Package 'Rbeast'

May 7, 2026

Type Package

Version 1.0.2

Date 2025-12-09

Title Bayesian Change-Point Detection and Time Series Decomposition

Author Tongxi Hu [aut],
Yang Li [aut],
Xuesong Zhang [aut],
Kaiguang Zhao [aut, cre],
Jack Dongarra [ctb],
Cleve Moler [ctb]

Maintainer Kaiguang Zhao <zhao.1423@osu.edu>

Depends R (>= 2.10.0),methods, utils

Description
BEAST is a Bayesian estimator of abrupt change, seasonality, and trend for decomposing univariate time series and 1D sequential data. Interpretation of time series depends on model choice; different models can yield contrasting or contradicting estimates of patterns, trends, and mechanisms. BEAST alleviates this by abandoning the single-best-model paradigm and instead using Bayesian model averaging over many competing decompositions. It detects and characterizes abrupt changes (changepoints, breakpoints, structural breaks, joinpoints), cyclic or seasonal variation, and nonlinear trends. BEAST not only detects when changes occur but also quantifies how likely the changes are true. It estimates not just piecewise linear trends but also arbitrary nonlinear trends. BEAST is generically applicable to any real-valued time series, such as those from remote sensing, economics, climate science, ecology, hydrology, and other environmental and biological systems. Example applications include identifying regime shifts in ecological data, mapping forest disturbance and land degradation from satellite image time series, detecting market trends in economic indicators, pinpointing anomalies and extreme events in climate records, and analyzing system dynamics in biological time series. Details are given in Zhao et al. (2019) <doi:10.1016/j.rse.2019.04.034>.

LazyData true

Imports grid

License GPL (>= 2)

URL https://github.com/zhaokg/Rbeast

NeedsCompilation yes

Repository CRAN

Date/Publication 2025-12-19 09:00:02 UTC

Contents

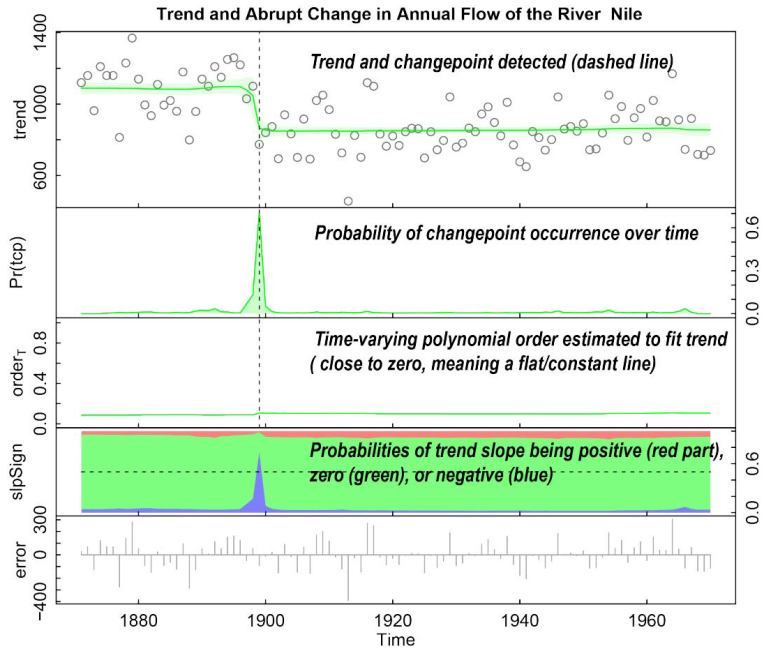
beast 2
beast.irreg 20
beast123 32
CNAchrom11 51
covid19 52
geeLandsat 53
googletrend_beach 55
imagestack 56
minesweeper 57
ohio 59
plot.beast 60
print.beast 62
simdata 63
tetris 64
tsextract 66
Yellowstone 68

Index **70**

`beast` *Bayesian changepoint detection and time series decomposition into trend, seasonality, and abrupt changes*

Description

`beast` is a high-level interface to the BEAST algorithm, a Bayesian model averaging approach for decomposing regular time series or other 1D sequential data into individual components, including abrupt changes, trends, and periodic/seasonal variations. BEAST is useful for changepoint detection (e.g., breakpoints, joinpoints, or structural breaks), nonlinear trend analysis, time series decomposition, and time series segmentation.



Usage

```
beast(
  y,
  start          = 1,
  deltat         = 1,
  season         = c("harmonic", "svd", "dummy", "none"),
  period         = NULL,
  scp.minmax     = c(0, 10), sorder.minmax   = c(0, 5),
  tcp.minmax     = c(0, 10), torder.minmax   = c(0, 1),
  sseg.min       = NULL,     sseg.leftmargin = NULL, sseg.rightmargin = NULL,
  tseg.min       = NULL,     tseg.leftmargin = NULL, tseg.rightmargin = NULL,
  s.complexfct   = 0.0,
  t.complexfct   = 0.0,
  method         = c("bayes", "bic", "aic", "aicc", "hic",
                     "bic0.25", "bic0.5", "bic1.5", "bic2"),
  detrend        = FALSE,
  deseasonalize  = FALSE,
  mcmc.seed      = 0,
  mcmc.burnin    = 200,
  mcmc.chains    = 3,
  mcmc.thin      = 5,
  mcmc.samples   = 8000,
  precValue      = 1.5,
  precPriorType  = c("componentwise", "uniform", "constant", "orderwise"),
  hasOutlier     = FALSE,
  ocp.minmax     = c(0, 10),
```

```
  print.param    = TRUE,
  print.progress = TRUE,
  print.warning  = TRUE,
  quiet          = FALSE,
  dump.ci        = FALSE,
  dump.mcmc      = FALSE,
  gui            = FALSE,
  ...
)
```

Arguments

y	y is a numeric vector representing an evenly spaced regular time series. Missing values such as NA and NaN are allowed. • If y is irregular or unordered in time (e.g., daily data spanning multiple years with leap years: 365 points in some years, 366 in others), use beast.irreg instead. • If y is a matrix or 3D array consisting of multiple regular or irregular time series (e.g., stacked images), use beast123 instead. • If y is an object of class "ts", "xts", or "zoo", its time attributes (i.e., start, end, frequency) are used to infer start, deltat, period, and season. User-specified values for these arguments are then ignored to honor the time attributes of y. For example, if y is a ts object with frequency = 1, season = "none" is always assumed; if frequency > 1 (i.e., some periodic component is present in y) but season = "none" is supplied, it is silently replaced by season="harmonic". If y is provided as a *list* of multiple time series, the multivariate version of the BEAST algorithm is invoked to decompose multiple series and detect common changepoints jointly. This feature is *experimental* and under active development. See ohio for a working example.
start	start is numeric (default 1.0) or Date; the time of the first data point of y. It can be specified as: • a scalar (e.g., 2021.0644); • a numeric vector c(year, month, day) (e.g., c(2021, 1, 24)); • an R Date object (e.g., as.Date("2021-01-24")).
deltat	deltat is numeric (default 1.0) or character string; the time interval between consecutive data points. Its unit must be consistent with start. • If start is numeric, the unit is arbitrary and is not interpreted by BEAST (e.g., 2021.3 can be be of any unit, a time, a distance, etc.). • If start is a c(year, month, day) vector or a Date, deltat is interpreted in units of years. For example, for a monthly series with start = c(2021, 1, 24), start is internally converted to a fractional year 2021 + (24 - 0.5)/365 = 2021.0644 and deltat = 1/12 can be used to specify a 1-month interval (1/12 yr). • Alternatively, deltat can be a character string encoding both the value and the unit, such as "7 days", "7d", "1/2 months", "1 mn", "1 year", or "1y".

