the number of days of the year for which raw data were available.
- `n` : the number of days in the year (365 or 366).
- `block` : a block number of 1 for year 1846 through to 162 for year 2007.

The row names of `BrestSurgeMaxima` are the years `1946:2007`.

Note

The raw data are missing for approximately 9% of the days. The data were declustered by the original providers in order to provide a series of independent surge heights at high tide. Specifically, these surge heights are separated by at least two days. A correction was applied to account for trend in the sea-level over the observation period. Although the declustering of the data means that the effective block size is smaller than `n`, it may be reasonable to suppose that the proportion `notNA/n` of non-missing values provides a useful measure of the extent to which the size of an annual maximum is likely to be affected by missingness.

Source

The dataset `Brest` in the `Renext` R package, specifically `Brest$OTdata` and `Brest$OTmissing`. Originally, the source was `https://data.shom.fr/`.

References

Deville Y. and Bardet L. (2023). Renext: Renewal Method for Extreme Values Extrapolation. R package version 3.1-4. doi:10.32614/CRAN.package.Renext

See Also

- `BrestSurgeMissing`: numbers of missing values in each month.
- `BrestSurgeDays`: Number of days per month in 1846-2007.

Examples

```
head(BrestSurgeMaxima)

# Time series plot of annual maxima surges
plot(rownames(BrestSurgeMaxima), BrestSurgeMaxima$maxima,
     ylab = "surge (cm)", xlab = "year", pch = 16)

# Time series plot of proportion of non-missing days
plot(rownames(BrestSurgeMaxima), BrestSurgeMaxima$notNA / BrestSurgeMaxima$n,
     ylab = "proportion of non-missing days", xlab = "year", pch = 16)

# Plot surges against the proportion of non-missing days
plot(BrestSurgeMaxima$notNA / BrestSurgeMaxima$n, BrestSurgeMaxima$maxima,
     ylab = "surge (cm)", xlab = "proportion of non-missing days", pch = 16)
```

BrestSurgeMissing — *Missing Values in Sea Surge Heights at Brest, France*

Description

Numbers of missing values in each month of the Brest sea surge heights data BrestSurgeMaxima.

Usage

```
BrestSurgeMissing
```

Format

BrestSurgeMissing is a data frame with 162 rows (years 1846 to 2007) and the 12 variables (one for each month of the year). Each value in the data frame gives the number of days for which the surge height data were missing in the month in question.

The row names of BrestSurgeMaxima are the years 1946:2007 and the column names are the abbreviated names of the months.

Source

The dataset Brest in the Renext R package, specifically Brest$OTmissing. Originally, the source was https://data.shom.fr/.

References

Deville Y. and Bardet L. (2023). Renext: Renewal Method for Extreme Values Extrapolation. R package version 3.1-4. doi:10.32614/CRAN.package.Renext

See Also

- BrestSurgeMaxima: Annual maxima surge heights at Brest, France.
- BrestSurgeDays: Number of days per month in 1846-2007.

Examples

```
head(BrestSurgeMissing)

# Proportion of missing values by year
propn_year <- rowSums(BrestSurgeMissing) /
  days_in_year(rownames(BrestSurgeMissing))
plot(rownames(BrestSurgeMissing), propn_year,
     ylab = "proportion of missing values", xlab = "year", pch = 16)

# Proportion of missing values by year and month
propn_year_month <- BrestSurgeMissing / BrestSurgeDays

# Proportion of missing values by month
plot(1:12, colMeans(propn_year_month), axes = FALSE,
        ylab = "proportion of missing values", xlab = "month", pch = 16)
axis(1, at = 1:12, labels = 1:12)
axis(2)
box()
```

---

confint_gev_methods *Methods for objects of class* "confint_gev"

Description

Methods for objects of class "confint_gev" returned from confint.evmissing.

Usage

```
## S3 method for class 'confint_gev'
print(x, ...)

## S3 method for class 'confint_gev'
plot(x, parm = c("mu", "sigma", "xi"), add = TRUE, digits = 2, ...)
```

Arguments

x	An object inheriting from class "confint_gev", an object returned after a call to confint.evmissing.
...	Further arguments. For print.confint_gev to pass arguments to print. For plot.confint_gev to pass graphical parameters to plot to create the initial plot of the profile log-likelihood.
parm	A character scalar specifying the parameter for which a profile log-likelihood is plotted. Must be a single component of c("mu", "sigma", "xi").
add	A logical scalar. If add = TRUE then the plot is annotated with a horizontal line indicating the critical value for the profile log-likelihood used to calculate the confidence limits, vertical lines indicating the values of these limits and a legend stating the confidence interval.

digits An integer. Passed to signif to round the confidence limits in the legend, if add = TRUE. The confidence level is hard-coded to be expressed to 3 significant figures.

Details

print.confint_gev. A numeric matrix with 2 columns giving the lower and upper confidence limits for the parameters specified by the argument parm in confint.evmissing. These columns are labelled as (1-level)/2 and 1-(1-level)/2, expressed as a percentage, by default 2.5% and 97.5%.

plot.confint_gev. A plot is produced of the profile log-likelihood for the parameter chosen by parm. Only the parameter values used to profile the log-likelihood in the call to confint.evmissing are included, so if faster = TRUE was used then the plot will not be of a smooth curve but will be triangular in the middle.

Value

print.confint_gev: the argument x is returned, invisibly.

plot.confint_gev: a numeric vector containing the confidence limits for the parameter requested in parm is returned invisibly.

Examples

See evmissing_methods.

See Also

gev_mle and evmissing_methods.

---

confint_return_level_methods

Methods for objects of class "confint_return_level"

Description

Methods for objects of class "confint_return_level" returned from confint.return_level.

Usage

```
## S3 method for class 'confint_return_level'
print(x, ...)

## S3 method for class 'confint_return_level'
plot(x, parm = 1, add = TRUE, digits = 2, ...)
```

Arguments

`x`	An object inheriting from class `"confint_return_level"`, a result of a call to `confint.return_level`.
`...`	Further arguments. For `print.confint_return_level` to pass arguments to `print`). For `plot.confint_return_level` to pass graphical parameters to `plot` to create the initial plot of the profile log-likelihood.
`parm`	An integer scalar. For which component, that is, which return level, in `x` we require a confidence interval.
`add`	A logical scalar. If `add = TRUE` then the plot is annotated with a horizontal line indicating the critical value for the profile log-likelihood used to calculate the confidence limits, vertical lines indicating the values of these limits and a legend stating the confidence interval.
`digits`	An integer. Passed to `signif` to round the confidence limits in the legend, if `add = TRUE`. The confidence level is hard-coded to be expressed to 3 significant figures.

Details

`print.confint_return_level`. A numeric matrix with 2 columns giving the lower and upper confidence limits for the parameters specified by the argument `parm` in `confint.return_level`. These columns are labelled as `(1-level)/2` and `1-(1-level)/2`, expressed as a percentage, by default `2.5%` and `97.5%`.

`plot.confint.return_level`. A plot is produced of the profile log-likelihood for the parameter chosen by `parm`.

Value

`print.confint_return_level`: the argument `x` is returned, invisibly.

`plot.confint_return_level`: a numeric vector containing the confidence interval for the return level chosen for the plot.

Examples

See `return_level_methods`.

See Also

`gev_mle`, `gev_return` and `return_level_methods`.

days — *Days in a Year or in a Month*

Description

Returns the number of days in each of a vector of years or months.

Usage

```
days_in_year(year)

days_in_month(year, month)
```

Arguments

`year`	An integer vector. The years of interest.
`month`	An integer vector. A subset of `1:12`.The months of interest.

Details

The length of the output vector is equal to the length of `month`. The argument `year` is recycled to the length of the output vector if necessary.

Value

A numeric vector of the numbers of days in each of the years in `year` or the months specified by `year` and `month`.

Examples

```
days_in_year(1999:2025)

days_in_month(2024, 1:12)
days_in_month(2025, 1:12)
days_in_month(2024:2025, 1:3)
```

evmissing_methods — *Methods for objects of class* `"evmissing"`

Description

Methods for objects of class `"evmissing"` returned from `gev_mle`.