season — Character (default "harmonic"). Specifies whether y has a periodic component and how it is modeled:

- "none": y is treated as trend-only; no periodic components are modeled. Seasonal parameters (e.g., sorder.minmax, scp.minmax, sseg.min) are ignored.
- "harmonic": y has a periodic/seasonal component. The term season is used broadly for any periodic variation in y. The period of the seasonal component is not a BEAST model parameter; it is treated as known and must be supplied via period. For "harmonic", the seasonal component is modeled as a harmonic curve (a combination of sines and cosines).
- "dummy": As for "harmonic", except that the periodic/seasonal component is modeled as a non-parametric curve. The harmonic order argument sorder.minmax is then irrelevant and ignored.
- "svd" (experimental): As for "harmonic", but with the seasonal component expressed as a linear combination of basis functions derived from a singular value decomposition (SVD). These basis functions can be more parsimonious than pure harmonic bases and may improve detection of subtle seasonal changepoints.

period — Numeric or character string. The period of the seasonal/periodic component in y. It is needed only when a periodic component is present (e.g., season = "harmonic", "svd", or "dummy") and is ignored for trend-only data (season = "none"). The period must be in the same time unit as deltat, and it should satisfy period = deltat * freq, where freq is the number of data points per period.

- In earlier versions, a separate freq argument was used; it is now obsolete and replaced by period. For backward compatibility, freq is still accepted via ..., but period takes precedence if both are provided.
- If period is missing, BEAST attempts to guess a plausible period via autocorrelation. This guess can be unreliable, so users are strongly encouraged to specify period explicitly whenever a periodic component is present.
- If period <= 0, season = "none" is assumed and a trend-only model is fitted.
- When start or deltat is specified using dates, period may also be provided as a string, such as "1 year", "12 months", "365d", or "366 days".

scp.minmax — A numeric vector of length 2 (integers >= 0); the minimum and maximum numbers of seasonal changepoints (scp) allowed in segmenting the seasonal component. Used only when y has a seasonal component (i.e., season = "harmonic", "svd", or "dummy") and ignored for trend-only data. scp.minmax[2] couldn't be smaller than scp.minmax[1].

- If scp.minmax[1] == scp.minmax[2], BEAST assumes a fixed number of seasonal changepoints and does not infer the posterior distribution over the number of changepoints, though it still estimates the probabilities and locations of most likely changepoints over time.
- If scp.minmax[1] == scp.minmax[2] == 0, both the min and max numbers of scp are set to 0. That is, no seasonal changepoints are allowed; a single global seasonal model is used, but the harmonic order may still be inferred if sorder.minmax[1] != sorder.minmax[2].

`sorder.minmax` A numeric vector of length 2 (integers >= 1); the minimum and maximum harmonic orders considered for the seasonal component. Used only when `season = "harmonic"` or `"svd"` and ignored for trend-only data or when `season = "dummy"`.

- If `sorder.minmax[1] == sorder.minmax[2]`, BEAST assumes a fixed harmonic order and does not infer the posterior distribution over harmonic orders.

`tcp.minmax` A numeric vector of length 2 (integers >= 0); the minimum and maximum numbers of trend changepoints (tcp) allowed in segmenting the trend component.

- If `tcp.minmax[1] == tcp.minmax[2]`, BEAST assumes a fixed number of trend changepoints and does not infer the posterior distribution over the number of changepoints, though it still estimates changepoint probabilities and locations.
- If `tcp.minmax[1] == tcp.minmax[2] == 0`, both the min and max numbers are set to 0. That is, no trend changepoints are allowed, and a single global polynomial trend is fitted. The polynomial order may still be inferred if `torder.minmax[1] != torder.minmax[2]`.

`torder.minmax` A numeric vector of length 2 (integers >= 0); the minimum and maximum polynomial orders considered for the trend component. If missing, `tcp.minmax` defaults to `c(0,1)`. Order 0 corresponds to a constant (flat) trend; order 1 corresponds to a linear trend.

- If `torder.minmax[1] == torder.minmax[2]`, BEAST assumes a fixed polynomial order and does not infer the posterior distribution over polynomial orders.

`sseg.min` An integer (> 0); the minimum segment length, in number of time steps, allowed between neighboring seasonal changepoints. When fitting a piecewise seasonal model, two seasonal changepoints cannot be closer than `sseg.min` time intervals. `sseg.min` is unitless (counts of time steps); the corresponding time window in the underlying units is `sseg.min * deltat`. If `NULL`, a default is chosen internally (typically based on the the period).

`sseg.leftmargin`
Integer (>= 0); the number of leftmost observations excluded from seasonal changepoint detection. No seasonal changepoints are allowed in the initial window of length `sseg.leftmargin`. This is specified in time steps; the corresponding time window is `sseg.leftmargin * deltat`. If `NULL`, it defaults to `sseg.min`.

`sseg.rightmargin`
Integer (>= 0); the number of rightmost observations excluded from seasonal changepoint detection. No seasonal changepoints are allowed in the final window of length `sseg.rightmargin`. This is specified in time steps; the corresponding time window is `sseg.rightmargin * deltat`. If `NULL`, it defaults to `sseg.min`.

`tseg.min` Integer (> 0); the minimum segment length, in number of time steps, allowed between neighboring trend changepoints. When fitting a piecewise polynomial trend, two trend changepoints cannot be closer than `tseg.min` time intervals. `tseg.min` is unitless (counts of time steps); the corresponding time window is `tseg.min * deltat`. If `NULL`, a default is chosen internally (often in reference to the presence or absence of a cyclic component).

`tseg.leftmargin`

Integer (`>= 0`); the number of leftmost observations excluded from trend changepoint detection. No trend changepoints are allowed in the initial window of length `tseg.leftmargin`. This is specified in time steps; the corresponding time window is `tseg.leftmargin * deltat`. If `NULL`, it defaults to `tseg.min`.

`tseg.rightmargin`

Integer (`>= 0`); the number of rightmost observations excluded from trend changepoint detection. No trend changepoints are allowed in the final window of length `tseg.rightmargin`. This is specified in time steps; the corresponding time window is `tseg.rightmargin * deltat`. If `NULL`, it defaults to `tseg.min`.

`s.complexfct` Numeric (default `0.0`); a hyperprior parameter–newly added in Version 1.02–controlling the complexity of the seasonal curve (i.e., the number of seasonal changepoints). A prior of the form $p(k) \propto \exp[\lambda(k+1)]$ is placed on the number of seasonal changepoints k, where λ is `seasonComplexityFactor`. Setting $\lambda = 0$ yields a non-informative prior $p(k) \propto 1.0$ where all model dimensions are equally likely *a priori*. Users may tune `seasonComplexityFactor` in the range `[-20, 20]` or an even wider range: negative values (e.g., $\lambda = -15.9$) favor fewer changepoints (simpler seasonal curves), whereas positive values (e.g., $\lambda = 5.76$) favor more changepoints (more complex curves).

`t.complexfct` Numeric (default `0.0`); analogous to `s.complexfct`, but for the trend component and the number of trend changepoints.