Usage

```
## S3 method for class 'evmissing'
coef(object, ...)

## S3 method for class 'evmissing'
vcov(object, ...)

## S3 method for class 'evmissing'
nobs(object, ...)

## S3 method for class 'evmissing'
logLik(object, ...)

## S3 method for class 'evmissing'
summary(object, digits = max(3, getOption("digits") - 3L), ...)

## S3 method for class 'summary.evmissing'
print(x, ...)

## S3 method for class 'evmissing'
confint(
  object,
  parm = "all",
  level = 0.95,
  profile = FALSE,
  mult = 2,
  faster = FALSE,
  epsilon = 1e-04,
  ...
)

## S3 method for class 'evmissing'
plot(
  x,
  adjust = TRUE,
  which = c("pp", "qq", "return", "density"),
  m = c(2, 10, 100, 1000),
  level = 0.95,
  profile = TRUE,
  num,
  npy = 1,
  ...
)
```

Arguments

`object`	An object inheriting from class "evmissing", a result of a call to `gev_mle`.
`...`	Further arguments. Only used in the following cases.

- plot.evmissing: to pass graphical parameters to the graphical functions plot, matplot, abline, lines, matlines and points. In particular, col, lty and lwd may be used to control the colour, type and width of lines and pch the type of plotting symbol. All data points are coloured black in all plots, which cannot be changed.
- print.summary.evmissing: to pass arguments to print.

digits
: An integer. Passed to signif to round the values in the summary.

x
: An object returned by summary.evmissing.

parm
: A character vector specifying the parameters for which confidence intervals are to be calculated. The default, which = "all", produces confidence intervals for all the parameters, that is, μ, σ and ξ. Otherwise, parm must be a subset of c("mu", "sigma", "xi").

level
: The confidence level required. A numeric scalar in (0, 1).

profile
: A logical scalar. If TRUE then confidence intervals based on a profile log-likelihood are returned. If FALSE then intervals based on approximate large sample normal theory, which are symmetric about the MLE, are returned.

mult
: A positive numeric scalar. Controls the increment by which the parameter of interest is increased/decreased when profiling above/below its MLE. The increment is mult * se / 100 where se is the estimated standard error of the estimator of the parameter. Decreasing mult profiles at more points but will be slower. The default, mult = 2 should be sufficiently small to produce a smooth looking plot of the profile log-likelihood using plot.confint_gev. To estimate the confidence limits more quickly, the value of mult can be increased and/or the argument faster set to TRUE.

faster
: A logical scalar. If faster = TRUE then the profiling of the log-likelihood in search of a lower (upper) confidence limit is started at the corresponding symmetric lower (upper) confidence limit.

epsilon
: Only relevant if profile = TRUE. A numeric vector of values that determine the accuracy of the confidence limits. epsilon is recycled to the length of the parameter vector parm.

  - If epsilon[i] > 0 then this value is passed as the argument epsilon to the itp::itp function, which estimates the parameter values for which the profile log-likelihood for parameter i drops to the value that defines the confidence limits, once profiling has been successful in finding an interval within which this value lies.
  - If epsilon[i] < 0 monotonic cubic spline interpolation is used, which will tend to be faster.
  - If epsilon[i] = 0 then linear interpolation is used, which will be faster still.

adjust
: If adjust = TRUE then the diagnostic plots produced by plot.evmissing are adjusted for the number of non-missing observations contributing to each block maximum. Otherwise, no adjustment is made.

which
: If supplied, this must either be a character scalar, one of "pp", "qq", "return" or "density" or a numeric scalar in 1:4, with 1 corresponding to "pp" etc. If which is missing then all four plots are produced in a 2 by 2 display.

`m` A numeric vector of return periods to label on the horizontal axis of the **return level plot**. Along with the data, the smallest and largest return period values in `m` influence the range of return periods for which return level estimates are plotted. All values in `m` must be greater than 1.

`num` An integer scalar. The number of return level estimates to calculate to produce the return level curve and pointwise confidence limits in the **return level plot**. The default setting is approximately 5 times `log(max(m), base = 10)`. If `profile = TRUE` then reducing `num` will speed up the calculation of the confidence limits, at the expense of a reduction in smoothness of the curves.

`npy` A numeric scalar. The number n_{py} of block maxima per year. If the blocks are of length 1 year then `npy = 1`. This is only used in the **return level plot**.

Details

The plots produced by `plot.evmissing` are of a similar form to the visual diagnostics is the `ismev` package and described in Coles (2001), that is, a probability plot (`which = "pp"` or `which = 1`), a quantile plot (`which ="qq"` of `which = 2`), a return level plot (`which = "return"` or `which = 3`) and a histogram of block maxima with a fitted GEV density superimposed (`which = "density"` or `which = 4`). Pointwise confidence bands of level `level` are added to the probability plot and quantile plot.

The default setting for confidence intervals for a return level plot produced by `plot.evmissing` is `profile = TRUE`, which uses `gev_return` and `confint.return_level`. The plot takes longer to produce, but it avoids the unrealistic feature of the lower confidence limits decreasing as we extrapolate to long return periods.

If `adjust = TRUE` then the empirical values based on the observed block maxima are adjusted for the number of non-missing raw observations in each block based on the fitted GEV parameter values for reduced block sizes. Passing `adjust = FALSE` is not sensible, but, if there are missing data, then it can serve to show that making the adjustment is necessary to give the correct impression of how well the model has fitted the data.

For `confint.evmissing`, the default, `epsilon = -1`, should work well enough in most circumstances, but to achieve a specific accuracy set `epsilon` to be a small positive value, for example, `epsilon = 1e-4`.

Value

`coef.evmissing`: a numeric vector of length 3 with names `c("mu", "sigma", "xi")`. The MLEs of the parameters μ, σ and ξ.

`vcov.evmissing`: a 3×3 matrix with row and column names `c("mu", "sigma", "xi")`. The estimated variance-covariance matrix for the model parameters μ, σ and ξ.

`nobs.evmissing`: a numeric scalar. The number of maxima used in the model fit.

`logLik.evmissing`: an object of class `"logLik"`: a numeric scalar with value equal to the maximised log-likelihood. The returned object also has attributes `nobs`, the number of maxima used in the model fit and `"df"` (degrees of freedom), which is equal to the number of total number of parameters estimated (3).

`summary.evmissing`: an object with class `"summary.evmissing"` containing the original function call and a matrix of estimates and estimated standard errors with row names `c("mu", "sigma", "xi")`. The object is printed by `print.summary.evmissing`.

print.summary.evmissing: the argument x is returned, invisibly.

confint.evmissing: an object of class c("confint_gev", "evmissing"). A numeric matrix with 2 columns giving the lower and upper confidence limits for each parameter. These columns are labelled as (1-level)/2 and 1-(1-level)/2, expressed as a percentage, by default 2.5% and 97.5%. The row names are the names of the parameters supplied in parm. The ordering "mu", "sigma", "xi" is retained regardless of the ordering of the parameters in parm. If profile = TRUE then the returned object has extra attributes crit, level and for_plot. The latter is a named list of length 3 with components mu, sigma and xi. Each components is a 2-column numeric matrix. The first column (named mu_values etc) contains values of the parameter and the second column the corresponding values of the profile log-likelihood. The profile log-likelihood is equal to the attribute crit at the limits of the confidence interval. The attribute level is the input argument level.

plot.evmissing: if a return level plot has been requested, a 3-column matrix containing the values plotted in the return level plot. Column 2 contains the estimated return levels and columns 1 and 3 the lower and upper confidence limits.

References

Coles, S. G. (2001) *An Introduction to Statistical Modeling of Extreme Values*, Springer-Verlag, London. doi:10.1007/9781447136750_3

Heffernan, J. E. and Stephenson, A. G. (2018). *ismev: An Introduction to Statistical Modeling of Extreme Values*. R package version 1.42. doi:10.32614/CRAN.package.ismev

See Also

gev_mle and confint_gev_methods.