`method` Character (default `"bayes"`). Specifies how model posterior probabilities are formulated or approximated:

- `"bayes"`: Full Bayesian formulation as described in Zhao et al. (2019).
- `"bic"`: Approximate posterior probabilities using the Bayesian Information Criterion (BIC), $\mathrm{BIC} = n\log(\mathrm{SSE}) + k\log(n)$, where k is the number of parameters and n the number of observations.
- `"aic"`: Approximate posterior probabilities using Akaike's Information Criterion (AIC), $\mathrm{AIC} = n\log(\mathrm{SSE}) + 2k$.
- `"aicc"`: Approximate posterior probabilities using the corrected AIC (AICc), $\mathrm{AICc} = \mathrm{AIC} + \frac{2k^2+2k}{n-k-1}$.
- `"hic"`: Approximate posterior probabilities using the Hannan-Quinn Information Criterion (HIC), $\mathrm{HIC} = n\log(\mathrm{SSE}) + 2k\log(\log(n))$.
- `"bic0.25"`: BIC-type approximation adopted from Kim et al. (2016) doi:10.1016/j.jspi.2015.09.008 with reduced complexity penalty, $\mathrm{BIC}_{0.25} = n\log(\mathrm{SSE}) + 0.25k\log(n)$.
- `"bic0.5"`: As above but with penalty factor `0.5`.
- `"bic1.5"`: As above but with penalty factor `1.5`.
- `"bic2"`: As above but with penalty factor `2.0`.

`detrend` Logical; if `TRUE`, a global trend is first fitted and removed from the series before running BEAST, and then added back to the BEAST result at the end.

`deseasonalize` Logical; if `TRUE`, a global seasonal model is first fitted and removed before running BEAST, and then added back to the BEAST result. Ignored if `season = "none"` (trend-only data).

`mcmc.seed` Integer (>= 0); seed for the random number generator used in the MCMC. If `mcmc.seed = 0`, an arbitrary seed is used and results may vary across runs. Using the same non-zero seed makes results reproducible on the same machine, though differences across platforms/CPUs may still arise due to differences in random number libraries or CPU instruction sets.

`mcmc.chains` Integer (> 0); the number of parallel MCMC chains.

`mcmc.thin` Integer (> 0); thinning factor. If `mcmc.thin = k`, every k-th iteration is retained and the others are discarded.

`mcmc.burnin` Integer (> 0); number of initial iterations discarded as burn-in for each chain.

`mcmc.samples` Integer (>= 0); number of samples collected per MCMC chain. The total number of iterations is `(mcmc.burnin + mcmc.samples * mcmc.thin) * mcmc.chains`.

`precValue` Numeric (> 0); hyperparameter for the precision priors on model coefficients. The default is `1.5`. It is used directly only when `precPriorType = "constant"` (see below); in other cases it serves as an initial value.

`precPriorType` Character; one of `"constant"`, `"uniform"`, `"componentwise"` (default), or `"orderwise"`. These control how precision parameters for the model coefficients are treated:

1. `"constant"`: A single precision parameter is fixed at `precValue`. The fit may be sensitive to the chosen value.
2. `"uniform"`: A single precision parameter is treated as a random variable with initial value `precValue`; its posterior is inferred via MCMC. The results are less sensitive to the initial choice of `precValue`.
3. `"componentwise"`: Separate precision parameters are used for different components (e.g., season vs. trend), each initialized by `precValue` and inferred via MCMC.
4. `"orderwise"`: Separate precision parameters are used for different components and for different orders within each component, all initialized by `precValue` and inferred via MCMC.

`hasOutlier` Logical; if `TRUE`, fit a model with an outlier component representing potential spikes or dips at isolated data points:

- `Y = trend + outlier + error` if `season = "none"`,
- `Y = trend + season + outlier + error` otherwise.

`ocp.minmax` Numeric vector of length 2 (integers >= 0); the minimum and maximum numbers of outlier-type changepoints (ocp) allowed in the series. Outliers refer to spikes or dips at isolated times that cannot be captured by the trend or seasonal components.

`print.param` Logical; if `TRUE`, the full list of internal BEAST parameters (`metadata`, `prior`, `mcmc`, `extra`) is printed before MCMC begins. These internal objects correspond one-to-one with the arguments of `beast`. For example, `prior$trendMinSepDist` corresponds to `tseg.min`. See `beast123` or inspect the source of `beast` for details.

`print.progress` Logical; if `TRUE`, a progress bar is printed.

`print.warning` Logical; if `TRUE`, warning messages are printed.

`quiet` Logical; if `TRUE`, suppress most console output (overrides `print.param` and `print.progress`).

dump.ci — Logical; if TRUE, credible intervals for the trend and seasonal components (e.g., out$trend$CI, out$season$CI) are computed and returned. Computing credible intervals is relatively time-consuming (due to sorting and summarizing many MCMC samples). If only symmetric uncertainty summaries are needed, the posterior standard deviations out$trend$SD and out$season$SD may suffice and dump.ci can be set to FALSE.

dump.mcmc — Logical; if TRUE, individual MCMC samples are dumped to the output.

gui — Logical; if TRUE, BEAST runs with a GUI window showing an animation of the MCMC sampling in model space (numbers and locations of trend/seasonal changepoints). This feature is *experimental* and currently available only on 64-bit Windows systems (not 32-bit Windows, macOS, or Linux).

... — Additional arguments reserved for future extensions and backward compatibility. Certain low-level settings are only available via beast123() (through metadata, prior, mcmc, and extra), not via beast().

Value

The result is an object of class "beast", i.e., a list with structure identical to the outputs of beast.irreg and beast123. The exact array dimensions depend on the input y. Below we assume a single time series input of length N.

time — Numeric vector of length N; the time associated with each observation. By default, this is simply 1:N if start, deltat, and any time attributes are missing.

data — A vector, matrix, or 3D array; a copy of the input y if the underlying extra$dumpInputData = TRUE. If extra$dumpInputData = FALSE, this is NULL. For irregular inputs (as in beast.irreg), this field stores the aggregated regular series at time intervals spaced by deltat (or metadata$deltaTime in the beast123 interface).

marg_lik — Numeric; the average marginal likelihood of the sampled models. Larger values indicate better fit for a given time series (e.g., -1 is better than -10; 10 is better than -1 or -10).

R2 — Numeric; coefficient of determination (R-squared) of the fitted model.

RMSE — Numeric; root mean squared error of the fitted model.

sig2 — Numeric; estimated variance of the residual error.

trend — trend is a list of outputs related to the estimated trend component:

- ncp: Mean number of trend changepoints. Since individual sampled models can have different numbers of changepoints, several alternative summaries (ncp_mode, ncp_median, ncp_pct90) are also provided. For example, if mcmc$samples = 10 and the numbers of changepoints across models are c(0, 2, 4, 1, 1, 2, 7, 6, 6, 1), then the mean ncp is 3.1, the median 2.5, the mode 1, and the 90th percentile (ncp_pct90) 6.5.
- ncp_mode: Mode of the number of trend changepoints.
- ncp_median: Median of the number of trend changepoints.
- ncp_pct90: 90th percentile of the number of trend changepoints.
- ncpPr: Numeric vector of length tcp.minmax[2] + 1; probability distribution over the number of trend changepoints in 0:tcp.minmax[2]. For example, ncpPr[1] is the probability of having no trend changepoint; ncpPr[i] is the probability of having i - 1 changepoints.