Examples

```
## Plymouth ozone data

# Make adjustment for the numbers of non-missing values per block
fit <- gev_mle(PlymouthOzoneMaxima)
coef(fit)
vcov(fit)
nobs(fit)
logLik(fit)
summary(fit)

## Model diagnostic plots

# When profile = FALSE the return confidence limits are unrealistic
# for long return periods
plot(fit, profile = FALSE)

# Create the return level plot only
# When profile = TRUE (the default) the confidence limits are fine
# but the plot takes longer
# For speed, we reduce the number, num, of points used to plot the curves
plot(fit, which = 3, num = 8)
```

```
# If we do not reflect the adjustment in the plot then it gives a false
# impression of how well the model has fitted the data
plot(fit, adjust = FALSE, profile = FALSE)

## Confidence intervals

# Confidence limits that are symmetric about the respective MLEs
confint(fit)

# Calling confint to produce a smooth profile log-likelihood plot
x <- confint(fit, profile = TRUE)
x
plot(x, parm = "xi")

# Doing this more quickly when we only want the approximate confidence limits
x <- confint(fit, profile = TRUE, mult = 32, faster = TRUE)
x
plot(x, parm = "xi", type = "b")
```

---

gev_bayes *GEV Bayesian Inference with Adjustment for Missing Data*

Description

Performs Bayesian inference using a GEV distribution using block maxima, with the option to make an adjustment for the numbers of non-missing raw values in each block.

Usage

```
gev_bayes(
  data,
  block_length,
  block,
  adjust = TRUE,
  discard = 0,
  init = "quartiles",
  prior = revdbayes::set_prior(prior = "flat", model = "gev"),
  n = 1000,
  ...
)
```

Arguments

`data` Either

- a numeric vector containing a time series of raw data,
- an object returned from `block_maxima`, a list with components `maxima`, `notNA` and `n`,

- a data frame or named list containing the same information, that is, the variables maxima, notNA and n, as an object returned from block_maxima, such as the data frame BrestSurgeMaxima.

block_length
: A numeric scalar. Used calculate the maxima of disjoint blocks of block_length contiguous values in the vector data. If length(data) is not an integer multiple of block_length then the values at the end of data that do not constitute a complete block of length block_length are discarded, without warning.

block
: A numeric vector with the same length as data. The value of block[i] indicates the block into which data[i] falls. For example, block could provide the year in which observation i was observed.

adjust
: A logical scalar or a numeric scalar in [0, 100].
  - If adjust = TRUE then the adjustment, described in **Details**, for the numbers of non-missing values underlying each block maximum is performed.
  - If adjust = FALSE then no adjustment is made, that is, the block maxima are treated as if the underlying raw data have no missing values.

discard
: A numeric scalar. Any block maximum for which greater than discard percent of the underlying raw values were missing is discarded. Whether or not an adjustment for missingness is made for the block maxima that remain is determined by adjust.

init
: Either a character scalar, one of "quartiles" or "moments", or a numeric vector of length 3 giving initial estimates of the GEV location, scale and shape parameters: μ, σ and ξ. If init = "quartiles" then initial estimates of μ and σ are based on sample quartiles of block maxima, ignoring the underlying numbers of non-missing raw data, and a value of 0 for ξ. If init = "moments" then instead we use the sample mean and variance of these maxima and an initial value of 0.1 for ξ.

prior
: Specifies a prior distribution for the GEV parameters. This is most easily set using revdbayes::set_prior. The default is a prior $\pi(\mu, \sigma, \xi) \propto \sigma^{-1}$ for $\sigma > 0$. See revdbayes::set_prior for details.

n
: A non-negative integer. The number of values to simulate from the posterior distribution for (μ, σ, ξ).

...
: Further arguments to be passed to rust::ru.

Details

The likelihood described in gev_mle is combined with the prior density provided by prior to produce, up to proportionality, a posterior density for (μ, σ, ξ).

A function to evaluate the log-posterior is passed to rust::ru to simulate a random sample from this posterior distribution using the generalised ratio-of-uniforms method, using relocation of the mode of the density to the origin to increase efficiency. The value of init is used as an initial estimate in a search for the posterior mode. Arguments to rust::ru can be passed via .... The default setting is trans = "none", that is, no transformation of the margins, and rotate = TRUE, rotation of the parameter axes to improve isotropy with a view to increasing efficiency.

Value

An object returned from `rust::ru`. The following components are added to this list

- `model`: = "gev".
- `data,prior`: the inputs `data` and `prior`.
- `call`: the call to `gev_bayes`.
- `maxima`: the vector of block maxima used to fit the model.
- `notNA`: the number of non-missing raw values on which the maxima in `maxima` are based.
- `n`: the maximal block length, that is, the largest number of values that could have been observed in each of these blocks.
- `adjust`: a logical scalar indicating whether or not the adjustment in the **Details** section of `gev_mle` was performed. This is `TRUE` only if the input argument `adjust` was `TRUE`.
- `adjust,discard` : the values of these input arguments.

The class of the returned object is `c("evpost", "ru", "bayes", "evmissing")`. Objects of class `"evpost"` have `print`, `summary` and `plot` S3 methods.

Examples

```
## Simulate data with missing values

set.seed(24032025)
blocks <- 50
block_length <- 365

# Simulate raw data from an exponential distribution
sdata <- sim_data(blocks = blocks, block_length = block_length)

block_length <- sdata$block_length
# Sample from the posterior based on block maxima from full data
post1 <- gev_bayes(sdata$data_full, block_length = block_length)
summary(post1)

# Sample with adjustment for the number of non-missing values per block
post2 <- gev_bayes(sdata$data_miss, block_length = block_length)
summary(post2)

# Do not make the adjustment
post3 <- gev_bayes(sdata$data_miss, block_length = block_length,
                   adjust = FALSE)
summary(post3)

# Remove all block maxima with greater than 25% missing values and
# do not make the adjustment
post4 <- gev_bayes(sdata$data_miss, block_length = block_length,
                   adjust = FALSE, discard = 25)
summary(post4)

## Brest sea surge data
```

```
post <- gev_bayes(BrestSurgeMaxima)
summary(post)
plot(post)
```

---

gev_influence *GEV Influence Curves*

Description

Calculates influence function curves for maximum likelihood estimators of Generalised Extreme Value (GEV) parameters.

Usage

```
gev_influence(z, mu = 0, sigma = 1, xi = 0)

## S3 method for class 'gev_influence'
plot(x, xvar = c("z", "y"), sep_xi = TRUE, vlines, ...)
```

Arguments

`z`	A numeric vector. Values of normal quantiles z at which to calculate the GEV influence function. See **Details**.
`mu, sigma, xi`	Numeric scalars supplying the values of the GEV parameters μ, σ and ξ.
`x`	An object inheriting from class `"gev_influence"`, from a call to `gev_influence`.
`xvar`	A logical scalar. If `xvar = "z"` then the influence curves are plotted against the standard normal quantiles in `x[, "z"]`. If `xvar = "y"` then the influence curves are plotted against the corresponding GEV quantiles in `x[, "y"]`.
`sep_xi`	A logical scalar. If `sep_xi = TRUE` then separate vertical scales are used for ξ and (μ, σ). The scale for ξ appears on the left and the scale for (μ, σ) on the right.
`vlines`	A numeric vector. If `vlines` is supplied then black dashed vertical lines are added to the plot at the values in `vlines` on the horizontal axis. This might be used to indicate the values of certain observations in a dataset.
`...`	For `plot.gev_influence`: to pass graphical parameters to the graphical functions `matplot` and `legend`. The parameters `col, lty` and `lwd` can be used to control line colour, type and width, with the parameters in the usual order, that is, (μ, σ, ξ).

Details

An influence function measures the effect on a parameter estimator of changing one observation in a sample. See Hampel (2005) and, in the context of extreme value analyses of threshold exceedances, Davison and Smith (1990). Let $\theta = (\mu, \sigma, \xi)$. The GEV influence curve is defined, as a function of an observation y, by $IC(y) = \{IC_\mu(y), IC_\sigma(y), IC_\xi(y)\} = i_\theta^{-1} \mathrm{d}\ell(y;\theta)/\mathrm{d}\theta$, where $\ell(y;\theta)$ is the GEV log-likelihood function and i_θ^{-1} is the GEV expected information matrix. The value of y is related to z by $y = G^{-1}\{\Phi(z)\}$, where Φ and G are the distribution functions of a standard normal and GEV(μ, σ, ξ) distribution, respectively. Plotting influence curves on the standard normal z scale can help to aid interpretation.

The value of $IC(y)$ is invariant to the value of μ. For a given value of ξ, the values of $IC_\mu(y)$ and $IC_\sigma(y)$ scale with the scale parameter σ, that is, doubling σ doubles $IC_\mu(y)$ and $IC_\sigma(y)$. Typically, interest focuses on the shape parameter ξ, but if the input scale parameter σ is large then this may hide the influence of y on ξ. Therefore, the default setting of `plot.gev_influence`, `sep_xi = TRUE`, separates the plotting of the influence curve for ξ from those of μ and σ.