- cpOccPr: Numeric vector of length N; changepoint occurrence probability at each time index for the trend component. Plotting cpOccPr yields a continuous "probability-of-being-a-changepoint" curve. A higher peak at a single time step does *not* necessarily imply a higher *overall* probability of a changepoint in a neighborhood compared to a broader peak with lower maximum but higher summed probability. For example, a window of cpOccPr values c(0, 0, 0.5, 0, 0) (peak 0.5, sum 0.5) likely indicates a smaller chance of a changepoint than another window c(0.1, 0.2, 0.21, 0.2, 0.1) (peak 0.21, sum 0.71).
- order: Numeric vector of length N; average polynomial order of the trend over sampled models at each time step. As a posterior average, it need not be an integer.
- cp: Numeric vector of length up to tcp.minmax[2]; estimated trend changepoint locations. These are obtained by smoothing cpOccPr with a window of size tseg.min and selecting up to tcp.minmax[2] peaks from the smoothed curve. Some entries may be NaN if fewer changepoints are identified. Not all reported changepoints should be treated as "true"; some may be false positives. Check the associated posterior probabilities (cpPr) for confidence.
- cpPr: Numeric vector of length tcp.minmax[2]; posterior probability associated with each changepoint in cp. Trailing entries are NaN if fewer changepoints are identified.
- cpCI: Numeric matrix of dimension tcp.minmax[2] x 2; credible intervals for changepoint locations in cp.
- cpAbruptChange: Numeric vector of length tcp.minmax[2]; magnitude of jumps in the trend at the detected changepoints.
- Y: Numeric vector of length N; estimated trend component (Bayesian model average over sampled trends).
- SD: Numeric vector of length N; posterior standard deviation of the trend.
- CI: Numeric matrix of dimension N x 2; credible interval for the trend, with lower and upper envelopes.
- slp: Numeric vector of length N; time-varying slope of the trend component.
- slpSD: Numeric vector of length N; posterior standard deviation of the slope.
- slpSgnPosPr: Numeric vector of length N; posterior probability that the slope is positive at each time point (i.e., increasing trend). For example, slpSgnPosPr = 0.80 at a given time indicates that 80% of sampled trends have a positive slope there.
- slpSgnZeroPr: Numeric vector of length N; posterior probability that the slope is effectively zero at each time point. The probability of negative slope can be obtained as 1 - slpSgnZeroPr - slpSgnPosPr.
- pos_ncp, neg_ncp, pos_ncpPr, neg_ncpPr, pos_cpOccPr, neg_cpOccPr, pos_cp, neg_cp, pos_cpPr, neg_cpPr, pos_cpAbruptChange, neg_cpAbruptChange, pos_cpCI, neg_cpCI: As above, but restricted to trend changepoints with positive jumps (pos_*) or negative jumps (neg_*) in the trend. For exam-

ple, pos_ncp is the mean number of trend changepoints where the trend level jumps up.

- inc_ncp, dec_ncp, inc_ncpPr, dec_ncpPr, inc_cpOccPr, dec_cpOccPr, inc_cp, dec_cp, inc_cpPr, dec_cpPr, inc_cpAbruptChange, dec_cpAbruptChange, inc_cpCI, dec_cpCI: As above, but restricted to changepoints where the trend slope increases (inc_*) or decreases (dec_*). For example, if the trend slope changes from 0.4 to 2.5, that changepoint contributes to inc_ncp.

season — season is a list analogous to trend, but for the seasonal/periodic component (when present):

- ncp: Mean number of seasonal changepoints.
- ncp_mode, ncp_median, ncp_pct90: As for trend, but for seasonal changepoints.
- ncpPr: Numeric vector of length scp.minmax[2] + 1; probability distribution over the number of seasonal changepoints in 0:scp.minmax[2]. For example, ncpPr[1] is the probability of having no seasonal changepoint.
- cpOccPr: Numeric vector of length N; seasonal changepoint occurrence probability over time. The same caveat applies as for trend$cpOccPr: the height of a single peak does not fully determine the overall probability of a changepoint in its neighborhood.
- order: Numeric vector of length N; average harmonic order needed to approximate the seasonal component. As a posterior average over piecewise harmonic/SVD-based curves, this need not be an integer.
- cp, cpPr, cpCI, cpAbruptChange: Analogous to the trend fields, but for the seasonal component, with length determined by scp.minmax[2].
- Y: Numeric vector of length N; estimated seasonal component (Bayesian model average).
- SD: Numeric vector of length N; posterior standard deviation of the seasonal component.
- CI: Numeric matrix of dimension N x 2; credible interval for the seasonal component.
- amp: Numeric vector of length N; time-varying amplitude of the seasonal component.
- ampSD: Numeric vector of length N; posterior standard deviation of the amplitude.
- pos_ncp, neg_ncp, pos_ncpPr, neg_ncpPr, pos_cpOccPr, neg_cpOccPr, pos_cp, neg_cp, pos_cpPr, neg_cpPr, pos_cpAbruptChange, neg_cpAbruptChange, pos_cpCI, neg_cpCI: Seasonal analogues of the trend outputs with the same names, restricted to seasonal changepoints with positive (pos_*) or negative (neg_*) jumps in the seasonal curve. For example, pos_ncp refers to the average number of seasonal changepoints at which the seasonal component jumps up.

outlier — outlier is a list analogous to trend or season, but for the outlier component (present only if setting hasOutlier=TRUE)

Note

The three functions `beast()`, `beast.irreg()`, and `beast123()` implement the same BEAST algorithm but with different APIs. There is a one-to-one correspondence between (1) the arguments of `beast()` / `beast.irreg()` and (2) the `metadata`, `prior`, `mcmc`, and `extra` lists used by `beast123()`. Examples include:

```
start               <->  metadata$startTime
deltat              <->  metadata$deltaTime
deseasonalize       <->  metadata$deseasonalize
hasOutlier          <->  metadata$hasOutlier
scp.minmax[1:2]     <->  prior$seasonMinKnotNum, prior$seasonMaxKnotNum
sorder.minmax[1:2]  <->  prior$seasonMinOrder, prior$seasonMaxOrder
sseg.min            <->  prior$seasonMinSepDist
tcp.minmax[1:2]     <->  prior$trendMinKnotNum, prior$trendMaxKnotNum
torder.minmax[1:2]  <->  prior$trendMinOrder, prior$trendMaxOrder
tseg.leftmargin     <->  prior$trendLeftMargin
tseg.rightmargin    <->  prior$trendRightMargin
s.complexfct        <->  prior$seasonComplexityFactor
t.complexfct        <->  prior$trendComplexityFactor
mcmc.seed           <->  mcmc$seed
dump.ci             <->  extra$computeCredible.
```

For large data sets, irregular time series, or stacked images, `beast123()` is generally the recommended interface.

References

1. Zhao, K., Wulder, M.A., Hu, T., Bright, R., Wu, Q., Qin, H., Li, Y., Toman, E., Mallick, B., Zhang, X. and Brown, M., 2019. Detecting change-point, trend, and seasonality in satellite time series data to track abrupt changes and nonlinear dynamics: A Bayesian ensemble algorithm. Remote Sensing of Environment, 232, p.111181 (the beast algorithm paper).
2. Zhao, K., Valle, D., Popescu, S., Zhang, X. and Mallick, B., 2013. Hyperspectral remote sensing of plant biochemistry using Bayesian model averaging with variable and band selection. Remote Sensing of Environment, 132, pp.102-119 (the Bayesian MCMC scheme used in beast).
3. Hu, T., Toman, E.M., Chen, G., Shao, G., Zhou, Y., Li, Y., Zhao, K. and Feng, Y., 2021. Mapping fine-scale human disturbances in a working landscape with Landsat time series on Google Earth Engine. ISPRS Journal of Photogrammetry and Remote Sensing, 176, pp.250-261(a beast application paper).