The example in **Examples** shows that extremely large block maxima have a strong positive influence on the MLE $\hat{\xi}$ and also that extremely small block maxima have a strong negative influence on $\hat{\xi}$,

Value

`gev_influence`: an object with class `c("gev_influence", "matrix", "array")`, a `length(z)` by 5 numeric matrix. The first two columns contain the input values in `z` and the corresponding values of `y`. Columns 3-5 contain the values of the GEV influence function for μ, σ and ξ respectively at the values of `z`.

`plot.gev_influence`: a list of the graphical parameters used in producing the plot, either the defaults or supplied via `...`, is returned invisibly.

References

Hampel, F. R., Ronchetti, E. M., Rousseeuw, P. J., and Stahel, W. A. (2005). Robust Statistics. Wiley-Interscience, New York. doi:10.1002/9781118186435

Davison, A. C. and Smith, R. L. (1990). Models for exceedances over high thresholds. Journal of the Royal Statistical Society: Series B (Methodological), 52(3):393–425. doi:10.1111/j.2517-6161.1990.tb01796.x

See Also

`gev_influence_rl`

Examples

```
# Influence curve for the default mu = 0, sigma = 1, xi = 0 case
z <- seq(from = -3, to = 3, by = 0.01)
inf <- gev_influence(z = z)
plot(inf)

# Influence curves based on the adjusted fit to the Plymouth ozone data
fit <- gev_mle(PlymouthOzoneMaxima)
pars <- coef(fit)
infp <- gev_influence(z = z, mu = pars[1], sigma = pars[2], xi = pars[3])
plot(infp)
```

gev_influence_rl *GEV Influence Curves for Return Levels*

Description

Calculates influence function curves for maximum likelihood estimators of 3 return levels based on Generalised Extreme Value (GEV) parameters.

Usage

```
gev_influence_rl(z, mu = 0, sigma = 1, xi = 0, m, npy = 1)

## S3 method for class 'gev_influence_rl'
plot(x, xvar = c("z", "y"), vlines, ...)
```

Arguments

`z`	A numeric vector. Values of normal quantiles z at which to calculate the GEV influence function. See **Details**.
`mu, sigma, xi`	Numeric scalars supplying the values of the GEV parameters μ, σ and ξ.
`m`	A numeric vector of length 3 containing 3 unique return periods in years. All entries in `m` must be greater than 1.
`npy`	A numeric scalar. The number n_{py} of block maxima per year. If the blocks are of length 1 year then `npy = 1`.
`x`	An object inheriting from class `"gev_influence_rl"`, returned from a call to `gev_influence_rl`.
`xvar`	A logical scalar. If `xvar = "z"` then the influence curves are plotted against the standard normal quantiles in `x[, "z"]`. If `xvar = "y"` then the influence curves are plotted against the corresponding GEV quantiles in `x[, "y"]`.
`vlines`	A numeric vector. If `vlines` is supplied then black dashed vertical lines are added to the plot at the values in `vlines` on the horizontal axis. This might be used to indicate the values of certain observations in a dataset.
`...`	For `plot.gev_influence_rl`: to pass graphical parameters to the graphical functions `matplot` and `legend`. The parameters `col, lty` and `lwd` can be used to control line colour, type and width, with the return levels in the order that they were supplied in `m`.

Details

See `gev_influence` for information about influence functions in general and influence curves for the parameters of a GEV distribution in particular. The GEV influence curves are reparameterised from (μ, σ, ξ) to the required return levels.

Value

`gev_influence_rl`: an object with class `c("gev_influence_rl", "matrix", "array")`, a `length(z)` by 5 numeric matrix. The first two columns contain the input values in `z` and the corresponding values of `y`. Columns 3-5 contain the values of the GEV influence function for the return levels in `m` respectively at the values of `z`.

`plot.gev_influence_rl`: a list of the graphical parameters used in producing the plot, either the defaults or supplied via `...`, is returned invisibly.

References

Hampel, F. R., Ronchetti, E. M., Rousseeuw, P. J., and Stahel, W. A. (2005). Robust Statistics. Wiley-Interscience, New York. doi:10.1002/9781118186435

Davison, A. C. and Smith, R. L. (1990). Models for exceedances over high thresholds. Journal of the Royal Statistical Society: Series B (Methodological), 52(3):393–425. doi:10.1111/j.2517-6161.1990.tb01796.x

See Also

`gev_influence`, `gev_return`

Examples

```
# Influence curves based on the adjusted fit to the Plymouth ozone data
z <- seq(from = -3, to = 3, by = 0.01)
fit <- gev_mle(PlymouthOzoneMaxima)
pars <- coef(fit)
m <- c(25, 50, 100)
infp <- gev_influence_rl(z = z, mu = pars[1], sigma = pars[2], xi = pars[3],
                         m = m)
plot(infp)
```

---

gev_mle *GEV ML Inference with Adjustment for Missing Data*

Description

Fits a GEV distribution to block maxima using maximum likelihood estimation, with the option to make an adjustment for the numbers of non-missing raw values in each block. The GEV location and scale parameters are adjusted to reflect the proportion of raw values that are missing.

Usage

```
gev_mle(
  data,
  block_length,
  block,
  adjust = TRUE,
```

```
  discard = 0,
  init = "quartiles",
  ...
)
```

Arguments

`data` Either

- a numeric vector containing a time series of raw data,
- an object returned from `block_maxima`, a list with components `maxima`, `notNA` and `n`,
- a data frame or named list containing the same information, that is, the variables `maxima`, `notNA` and `n`, as an object returned from `block_maxima`, such as the data frame `BrestSurgeMaxima`.

`block_length` A numeric scalar. Used calculate the maxima of disjoint blocks of `block_length` contiguous values in the vector `data`. If `length(data)` is not an integer multiple of `block_length` then the values at the end of `data` that do not constitute a complete block of length `block_length` are discarded, without warning.

`block` A numeric vector with the same length as `data`. The value of `block[i]` indicates the block into which `data[i]` falls. For example, `block` could provide the year in which observation `i` was observed.

`adjust` A logical scalar or a numeric scalar in `[0, 100]`.

- If `adjust = TRUE` then the adjustment, described in **Details**, for the numbers of non-missing values underlying each block maximum is performed.
- If `adjust = FALSE` then no adjustment is made, that is, the block maxima are treated as if the underlying raw data have no missing values.

`discard` A numeric scalar. Any block maximum for which greater than `discard` percent of the underlying raw values were missing is discarded. Whether or not an adjustment for missingness is made for the block maxima that remain is determined by `adjust`.

`init` Either a character scalar, one of `"quartiles"` or `"moments"`, or a numeric vector of length 3 giving initial estimates of the GEV location, scale and shape parameters: μ, σ and ξ. If `init = "quartiles"` then initial estimates of μ and σ are based on sample quartiles of block maxima, ignoring the underlying numbers of non-missing raw data, and a value of 0 for ξ. If `init = "moments"` then instead we use the sample mean and variance of these maxima and an initial value of 0.1 for ξ.

`...` Further arguments to be passed to `stats::optim`.

Details

If `data` is a numeric vector then exactly one of the arguments `block_length` or `block` must be supplied. The parameters are fitted using maximum likelihood estimation.

The adjustment for the numbers of non-missing values underlying the block maxima is based on the strong assumptions that missing values occur completely at random and that the raw data are independent and identically distributed. We suppose that a block maximum M_n based on a full (no missing raw values) block of length n has a GEV(μ, σ, ξ) distribution, with distribution function $G(x)$. Let n_i be the number of non-missing values in block i and let M_{n_i} denote the block maximum of such a block. We suppose that M_{n_i} has a GEV$(\mu(n_i), \sigma(n_i), \xi)$ distribution, where

$$\mu(n_i) = \mu + \sigma[(n_i/n)^\xi - 1]/\xi,$$
$$\sigma(n_i) = \sigma(n_i/n)^\xi.$$

These expressions are based on inferring the parameters of an approximate GEV distribution for M_{n_i} from its approximate distribution function $[G(x)]^{n_i/n}$.