See Also

`beast.irreg`, `beast123`, `minesweeper`, `tetris`, `geeLandsat`

Examples

```
library(Rbeast)



#-----------------------------------Example 1--------------------------------------#
# 'googletrend_beach' is the monthly Google Trends popularity of searching for 'beach'
# in the US from 2004 to 2022. Sudden changes coincide with known extreme weather
# events (e.g., 2006 North American Blizzard, 2011 record US summer heat, record warm
# January in 2016) or the COVID-19 outbreak.

out <- beast(googletrend_beach)

plot(out)
plot(out, vars = c("t", "slpsgn"))  # plot trend and slope-sign probabilities only.
                                    # In the slpsgn panel, the upper red band is the
                                    # probability that the trend slope is positive,
                                    # the middle green band the probability that the
                                    # slope is effectively zero, and the lower blue
                                    # band the probability that it is negative.
                                    # See ?plot.beast for details.

#-----------------------------------Example 2--------------------------------------#
# Yellowstone is a half-monthly satellite NDVI time series of length 774 starting
# from July 1-15, 1981 (start ~ c(1981, 7, 7)) at a forest site in Yellowstone.
# There are 24 data points per year. The 1988 Yellowstone Fire caused a sudden drop
# in greenness. Note that the beast function  hanldes only evenly-spaced regular time
# series. Irregular data need to be first  aggegrated at a regular time interval of
# your choice--the aggregation functionality is implemented in beast.irreg() and beast123().

data(Yellowstone)
plot(1981.5 + (0:773) / 24, Yellowstone, type = "l") # A sudden drop in greenness in 1988
                                                     # due to the 1988 Yellowstone Fire

# Yellowstone is a plain numeric vector (no time attributes). Below, no extra
# arguments are supplied, so default values (start = 1, deltat = 1) are used and
# time is 1:774. The period is missing and is guessed via autocorrelation.
# Autocorrelation-based period estimates can be unreliable; examples below show
# how to specify time/period explicitly.

o <- beast(Yellowstone)   # By default, time = 1:length(Yellowstone), season = "harmonic"
plot(o)

# o <- beast(Yellowstone, quiet = TRUE)                       # suppress warnings
# o <- beast(Yellowstone, quiet = TRUE, print.progress = FALSE) # suppress all output

#-----------------------------------Example 3--------------------------------------#
# Time information (start, deltat, period) specified explicitly of Yellowstone.

# (1) Arbitrary unit: 1981.5137 can be interpreted flexibly in any units not neccessarily
# referting to times
o <- beast(Yellowstone, start = 1981.5137, deltat = 1/24, period = 1.0)
```

```
# Strings can be used to explicitly specify time unitsas dates or years:
# o <- beast(Yellowstone, start = 1981.5137, deltat = "1/24 year", period = 1.0)
# o <- beast(Yellowstone, start = 1981.5137, deltat = "0.5 mon",   period = 1.0)
# o <- beast(Yellowstone, start = 1981.5137, deltat = 1/24, period = "1 year")
# o <- beast(Yellowstone, start = 1981.5137, deltat = 1/24, period = "365 days")

# (2) start as year-month-day (unit is year: deltat = 1/24 year = 0.5 month)
# o <- beast(Yellowstone, start = c(1981, 7, 7), deltat = 1/24, period = 1.0)

# (3) start as Date (unit is year: deltat = 1/24 year = 0.5 month)
# o <- beast(Yellowstone, start = as.Date("1981-07-07"), deltat = 1/24, period = 1.0)

print(o)     # 'o' is a list with many fields
str(o)       # See a list of fields in the BEAST output o

plot(o)                            # plot many default variables
plot(o, vars = c("y","s","t"))     # plot the data, seasonal, and trend components only
plot(o, vars = c("s","scp","samp","t","tcp","tslp")) # selected variables (see ?plot.beast)
plot(o, vars = c("s","t"), col = c("red","blue"))    # Specify colors of selected subplots

plot(o$time, o$season$Y, type = "l")    # fitted seasonal component
plot(o$time, o$season$cpOccPr)          # seasonal changepoint probabilities
plot(o$time, o$trend$Y, type = "l")     # fitted trend component
plot(o$time, o$trend$cpOccPr)           # trend changepoint probabilities
plot(o$time, o$season$order)            # average harmonic order over time

plot(o, interactive = TRUE)            # interactively choose variables to plot




#-----------------------------------Example 4--------------------------------------#
# Specify other arguments explicitly.  Default values are used for missing parameters.
# Note that beast(), beast.irreg(), and beast123() call the same internal C/C++ library,
# so in beast(), the input parameters will be converted to metadata, prior, mcmc, and
# extra parameters as explained for the beast123() function. Or type 'View(beast)' to
# check the parameter assignment in the code.


 # In R's terminology, the  number of datapoints per period is also called 'freq'. In this
 # version, the 'freq' argument is obsolete and replaced by 'period'.

 # period=deltat*number_of_datapoints_per_period = 1.0*24=24 because deltat is set to 1.0 by
 # default and this signal has 24 samples per period.
 o = beast(Yellowstone, period=24.0, mcmc.samples=5000, tseg.min=20)

 # period=deltat*number_of_datapoints_per_period = 1/24*24=1.0.
 # o = beast(Yellowstone, deltat=1/24 period=1.0, mcmc.samples=5000, tseg.min=20)

 o = beast(
     Yellowstone,            # Yellowstone: a pure numeric vector wo time info
     start   = 1981.51,
```

```
    deltat  = 1/24,
    period  = 1.0,           # Period=delta*number_of_datapoints_per_period
    season  = 'harmonic',    # Periodic compnt exisits,fitted as a harmonic curve
    scp.minmax     = c(0,3), # Min and max numbers of seasonal changpts allowed
    sorder.minmax  = c(1,5), # Min and max harmonic orders allowed
    sseg.min       = 24,     # The min length of segments btw neighboring chnpts
                        # '24' means 24 datapoints; the unit is datapoint.
    sseg.leftmargin= 40,     # No seasonal chgpts allowed in the starting 40 datapoints
    tcp.minmax     = c(0,10),# Min and max numbers of changpts allowed in the trend
    torder.minmax  = c(0,1), # Min and maxx polynomial orders to fit trend
    tseg.min       = 24,     # The min length of segments btw neighboring trend chnpts
    tseg.leftmargin= 10,     # No trend chgpts allowed in the starting 10 datapoints
    s.complexfct   = 0.26,   # Manually tune it to fit a more complex seasonal curve
                        #  than the non-informative prior on number of changepts
    t.complexfct   = -5.2,   # Manually tune it to fit a more complex trend curve
                        #  than the non-informative prior on number of changepts
   deseasonalize  = TRUE,   # Remove the global seasonality before fitting the beast model
    detrend        = TRUE,   # Remove the global trend before fitting the beast model
    mcmc.seed      = 0,      # A seed for mcmc's random nummber generator; use a
                             # non-zero integer to reproduce results across runs
    mcmc.burnin    = 500,    # Number of initial iterations discarded
    mcmc.chains    = 2,      # Number of chains
    mcmc.thin      = 3,      # Include samples every 3 iterations
    mcmc.samples   = 6000,   # Number of samples taken per chain
                             # total iteration: (500+3*6000)*2
    print.param     = FALSE  # Do not print the parameters
    )
plot(o)
plot(o,vars=c('t','slpsgn') )         # plot only trend and slope sign
plot(o,vars=c('t','slpsgn'), relative.heights =c(.8,.2) ) # run "?plot.beast" for more info
plot(o, interactive=TRUE)



#----------------------------------Example 5--------------------------------------#
# Run an interactive GUI to visualize how BEAST is samplinig from the possible model
# spaces in terms of the numbers and timings of seasonal and trend changepoints.
# The GUI inferface allows changing the option parameters interactively. This GUI is
# only available on Win x64 machines, not Mac or Linux.

## Not run:
beast(Yellowstone, period=24, gui=TRUE)

## End(Not run)

#----------------------------------Example 6--------------------------------------#
# Apply beast to trend-only data. 'Nile' is the ANNUAL river flow of the river
# Nile at Aswan since 1871. It is a 'ts' object; its time attributes (start=1871,
# end=1970,frequency=1) are used to replace the user-supplied start,deltat, and freq,
# if any.


data(Nile)
```

```
plot(Nile)
attributes(Nile) # a ts object with time attributes (i.e., tsp=(start,end,freq)

o = beast(Nile)  # start=1871, delta=1, and freq=1 taken from Nile itself
plot(o)

o = beast(Nile,             # the same as above. The user-supplied values (i.e., 2023,
          start=2023,      # 9999) are ignored bcz Nile carries its own time attributes.
          period=9999,     # Its frequency tag is 1 (i.e., trend-only), so season='none'
           season='harmonic' # is used instead of the supplied 'harmonic'