A likelihood is constructed as the product of contributions from the maxima from distinct blocks, under the assumption that these maxima are independent. Let $\theta = (\mu, \sigma, \xi)$ and let $\ell_F(\underline{z}; \theta)$ denote the usual, unadjusted, GEV log-likelihood for the full-data case where there are no missing values. It can be shown that our adjusted log-likelihood $\ell(\theta, \underline{z})$ is given by

$$\ell(\theta, \underline{z}) = \ell_F(\underline{z}; \theta) - \sum_{i=1}^{n} p_i \log G(z_i; \theta)$$

where $p_i = 1 - n_i/n$ is the proportion of missing values in block i.

The negated log-likelihood is minimised using a call to `stats::optim` with `hessian = TRUE`. If `stats::optim` throws an error then a warning is produced and the returned object has `NA` values for the components `par`, `loglik`, `vcov` and `se` and an extra component `optim_error` containing the error message. If the estimated observed information matrix is singular then a warning is produced and the returned object has `NA` values for the components `vcov` and `se`.

Value

A list, returned from `stats::optim` (the MLEs are in the component `par`), with the additional components:

- `loglik`: value of the maximised log-likelihood.
- `vcov`: estimated variance-covariance matrix of the parameters.
- `se`: estimated standard errors of the parameters.
- `maxima`: the vector of block maxima used to fit the model.
- `notNA`: the number of non-missing raw values on which the maxima in `maxima` are based.
- `n`: the maximal block length, that is, the largest number of values that could have been observed in each of these blocks.
- `adjust`,`discard` : the values of these input arguments.

The call to `gev_mle` is provided in the attribute `"call"`.

The class of the returned object is `c("evmissing", "mle", "list")`.

Objects inheriting from class `"evmissing"` have `coef`, `logLik`, `nobs`, `summary`, `vcov` and `confint` methods. See `evmissing_methods`.

References

Simpson, E. S. and Northrop, P. J. (2026) Accounting for Missing Data When Modelling Block Maxima, *Environmetrics* **37**(2): e70075. doi:10.1002/env.70075.

Examples

```
## Simulate raw data from an exponential distribution

set.seed(13032025)
blocks <- 50
block_length <- 365
sdata <- sim_data(blocks = blocks, block_length = block_length)

# sdata$data_full have no missing values
# sdata$data_miss have had missing values created artificially

# Fit a GEV distribution to block maxima from the full data
fit1 <- gev_mle(sdata$data_full, block_length = sdata$block_length)
summary(fit1)

# An identical fit supplying the block indicator instead of block_length
fit1b <- gev_mle(sdata$data_full, block = sdata$block)
summary(fit1b)

# Make adjustment for the numbers of non-missing values per block
fit2 <- gev_mle(sdata$data_miss, block_length = sdata$block_length)
summary(fit2)

# Do not make the adjustment
fit3 <- gev_mle(sdata$data_miss, block_length = sdata$block_length,
                adjust = FALSE)
summary(fit3)

# Remove all block maxima with greater than 25% missing values and
# do not make the adjustment
fit4 <- gev_mle(sdata$data_miss, block_length = sdata$block_length,
                adjust = FALSE, discard = 25)
summary(fit4)

## Plymouth ozone data

# Calculate the values in Table 4 of Simpson and Northrop (2026)
# discard = 50 is chosen to discard data from 2001 and 2006
fit1 <- gev_mle(PlymouthOzoneMaxima)
fit2 <- gev_mle(PlymouthOzoneMaxima, adjust = FALSE)
fit3 <- gev_mle(PlymouthOzoneMaxima, discard = 50)
fit4 <- gev_mle(PlymouthOzoneMaxima, adjust = FALSE, discard = 50)
se <- function(x) return(sqrt(diag(vcov(x))))
MLEs <- cbind(coef(fit1), coef(fit2), coef(fit3), coef(fit4))
SEs <- cbind(se(fit1), se(fit2), se(fit3), se(fit4))
round(MLEs, 2)
round(SEs, 2)
```

`gev_return` *Return Level Inferences*

Description

Calculates point estimates of m-year return levels for fitted model objects returned from gev_mle.

Usage

```
gev_return(x, m = 100, npy = 1)
```

Arguments

x	An object inheriting from class evmissing returned from gev_mle.
m	A numeric vector. Values of m, the return periods of interest, in years.
npy	A numeric scalar. The number n_{py} of block maxima per year. If the blocks are of length 1 year then npy = 1.

Details

For $\xi \neq 0$, the m-year return level is given by $z_m = \mu + \sigma(y_p^{-\xi} - 1)/\xi$, where $y_p = -\log(1-p)$ and $p = 1 - (1 - 1/m)^{1/n_{py}}$. For $\xi = 0$, $z_m = \mu - \sigma \log y_p$. Equivalently, we could note that $z_m = \mu - \sigma BC(y_p, -\xi)$, where $BC(x, \lambda)$ is a Box-Cox transformation.

Value

An object with class c("return_level", "numeric", "evmissing"). A numeric vector containing the MLEs of the required return levels, with names indicating the return period. The fitted model object returned from gev_mle is included as an attribute called "gev_mle". The input arguments m and npy are also included as attributes as is the call to gev_return.

References

Coles, S. G. (2001) *An Introduction to Statistical Modeling of Extreme Values*, Springer-Verlag, London. doi:10.1007/9781447136750_3

See Also

return_level_methods for print, summary, coef, vcov and confint methods.

Examples

```
## Simulate raw data from an exponential distribution

set.seed(13032025)
blocks <- 50
block_length <- 365
sdata <- sim_data(blocks = blocks, block_length = block_length)

# sdata$data_full have no missing values
# sdata$data_miss have had missing values created artificially

# Fit a GEV distribution to block maxima from the full data
fit1 <- gev_mle(sdata$data_full, block_length = sdata$block_length)
```

```
summary(fit1)

# Make adjustment for the numbers of non-missing values per block
fit2 <- gev_mle(sdata$data_miss, block_length = sdata$block_length)
summary(fit2)

gev_return(fit1, m = c(100, 1000))
gev_return(fit2, m = c(100, 1000))

## Plymouth ozone data

fit <- gev_mle(PlymouthOzoneMaxima)
rl <- gev_return(fit, m = c(100, 200))

# Symmetric confidence intervals
sym <- confint(rl)

# Profile-based confidence intervals

prof <- confint(rl, profile = TRUE)
prof
plot(prof, digits = 4)
plot(prof, parm = 2, digits = 3)

# Doing this more quickly when we only care about the confidence limits
prof <- confint(rl, profile = TRUE, mult = 32, faster = TRUE)
plot(prof, digits = 3, type = "b")
plot(prof, parm = 2, digits = 3, type = "b")
```

`gev_ts` *GEV ML Inference with Adjustment for Missing Data (Stationary Sequences)*

Description

Fits a GEV distribution to block maxima using maximum likelihood estimation, making an adjustment for the locations of missing raw values in each block. The GEV location and scale parameters are adjusted to reflect the proportion of raw values that are missing and the time series dependence in the data.

Usage

```
gev_ts(
  data,
  block_length,
  block,
  pseudo = TRUE,
  sliding = FALSE,
  init = "quartiles",
```

```
  ...
)
```

Arguments

`data` Either

- a numeric vector containing a time series of raw data,
- an object returned from `block_maxima_ts` or `sliding_block_maxima_ts`, a list with components `maxima`, `notNA`, `n`, `whereNA`, `pseudo_maxima`, `full_maxima` and `partial_maxima`,
- a named list containing the same information, that is, the variables `maxima`, `notNA`, `n`, `whereNA`, `pseudo_maxima`, `full_maxima` and `partial_maxima` as an object returned from `block_maxima_ts` or `sliding_block_maxima_ts`.

There must be at least one full block of data, that is, at least one block for which no data are missing.

`block_length` A numeric scalar. Used calculate the maxima of disjoint blocks of `block_length` contiguous values in the vector `data`. If `sliding = FALSE` and if `length(data)` is not an integer multiple of `block_length`, then the values at the end of `data` that do not constitute a complete block of length `block_length` are discarded, without warning.

`block` A numeric vector with the same length as `data`. The value of `block[i]` indicates the block into which `data[i]` falls. For example, `block` could provide the year in which observation `i` was observed. Not applicable if `sliding = TRUE`. If `sliding = TRUE`, then `block_length` must be supplied.

`pseudo` A logical scalar. If `pseudo = TRUE` then the pseudo-maxima returned from `block_maxima_ts` are used to estimate the value of r_i for an incomplete, partially-observed block. See **Details**. If `pseudo = FALSE` then the ratio n_i/n is used, as in `gev_mle()`.