  )


#----------------------------------Example 7--------------------------------------#
# NileVec is  a pure data vector. The first run below is WRONG bcz NileVec was assumed
# to have a perodic component by default and beast gets a best estimate of freq=6 while
# the true value is freq=1. To fit a trend-only model, season='none' has to be explicitly
# specified, as in the 2nd & 3rd funs.

NileVec = as.vector(Nile) # NileVec is not a ts obj but a pure numeric data vector
o       = beast(NileVec)  # WRONG WAY to call: No time attributes available to interpret
                        # NileVec. By default, beast assumes season='harmonic', start=1,
                          # & deltat=1. 'freq' is missing and guessed to be 6 (WRONG).

plot(o)                   # WRONG Results: The result has a suprious seasonal component

o=beast(NileVec,season='none') # The correct way to call: Use season='none' for trend-only
                               # analysis; the default time is the integer indices
                               # "1:length(NileVec)'.
print(o$time)

o=beast(NileVec,               # Recommended way to call: The true time attributes are
        start  = 1871,         # given explicitly through start and deltat (or freq if
        deltat = 1,            # there is a  cyclic/seasonal cmponent).
        season = 'none')
print(o$time)
plot(o)



#----------------------------------Example 8--------------------------------------#
# beast can handle missing data. co2 is a monthly time series (i.e.,freq=12) starting
# from Jan 1959. We generate some missing values at random indices

## Not run:

data(co2)
attributes(co2)                         # A ts object with time attributes (i.e., tsp)
badIdx      = sample( 1:length(co2), 50) # Get a set of random indices
co2[badIdx] = NA                        # Insert some data gaps

out=beast(co2) # co2 is a ts object and its 'tsp' time attributes are used to get the
               # true time info. No need to specify 'start','deltat', & freq explicity.
```

```
 out=beast(co2,                  # The supplied time/period values will be ignored bcz
           start  = c(1959,1,15),# co2 is a ts object; the correct period = 1 will be
           deltat = 1/12,        # used.
           period = 365)
 print(out)
 plot(out)

## End(Not run)



#-----------------------------------Example 9--------------------------------------#
# Apply beast to time seris-like sequence data: the unit of sequences is not
# necessarily time.


  data(CNAchrom11) # DNA copy number alterations in Chromesome 11 for cell line GM05296
                   # The data is orderd by genomic position (not time), and the values
                  # are the log2-based intensity ratio of copy numbers between the sample
                   # the reference. A value of zero means no gain or loss in copy number.
  o = beast(CNAchrom11,season='none') # season is a misnomer here bcz the data has nothing
                                      # to do with time. Regardless, we fit only a trend.
  plot(o)




#-----------------------------------Example 10-------------------------------------#
# Apply beast to time seris-like data: the unit of sequences is not necessarily time.


  # Age of Death of Successive Kings of England
  # If the data link is deprecated, install the time series data library instead,
  # which is available at https://pkg.yangzhuoranyang.com/tsdl/
  # install.packages("devtools")
  # devtools::install_github("FinYang/tsdl")
  # kings = tsdl::tsdl[[293]]

  kings = scan("http://robjhyndman.com/tsdldata/misc/kings.dat",skip=3)
  out   = beast(kings,season='none')
  plot(out)




#-----------------------------------Example 11-------------------------------------#
# Another example from the tsdl data library



  # Number of monthly births in New York from Jan 1946 to Dec 1959
  # If the data link becomes invalid, install the time series data package instead
```

```
  # install.packages("devtools")
  # devtools::install_github("FinYang/tsdl")
  # kings = tsdl::tsdl[[534]]

  births = scan("http://robjhyndman.com/tsdldata/data/nybirths.dat")
  out    = beast(births,start=c(1946,1,15), deltat=1/12 )
  plot(out) # the result is wrong bcz the guessed freq via auto-correlation by beast
            # is 2 rather than 12, so we recommend always specifying 'freq' explicitly
            # for those time series with a periodic component, as shown below.
  out    = beast(births,start=c(1946,1,15), deltat=1/12, freq  =12 )
  out    = beast(births,start=c(1946,1,15), deltat=1/12, period=1.0 )
  plot(out)

  # Fit the seasonal component using a singular-value-decompostion-based basis functions
  # rathat than the default harmonic sin/cos basis functions.
  out    = beast(births,start=c(1946,1,15), deltat=1/12, period=1.0, season="svd" )
  plot(out)



#-----------------------------------Example 12--------------------------------------#
#    Daily confirmed COVID-19 new cases and deaths across the globe

 ## Not run:
 data(covid19)
 plot(covid19$date, covid19$newcases, type='l')

 newcases = sqrt( covid19$newcases )  # Apply a square root-transformation

 # This ts varies periodically every 7 days. 7 days can't be precisely represented
 # in  the unit of year bcz some years has 365 days and others has 366. BEAST can hanlde
 # this in two ways.


 #(1) Use the date number as the time unit--the num of days lapsed since 1970-01-01.

  datenum  = as.numeric(covid19$date)
  o        = beast(newcases, start=min(datenum), deltat=1, period=7)
  o$time   = as.Date(o$time, origin='1970-01-01') # Convert from integers to Date.
  plot(o)

 #(2) Use strings to explicitly specify deltat and period with a unit.

  startdate = covid19$date[1]
  o         = beast(newcases, start=startdate, deltat='1day', period='7days')
  plot(o)


## End(Not run)

#-----------------------------------Example 13--------------------------------------#
# The old API interface of beast is still made available but NOT recommended. It is
# kept mainly to ensure the working of the sample code on Page 475 in the text
```

```
# Ecological Metods by Drs. Southwood and Henderson.

## Not run:

  # The interface as shown here will be deprecated and NOT recommended.
  beast(Yellowstone, 24)  #24 is the freq: number of datapoints per period


  # Specify the model or MCMC parameters through opt as in Rbeast v0.2
  opt=list()             #Create an empty list to append individual model parameters
  opt$period=24          #Period of the cyclic component (i.e.,freq in the new version)
  opt$minSeasonOrder=2   #Min harmonic order allowed in fitting season component
  opt$maxSeasonOrder=8   #Max harmonic order allowed in fititing season component
  opt$minTrendOrder=0    #Min polynomial order allowed to fit trend (0 for constant)
  opt$maxTrendOrder=1    #Max polynomial order allowed to fit trend (1 for linear term)
  opt$minSepDist_Season=20#Min separation time btw neighboring season changepoints
  opt$minSepDist_Trend=20 #Min separation time btw neighboring trend  changepoints
  opt$maxKnotNum_Season=4 #Max number of season changepoints allowed
  opt$maxKnotNum_Trend=10 #Max number of trend changepoints allowed
  opt$omittedValue=NA    #A customized value to indicate bad/missing values in the time
                          #series, in additon to those NA or NaN values.

  # The following parameters used to configure the reverisible-jump MCMC (RJMCC) sampler
  opt$chainNumber=2       #Number of parallel MCMC chains
  opt$sample=1000         #Number of samples to be collected per chain
  opt$thinningFactor=3    #A factor to thin chains
  opt$burnin=500          #Number of burn-in samples discarded at the start
  opt$maxMoveStepSize=30  #For the move proposal, the max window allowed in jumping from
                           #the current changepoint
  opt$resamplingSeasonOrderProb=0.2 #The probability of selecting a re-sampling proposal
                                    #(e.g., resample seasonal harmonic order)
  opt$resamplingTrendOrderProb=0.2  #The probability of selecting a re-sampling proposal
                                    #(e.g., resample trend polynomial order)

  opt$seed=65654    #A seed for the random generator: If seed=0,random numbers differ
                    #for different BEAST runs. Setting seed to a chosen non-zero integer
                    #will allow reproducing the same result for different BEAST runs.

  beast(Yellowstone, opt)

## End(Not run)

#-----------------------------------Example 14--------------------------------------#
# Fit a model with an outlier component: Y = trend + outlier + error.
# Outliers here refer to spikes or dips at isolated points that can't be capatured by the
# trend
## Not run:
 NileVec        = as.vector(Nile)
 NileVec[50]    = NileVec[50] + 1500              # Add an large artificial spike at t=50
 o = beast(NileVec, season='none', hasOutlier=TRUE)
 plot(o)

 o = beast(Nile, season='none', hasOutlier=TRUE)     # Fit a model with outliers
```

```
plot(o)

## End(Not run)
```