`sliding` A logical scalar. If `sliding = TRUE` then inferences are based on sliding block maxima returned by `sliding_block_maxima_ts` and `block_length` must be supplied. If `sliding = FALSE` then they are based on disjoint block maxima returned from `block_maxima_ts`.

`init` Either a character scalar, one of `"quartiles"` or `"moments"`, or a numeric vector of length 3 giving initial estimates of the GEV location, scale and shape parameters: μ, σ and ξ. If `init = "quartiles"` then initial estimates of μ and σ are based on sample quartiles of block maxima, ignoring the underlying numbers of non-missing raw data, and a value of 0 for ξ. If `init = "moments"` then instead we use the sample mean and variance of these maxima and an initial value of 0.1 for ξ.

`...` Further arguments to be passed to `stats::optim`.

Details

If `data` is a numeric vector then exactly one of the arguments `block_length` or `block` must be supplied if `sliding = FALSE` and only `block_length` can be supplied if `sliding = TRUE`. The parameters are fitted using maximum likelihood estimation.

The adjustment for the numbers of non-missing values underlying the block maxima is based on the strong assumption that missing values occur completely at random. We suppose that a block maximum M_n based on a full (no missing raw values) block of length n has a GEV(μ, σ, ξ) distribution, with distribution function $G(x)$. Let n_i be the number of non-missing values in block i and let M_{n_i} denote the block maximum of such a block. We suppose that, M_{n_i} has a GEV$(\mu(r_i), \sigma(r_i), \xi)$ distribution, where

$$\mu(r_i) = \mu + \sigma[r_i^{\xi} - 1]/\xi,$$

$$\sigma(r_i) = \sigma r_i^{\xi},$$

for some $n_i/n \leq r_i \leq 1$. These expressions are based on the M_{n_i} having approximately a GEV distribution with distribution function $G(x)^{r_i}$.

For a full block, $r_i = 1$. If `pseudo = TRUE`, then, for an incomplete, partially-observed block, the value of r_i is estimated using the pseudo-maxima returned from `block_maxima_ts` and the GEV distribution function based on the current value of (μ, σ, ξ) in the optimisation routine. Suppose that we have a vector M_i of pseudo-maxima resulting from a particular incomplete block i. It can be shown that the components of $V_i = -\log G(M_i)$ each have an exponential distribution with mean $1/r_i$. We estimate r_i using the reciprocal of the mean of the values in V_i.

The negated log-likelihood is minimised using a call to `stats::optim` with `hessian = TRUE`. If `stats::optim` throws an error then a warning is produced and the returned object has `NA` values for the components `par`, `loglik`, `vcov` and `se` and an extra component `optim_error` containing the error message. If the estimated observed information matrix is singular then a warning is produced and the returned object has `NA` values for the components `vcov` and `se`.

Value

A list, returned from `stats::optim` (the MLEs are in the component `par`), with the additional components:

- `loglik`: value of the maximised log-likelihood.
- `vcov`: estimated variance-covariance matrix of the parameters.
- `se`: estimated standard errors of the parameters.
- `maxima`: the vector of block maxima used to fit the model.
- `notNA`: the number of non-missing raw values on which the maxima in `maxima` are based.
- `n`: the maximal block length, that is, the largest number of values that could have been observed in each of these blocks.
- `rvec`: a vector of the values used for $r_1, ..., r_b$, where b is the number of blocks. The content depends on the argument `pseudo`.
- `rhats`: if `pseudo = TRUE`, a vector of the subset of `rvec` for partially-observed blocks only. The attributes `"propn_notNA"` and `"unconstrained"` give, respectively, the values of n_i/n for these blocks and the estimates of r_i before they are constrained to lie in the interval $[n_i/n, 1]$.
- `sliding`: the input argument `sliding`.

The call to `gev_ts` is provided in the attribute `"call"`.

See Also

gev_mle provides an adjustment for missing data in the case where the raw data can be assumed to be independent and identically distributed.

Examples

```
set.seed(1632026)
blocks <- 50
block_length <- 90
missing_args <- list(p0miss = 0.5, min = 0, max = 0.4)
sdata <- sim_data(blocks = blocks, block_length = block_length,
                  missing_args = missing_args)

# Using disjoint blocks
pt <- gev_ts(sdata$data_miss, block_length = 90, pseudo = TRUE)
pf <- gev_ts(sdata$data_miss, block_length = 90, pseudo = FALSE)
pf2 <- gev_mle(sdata$data_miss, block_length = 90)

# Using sliding blocks
## Not run:
pts <- gev_ts(sdata$data_miss, block_length = 90, pseudo = TRUE,
              sliding = TRUE)
pfs <- gev_ts(sdata$data_miss, block_length = 90, pseudo = FALSE,
              sliding = TRUE)

## End(Not run)
```

gev_weighted *Weighted GEV ML Inference with Adjustment for Missing Data*

Description

Fits a GEV distribution to block maxima using maximum likelihood estimation, with the option to make an adjustment for the numbers of non-missing raw values in each block using one of the two weighting schemes proposed in McVittie and Murphy (2025).

Usage

```
gev_weighted(data, scheme = 1, block_length, block, init = "quartiles", ...)
```

Arguments

data
: Either
  - a numeric vector containing a time series of raw data,
  - an object returned from block_maxima, a list with components maxima, notNA and n,

- a data frame or named list containing the same information (variables maxima, notNA and n) as an object returned from block_maxima, such as the data frame BrestSurgeMaxima.

scheme
: A numeric scalar. Pass scheme = 1 for the first weighting scheme, or scheme = 2 for the second weighting scheme, in McVittie and Murphy (2025). Any value other than 1 or 2 result in an unweighted fit, that is, all weight are set to 1. See **Details**.

block_length
: A numeric scalar. Used calculate the maxima of disjoint blocks of block_length contiguous values in the vector data. If length(data) is not an integer multiple of block_length then the values at the end of data that do not constitute a complete block of length block_length are discarded, without warning.

block
: A numeric vector with the same length as data. The value of block[i] indicates the block into which data[i] falls. For example, block could provide the year in which observation i was observed.

init
: Either a character scalar, one of "quartiles" or "moments", or a numeric vector of length 3 giving initial estimates of the GEV location, scale and shape parameters: μ, σ and ξ. If init = "quartiles" then initial estimates of μ and σ are based on sample quartiles of block maxima, ignoring the underlying numbers of non-missing raw data, and a value of 0 for ξ. If init = "moments" then instead we use the sample mean and variance of these maxima and an initial value of 0.1 for ξ.

...
: Further arguments to be passed to stats::optim.

Details

Suppose that a full (no missing values) block will contain n raw values. Let n_i be the number of non-missing values, and m_i the observed block maximum, in block i. The contribution of block maximum m_i to the GEV log-likelihood is weighted (multiplied) by the weight w_i. The set of weights depends on the weighting scheme chosen by scheme as follows.

- If scheme = 1 then $w_i = n_i/n$, for $i = 1, ..., n$.
- If scheme = 2 then $w_i = \hat{F}(m_i)^{n-n_i}$, for $i = 1, ..., n$, where $\hat{F}$ is the empirical distribution function of $m_1, ..., m_n$.

For any other value of scheme all weights are set to 1, that is, an unweighted fit is performed.

For each weighting scheme, the larger the number $n - n_i$ of missing values the smaller the weight. See McVittie and Murphy (2025) for further details.

Value

A list, returned from stats::optim (the MLEs are in the component par), with the additional components:

- loglik: value of the maximised log-likelihood.
- vcov: estimated variance-covariance matrix of the parameters.
- se: estimated standard errors of the parameters.
- maxima: the vector of block maxima used to fit the model.

- `notNA`: the number of non-missing raw values on which the maxima in `maxima` are based.
- `n`: the maximal block length, that is, the largest number of values that could have been observed in each of these blocks.
- `weights`: the weights used in the fitting.

The call to `gev_mle` is provided in the attribute `"call"`.

The class of the returned object is `c("evmissing", "weighted_mle", "list")`.

Objects inheriting from class `"evmissing"` have `coef`, `logLik`, `nobs`, `summary`, `vcov` and `confint` methods. See `evmissing_methods`.