`beast.irreg` *Bayesian decomposition of irregular time series for changepoints, trend, and seasonality*

Description

`beast.irreg` applies the BEAST (Bayesian Estimator of Abrupt change, Seasonal change, and Trend) model to *irregular* or unordered time series or 1D sequential data. BEAST is a Bayesian model averaging algorithm for decomposing time series or other 1D sequential data into individual components, including abrupt changes, trends, and periodic/seasonal variations. It is useful for changepoint detection (e.g., breakpoints or structural breaks), nonlinear trend analysis, time series decomposition, and time series segmentation. `beast123` is a low-level interface to BEAST.

Internally, irregular observations are first aggregated into an evenly spaced (regular) time series at a user-specified time interval, and then decomposed into individual components such as abrupt changes, nonlinear trends, and periodic/seasonal variations.

Usage

```
beast.irreg(
     y,
     time,
     deltat         = NULL,
     season         = c("harmonic", "svd", "dummy", "none"),
     period         = NULL,
     scp.minmax     = c(0, 10),   sorder.minmax   = c(0, 5),
     tcp.minmax     = c(0, 10),   torder.minmax   = c(0, 1),
    sseg.min      = NULL,      sseg.leftmargin = NULL,  sseg.rightmargin = NULL,
    tseg.min      = NULL,      tseg.leftmargin = NULL,  tseg.rightmargin = NULL,
     s.complexfct   = 0.0,
     t.complexfct   = 0.0,
     method         = c("bayes", "bic", "aic", "aicc", "hic",
                    "bic0.25", "bic0.5", "bic1.5", "bic2"),
     detrend        = FALSE,
     deseasonalize  = FALSE,
     mcmc.seed      = 0,
     mcmc.burnin    = 200,
     mcmc.chains    = 3,
     mcmc.thin      = 5,
     mcmc.samples   = 8000,
     precValue      = 1.5,
     precPriorType  = c("componentwise", "uniform", "constant", "orderwise"),
     hasOutlier     = FALSE,
```

```
        ocp.minmax     = c(0, 10),
        print.param    = TRUE,
        print.progress = TRUE,
        print.warning  = TRUE,
        quiet          = FALSE,
        dump.ci        = FALSE,
        dump.mcmc      = FALSE,
        gui            = FALSE,
        ...
    )
```

Arguments

y	y is a numeric vector for an irregular or unordered time series. Missing values such as NA and NaN are allowed. • If y is evenly spaced in time (regular), use beast instead. • If y is a matrix or 3D array (e.g., a stack of images) consisting of multiple regular or irregular time series, use beast123 instead. If y is a *list* of multiple time series, the multivariate version of the BEAST algorithm is invoked to jointly decompose the series and detect common change-points. This feature is currently experimental and under development. See ohio for a working example.
time	time is a vector (or list) specifying the times of each observation in y. Its length must match the time dimension of y. It can be numeric, Date, character, or a list of date components. Supported formats include: 1. **Numeric vector** For example c(1984.23, 1984.27, 1984.36, ...). The unit of time is irrelevant to BEAST as long as it is used consistently with deltat and period. 2. **R** Date **vector** For example as.Date(c("1984-03-27", "1984-04-10", "1984-05-12", ...)). 3. **Character vector of date strings** For example: • c("1984-03-27", "1984-04-10", "1984-05-12") • c("1984/03/27", "1984,04,10", "1984 05 12") (delimiters may differ as long as the year-month-day order is consistent) • c("LT4-1984-03-27", "LT4-1984-04-10", "LT4-1984+05,12") • c("LT4-1984087ndvi", "LT4-1984101ndvi", "LT4-1984133ndvi") • c("1984,,abc 3/ 27", "1984,,ddxfdd 4/ 10", "ggd1984,, 5/ ttt 12") BEAST uses several heuristics to parse date strings without an explicit format specifier and may fail for ambiguous patterns (e.g., in "LC8-2020-09-20-1984" it is unclear whether 2020 or 1984 is the year). For robust parsing, consider using a list with an explicit format (see the next). 4. **List with date strings and format** A list of the form time = list(datestr = ..., strfmt = "..."), where

time$datestr is a character vector of date strings and time$strfmt is a format specifier describing how to extract year, month, day, or day-of-year. Three classes of formats are supported:

- (a) *Fixed positions for year/month/day.*
  For example, to extract 2001/12/02 etc. from time$datestr = c("P23R34-2001.1202333xd", "O93X94-2002.1108133fd", "TP3R34-2009.0122333td") use time$strfmt = "P23R34-yyyy.mmdd333xd", where yyyy, mm, and dd mark the positions of year, month, and day; all other characters are wildcards.
- (b) *Fixed positions for year and day-of-year (doy).*
  For example, to extract years and days-of-year from strings like "P23R342001888045" use time$strfmt = "123123yyyy888doy", where yyyy and doy mark year and day-of-year; all other positions are wildcards. doy must be three digits.
- (c) *Patterns based on separators between Y/M/D.*
  For example, to extract 2002/12/02 from "2002,12/02", " 2002 , 12/2", "2002,12 /02 " use time$strfmt = "Y,M/D"; whitespace is ignored. To extract 2002/12/02 from "2--12, 2012", use time$strfmt = "D--M,Y".

5. **List of date components**
   A list specifying dates component-wise, using either
   - time$year, time$month, and time$day, or
   - time$year and time$doy

   where each element is a vector of the same length as the time dimension of y. For the doy representation, values must lie between 1 and 365/366.

deltat

Numeric or character; the time interval used to *aggregate* the irregular time series into a regular one.