References

McVittie, J. H. and Murphy, O. A. (2025) Weighted Parameter Estimators of the Generalized Extreme Value Distribution in the Presence of Missing Observations. doi:10.48550/arXiv.2506.15964

Examples

```
## Simulate raw data from an exponential distribution

set.seed(13032025)
blocks <- 50
block_length <- 365
sdata <- sim_data(blocks = blocks, block_length = block_length)

# sdata$data_full have no missing values
# sdata$data_miss have had missing values created artificially

## Fits to full data: fit0, fit 1 and fit2 should give the same results

# Fit a GEV distribution to block maxima from the full data
fit0 <- gev_mle(sdata$data_full, block_length = sdata$block_length)
summary(fit0)

# Fit to the full data using weighting scheme 1
fit1 <- gev_weighted(sdata$data_full, scheme = 1,
                     block_length = sdata$block_length)
summary(fit1)

# Fit to the full data using weighting scheme 2
fit2 <- gev_weighted(sdata$data_full, scheme = 2,
                     block_length = sdata$block_length)
summary(fit2)

## Fits to the data with missings

#  Make adjustment for the numbers of non-missing values per block
fit0 <- gev_mle(sdata$data_miss, block_length = sdata$block_length)
summary(fit0)

# Make adjustment using weighting scheme 1
fit1 <- gev_weighted(sdata$data_miss, scheme = 1,
```

```
                     block_length = sdata$block_length)
summary(fit1)

# Make adjustment using weighting scheme 2
fit2 <- gev_weighted(sdata$data_miss, scheme = 2,
                     block_length = sdata$block_length)
summary(fit2)
```

---

plot.block_maxima *Plot method for objects inheriting from class* "block_maxima"

Description

Plot method for objects inheriting from "block_maxima" returned from block_maxima, block_maxima_ts or sliding_block_maxima_ts.

Usage

```
## S3 method for class 'block_maxima'
plot(x, which = 1, ...)
```

Arguments

x	An object inheriting from class "block_maxima".
which	If which = 1 then the sliding block maxima are plotted against block number. If which = 2 then the sliding block maxima are plotted against the proportion of non-missing raw values.
...	Further arguments to plot.

Details

When which = 1 we obtain a time series plot in which there are periods where the value does not change. When which = 2 we expect to see that the sliding block maximum tends to be smaller for blocks with a larger proportion of missing values.

Value

Nothing is returned.

Examples

```
### Plymouth Ozone Data

## Sliding maxima
# Time series plots of sliding block maxima
plot(PlymouthOzoneSlidingMaxima)
# Plot maxima against the proportion of non-missing daily values
plot(PlymouthOzoneSlidingMaxima, which = 2)
```

```
## Disjoint maxima
bm <- block_maxima(PlymouthOzone$Ozone, block = PlymouthOzone$Year)
# Time series plots of block maxima
plot(bm)
# Plot maxima against the proportion of non-missing daily values
plot(bm, which = 2)
```

PlymouthOzone *Ozone Levels at Plymouth, UK*

Description

Daily maximum ozone levels at Plymouth in London (UK) for the years 1998-2024 inclusive.

Usage

```
PlymouthOzone
```

Format

`PlymouthOzone` is a data frame with 9862 rows and the 3 variables:

- `Date`: with class `"Date"` in the format `YYYY-MM-DD`.
- `Year`: Values in 1998-2024.
- `Ozone`: daily maximum ozone level in μg/m^3.

Source

The Department for Environment Food and Rural Affair (DEFRA). The Plymouth Centre monitoring site at the UK-AIR database Data Selector.

See Also

`PlymouthOzoneMaxima` for the annual maxima and numbers of missing values per year.

Examples

```
head(PlymouthOzone)

# Time series plot of annual maxima ozone levels
plot(PlymouthOzone$Date, PlymouthOzone$Ozone, xlab = "year",
     ylab = "ozone (micrograms / metre cubed)", pch = 16)
```

PlymouthOzoneMaxima *Annual Maxima Ozone Levels at Plymouth, UK*

Description

Annual maxima of daily maximum ozone levels at Plymouth in Devon (UK) for the years 1998-2024 inclusive.

Usage

```
PlymouthOzoneMaxima
```

Format

`PlymouthOzoneMaxima` is a data frame with 27 rows (years 1998 to 2024) and the 4 variables:

- `maxima`: annual maximum ozone level in μg/m^3.
- `notNA` : the number of days of the year for which raw data were available.
- `n` : the number of days in the year (365 or 366).
- `block` : a block number of 1 for year 1998 through to 27 for year 2024.

The row names of `PlymouthOzoneMaxima` are the years `1998:2024`. The raw data are missing for approximately 10% of the days.

Source

The Department for Environment Food and Rural Affair (DEFRA). The Plymouth Centre monitoring site at the UK-AIR database Data Selector.

See Also

`PlymouthOzone` for the raw time series.

Examples

```
head(PlymouthOzoneMaxima)

# Time series plot of annual maxima ozone levels
plot(rownames(PlymouthOzoneMaxima), PlymouthOzoneMaxima$maxima,
     ylab = "ozone (micrograms / metre cubed)", xlab = "year", pch = 16)

# Time series plot of proportion of non-missing days
plot(rownames(PlymouthOzoneMaxima),
     PlymouthOzoneMaxima$notNA / PlymouthOzoneMaxima$n,
     ylab = "proportion of non-missing days", xlab = "year", pch = 16)

# Plot ozone levels against the proportion of non-missing days
plot(PlymouthOzoneMaxima$notNA / PlymouthOzoneMaxima$n,
```

```
PlymouthOzoneMaxima$maxima,
ylab = "ozone (micrograms / metre cubed)",
xlab = "proportion of non-missing days", pch = 16)
```

PlymouthOzoneSlidingMaxima *Sliding Maxima of Ozone Levels at Plymouth, UK*

Description

Maxima of sliding (overlapping) block of length 365 days of daily maximum ozone levels at Plymouth in Devon (UK) for the years 1998-2024 inclusive.

Usage

```
PlymouthOzoneSlidingMaxima
```

Format

`PlymouthOzoneSlidingMaxima` is an object returned from `sliding_block_maxima_ts`, a list inheriting from class `sliding_block_maxima_ts`. It was created using `sliding_block_maxima_ts(PlymouthOzone$Ozone, block_length = 365)`. This includes components

- `full_maxima`: a numeric vector of 130 full-block maxima, all equal to 143 μg/m^3.
- `partial_maxima`: a numeric vector of 9368 partial-block maxima.
- `maxima`: a numeric vector of all maxima.

See `sliding_block_maxima_ts` for a description of the other components.

Source

The Department for Environment Food and Rural Affair (DEFRA). The Plymouth Centre monitoring site at the UK-AIR database Data Selector.

See Also

`PlymouthOzone` for the raw time series.

Examples

```
# Time series plots of sliding block maxima
plot(PlymouthOzoneSlidingMaxima)

# Plot maxima against the proportion of non-missing daily values
plot(PlymouthOzoneSlidingMaxima, which = 2)
```

return_level_methods *Methods for objects of class* "return_level"

Description

Methods for objects of class "return_level" returned from gev_return.

Usage

```
## S3 method for class 'return_level'
coef(object, ...)

## S3 method for class 'return_level'
print(x, ...)

## S3 method for class 'return_level'
vcov(object, ...)

## S3 method for class 'return_level'
summary(object, digits = max(3, getOption("digits") - 3L), ...)

## S3 method for class 'summary.return_level'
print(x, ...)

## S3 method for class 'return_level'
confint(
  object,
  parm = 1:length(object),
  level = 0.95,
  profile = FALSE,
  mult = 2,
  faster = FALSE,
  epsilon = 1e-04,
  ...
)
```

Arguments

object, x	An object inheriting from class "return_level", a result of a call to gev_return. object is a named numeric vector of MLEs of return levels. For print.summary.return_level, this is an object returned by the function summary.return_level.
...	Further arguments. Only used for print.summary.return_level to pass arguments to print.
digits	An integer. Passed to signif to round the values in the summary.

parm — A numeric vector. For which components, that is, which return levels, in `object` we require a confidence interval.

level — The confidence level required. A numeric scalar in (0, 1).

profile — A logical scalar. If `TRUE` then confidence intervals based on a profile log-likelihood are returned. If `FALSE` then intervals based on approximate large sample normal theory, which are symmetric about the MLE, are returned.

mult — A positive numeric scalar. Controls the increment by which the parameter of interest is increased/decreased when profiling above/below its MLE. The increment is `mult * se / 100` where `se` is the estimated standard error of the estimator of the return level. Decreasing `mult` profiles at more points but will be slower.

faster — A logical scalar. If `faster = TRUE` then the profiling of the log-likelihood in search of a lower (upper) confidence limit is started at the corresponding symmetric lower (upper) confidence limit.

epsilon — Only relevant if `profile = TRUE`. A numeric vector of values that determine the accuracy of the confidence limits. `epsilon` is recycled to the length of the parameter vector `parm`.

- If `epsilon[i] > 0` then this value is passed as the argument `epsilon` to the `itp::itp` function, which estimates the parameter values for which the profile log-likelihood for parameter `i` drops to the value that defines the confidence limits, once profiling has been successful in finding an interval within which this value lies.
- If `epsilon[i] < 0` monotonic cubic spline interpolation is used, which will tend to be faster.
- If `epsilon[i] = 0` then linear interpolation is used, which will be faster still.

Details

For `confint.return_level`, the default, `epsilon = -1`, should work well enough in most circumstances, but to achieve a specific accuracy set `epsilon` to be a small positive value, for example, `epsilon = 1e-4`.

Value

`print.return_level` and `coef.return_level`: a numeric vector containing the MLEs of return return levels.

`vcov.return_level`: a `length(object)` by `length(object)` matrix with row and column names indicating the return periods of the return levels. The estimated variance-covariance matrix for the return levels in `object`. The diagonal elements give the estimated variances associated with the individual return level estimates.

`summary.return_level`: an object containing the original function call and a matrix of estimates of return levels and associated estimated standard errors with row names indicating the respective return periods. The object is printed by `print.summary.return_level`.

`print.summary.return_level`: the argument `x` is returned, invisibly.

`confint.return_level`: an object of class `c("confint_return_level", "evmissing")`. A numeric matrix with 2 columns giving the lower and upper confidence limits for each return level.

These columns are labelled as (1-level)/2 and 1-(1-level)/2, expressed as a percentage, by default 2.5% and 97.5%. The row names indicate the return levels. If profile = TRUE then the returned object has extra attributes crit, level and for_plot. The latter is a named list of length parm with components named after the return periods. Each components is a 2-column numeric matrix. The first column contains values of the return level and the second column the corresponding values of the profile log-likelihood. The profile log-likelihood is equal to the attribute crit at the limits of the confidence interval. The attribute level is the input argument level.

See Also

gev_mle and gev_return, for examples of the use of confint.return_level.

Examples

```
## Plymouth ozone data

# See ?gev_return for confidence intervals for return levels
fit <- gev_mle(PlymouthOzoneMaxima)
rl <- gev_return(fit, m = c(100, 200))
rl
vcov(rl)
summary(rl)
```

---

sim_data *Simulate Raw Data*

Description

Simulates data from a user-supplied distribution and creates missing values artificially. Functions mcar and mcar2 provides an example mechanisms for doing this based on a Missing Completely At Random (MCAR) assumption.

Usage

```
sim_data(
  blocks = 50,
  block_length = 365,
  distn = "exp",
  missing_fn = mcar,
  missing_args = formals(missing_fn)$missing_args,
  ...
)

mcar(
  sim_data,
  blocks,
  block_length,
  missing_args = list(p0miss = 0, min = 0, max = 0.5)
```

```
)

mcar2(sim_data, missing_args = list(pmiss = 0.5))
```

Arguments

blocks	A numeric scalar. The number of blocks of data required. Usually, this will be a positive integer, but `blocks = 0` returns a list containing in the input arguments, in particular, `distn`, `distn_args` and `block_length`. This feature is provided so that a simulation setup could be replicated in without actually simulating data.
block_length	A numeric scalar. The number of raw observations per block.
distn	A character scalar. Specifies the distribution from which raw data are simulated. The name in the `xxx` part of the `dxxx, pxxx, qxxx` and `rxxx` distributional functions in the `stats` package. See `stats::Distributions`.
missing_fn	A function to simulate the positions of the missing values within each block year. See **Details**.
missing_args	Arguments to be passed to `missing_fn`. If `missing_fn` is `mcar` then a subset of `p0miss`, `min` and `max` may be supplied in the list `missing_args`. The values of the remaining components will be set at their default values.
...	Further arguments to the function `stats::rxxx`. The argument `n` is set within `sim_data` to be equal to `block_length * blocks`.
sim_data	A numeric vector of raw observations into, some of which will be made missing.

Details

The function `missing_fn` must return a, possibly empty, subset of `c(1, 2, ..., block_length)`. This function is applied within each simulated block, independently of other blocks.

The default function `mcar` simulates the numbers of missing values in the blocks as follows.

- A proportion `p0miss` of the blocks have **no** missing values.
- In the other blocks, the number of missing values is `ceiling(prop_miss * block_length)`, where `prob_miss` is a value simulated from a Uniform(`min`, `max`) distribution. The positions of these missing values within the block is random.

The function `mcar2` identifies at random a proportion `pmiss` of the simulated raw observations to become missing.

Care may need to be taken if these simulated data are used as input to `gev_mle` using an approach that discards block maxima based on more than a certain percentage of missing values, that is, with `discard > 0`. For example, using the default argument `blocks = 50` and `missing_fn = mcar`, with its default `missing_args`, may result in a sample size of retained block maxima that contains insufficient information to make reliable inferences, leading to difficulties finding an appropriate MLE for the shape parameter ξ and/or a singular observed information matrix.

Value

If `blocks > 0`, a list with the following components:

- `data_full`: simulated raw data with no missing values.

- data_miss: simulated data after missing values have been created.
- blocks, block_length: the respective input values of blocks and block_length.
- block: a block indicator vector, suitable as an argument to gev_mle.
- distn: the input argument distn.
- distn_args: further arguments to stats::rxxx supplied via ....

If blocks = 0, a list containing all the inputs arguments.

See Also

gev_mle and gev_bayes.

Examples

```
# Using missing_fn = mcar
sdata <- sim_data()

# Using missing_fn = mcar2
sdata2 <- sim_data(missing_fn = mcar2)
```

---

sliding_block_maxima_ts
Sliding Block Maxima for a Time Series

Description

Extracts sliding block maxima and missing value information for each block. Works like block_maxima_ts but returns information for sliding (overlapping) blocks rather than disjoint blocks and defines blocks using only block length, that is, there is no block argument.

Usage

```
sliding_block_maxima_ts(data, block_length)
```

Arguments

data	A numeric vector containing a time series of raw data.
block_length	A numeric scalar. Used calculate the maxima of sliding blocks of block_length contiguous values in the vector data.

Details

The maxima are calculated for **all** blocks of length block_length present in data, starting with the first block data[1:block_length] and sliding the block repeatedly by one observation until reaching the final block data[(length(data) - block_length + 1):length(data)].

Value

A list, with class c("list", "block_maxima", "sliding", "evmissing"), containing the following components:

- maxima: the block maxima.
- notNA: the numbers of non-missing observations in each block.
- n: the maximal block length, that is, the largest number of values that could have been observed in each block.
- whereNA: a named list containing, for each block, the positions of any missing values in the block. For example, if (only) the first and fifth observations in block 3 are missing then the third component (named block3) of whereNA is c(1, 5). If a block has no missing values then its component in whereNA is integer(0).
- pseudo_maxima: a numeric matrix containing (pseudo) block maxima created by applying the missing value patterns from incomplete blocks to all full blocks, that is, blocks without any missing values. Each column contains the pseudo-maxima resulting from a particular incomplete block. The columns are labelled by the number of the incomplete block and the columns by the number of the full block. If an incomplete block contains all missing values then its entry in pseudo_maxima is NA. If there are no full blocks or no incomplete blocks then pseudo_maxima is NA.
- full_maxima: a numeric vector of maxima from full blocks.
- partial_maxima: a numeric vector of maxima from partial blocks.

If a block contains only missing values then its value of maxima is NA, its value of notNA is 0 and whereNA contains the positions of all the observations in the block.

If block is supplied then these vectors are named using the values in block. Otherwise, these vectors do not have names.

Examples

```
## Simulate example data
set.seed(7032025)
data <- rexp(15)

# Create some missing values
data[c(5, 7:8)] <- NA
# 5 blocks (columns), each with 3 observations
matrix(data, ncol = 5)

block_length <- 3
x <- sliding_block_maxima_ts(data, block_length = block_length)
```

Index