The BEAST model is formulated for regular (evenly spaced) data; therefore beast.irreg first re-bins the irregular time series into regular intervals of length deltat. If deltat is missing, a heuristic guess is used. The unit of deltat must be consistent with time:

- If time is numeric, the unit of deltat is arbitrary but must be consistent with time.
- If time is a Date vector or date strings, the default unit of deltat is fractional years.

To specify the unit explicitly, supply a character string, for example "7 days", "7d", "1/2 months", "1 mn", "1.0 year", or "1y".

season

Character (default "harmonic"); indicates whether y has a periodic component, and how it should be modeled. Allowed values are:

- "none": y is trend-only; no periodic component is modeled. Seasonal arguments (e.g., sorder.minmax, scp.minmax, sseg.min) are ignored.
- "harmonic": y has a periodic/seasonal component. Here, 'season' is used broadly to mean any periodic variation. The period is a known constant supplied via period and is *not* estimated as a model parameter. The seasonal component is modeled as a harmonic curve (sum of sines and cosines).
- "dummy": As in "harmonic", but the periodic component is modeled via non-parametric dummy bases. The harmonic-order argument sorder.minmax is irrelevant and ignored.

- "svd": (experimental) As in "harmonic", but the seasonal component is modeled as a linear combination of basis functions derived via singular value decomposition (SVD). These SVD-based bases can be more parsimonious than the standard harmonic bases and can improve sensitivity to subtle changepoints.

period — Numeric or character. The period of the seasonal/periodic component in y. Required only when a periodic component is present (i.e., season is "harmonic", "svd", or "dummy"), and ignored if season = "none".

The period must have units consistent with deltat, and satisfies period = deltat * freq, where freq is the number of samples per period. The historical freq argument (number of data points per period) is still supported via ..., but is deprecated; if both period and freq are supplied, period takes precedence.

period (or equivalently freq) is not itself a BEAST model parameter and must usually be specified by the user. If period is missing, BEAST attempts to guess it via autocorrelation before fitting the model.

If period <= 0, season = "none" is assumed and a trend-only model is fitted. To specify units explicitly for date-based time axes, use a string such as "1.0 year", "12 months", "365d", or "366 days".

scp.minmax — Integer vector of length 2 (>= 0); minimum and maximum numbers of *seasonal* changepoints (SCPs) allowed in the seasonal component. Used only when a seasonal component is present (i.e., season != "none") and ignored otherwise.

If scp.minmax[1] == scp.minmax[2], BEAST assumes a fixed number of seasonal changepoints and does not infer the posterior on the number of changepoints, but still estimates when those changepoints occur.

If both values are 0, no seasonal changepoints are allowed; a global harmonic (or SVD/dummy) model is used, but the most likely harmonic order is still inferred if sorder.minmax[1] != sorder.minmax[2].

sorder.minmax — Integer vector of length 2 (>= 1); minimum and maximum harmonic orders considered for the seasonal component. Used only if season = "harmonic" or "svd" and ignored for trend-only data or when season = "dummy".

If sorder.minmax[1] == sorder.minmax[2], BEAST assumes a fixed harmonic order and does not infer the posterior distribution of harmonic orders.

tcp.minmax — Integer vector of length 2 (>= 0); minimum and maximum numbers of *trend* changepoints (TCPs) allowed in the trend component.

If tcp.minmax[1] == tcp.minmax[2], BEAST assumes a fixed number of trend changepoints and does not infer the posterior on the number of trend changepoints, but still estimates their occurrence probabilities over time.

If both values are 0, no trend changepoints are allowed; a global polynomial trend is used, but the most likely polynomial order is still inferred if torder.minmax[1] != torder.minmax[2].

torder.minmax — Integer vector of length 2 (>= 0); minimum and maximum polynomial orders considered for the trend component. Order 0 corresponds to a constant (flat) trend; order 1 is a line.

If torder.minmax[1] == torder.minmax[2], BEAST assumes a fixed polynomial order and does not infer the posterior distribution of trend orders.

`sseg.min` Integer (> `0`); minimum length (in data points) of a segment between two neighboring seasonal changepoints. When fitting a piecewise seasonal model, no two seasonal changepoints are allowed to occur within `sseg.min` time steps.

`sseg.min` is unitless (number of time intervals); in the original time unit the window length is `sseg.min * deltat`. If `NULL`, a default based on the implied frequency is used.

`sseg.leftmargin`

Integer (>= `0`); number of leftmost data points excluded from seasonal changepoint detection. No seasonal changepoints are allowed in the initial window of length `sseg.leftmargin`.

`sseg.leftmargin` is unitless (number of samples). In the original time unit, the excluded window length is `sseg.leftmargin * deltat`. If `NULL`, defaults to `sseg.min`.

`sseg.rightmargin`

Integer (>= `0`); number of rightmost data points excluded from seasonal changepoint detection. No seasonal changepoints are allowed in the ending window of length `sseg.rightmargin`.

`sseg.rightmargin` is unitless, and the corresponding time window is `sseg.rightmargin * deltat`. If `NULL`, defaults to `sseg.min`.

`tseg.min` Integer (> `0`); minimum length (in data points) of a segment between two neighboring trend changepoints. When fitting a piecewise polynomial trend, no two trend changepoints are allowed within a window of length `tseg.min`.

`tseg.min` is unitless; in the original time unit the window is `tseg.min * deltat`. If `NULL`, a default based on the implied frequency is used when a cyclic component is present.

`tseg.leftmargin`

Integer (>= `0`); number of leftmost data points excluded from trend changepoint detection. No trend changepoints are allowed in the starting window of length `tseg.leftmargin`.

`tseg.leftmargin` is unitless; the excluded time window is `tseg.leftmargin * deltat`. If `NULL`, defaults to `tseg.min`.

`tseg.rightmargin`

Integer (>= `0`); number of rightmost data points excluded from trend changepoint detection. No trend changepoints are allowed in the ending window of length `tseg.rightmargin`.

`tseg.rightmargin` is unitless; the excluded time window is `tseg.rightmargin * deltat`. If `NULL`, defaults to `tseg.min`.

`s.complexfct` Numeric (default `0.0`). A hyperprior parameter–newly added in Version 1.02–controlling the complexity of the seasonal curve (i.e., the number of seasonal changepoints). A prior of the form $p(k) \propto \exp[\lambda(k+1)]$ is placed on the number of seasonal changepoints k, where λ is `seasonComplexityFactor`. Setting $\lambda = 0$ yields a non-informative prior $p(k) \propto 1.0$ where all model dimensions are equally likely *a priori*. Users may tune `seasonComplexityFactor` in the range `[-20, 20]` or an even wider range: negative values (e.g., $\lambda = -15.9$) favor fewer changepoints (simpler seasonal curves), whereas positive values (e.g., $\lambda = 5.76$) favor more changepoints (more complex curves).

`t.complexfct` Numeric scalar (default `0.0`); analogous to `s.complexfct` but for the trend component and the number of trend changepoints.

`method` Character (default `"bayes"`); specifies how model posterior probabilities are formulated or approximated:

- `"bayes"`: full Bayesian formulation as in Zhao et al. (2019).
- `"bic"`: BIC approximation, `bic = n * ln(SSE) + k * ln(n)`, where `k` and `n` are the numbers of parameters and data points.
- `"aic"`: AIC approximation, `aic = n * ln(SSE) + 2k`.
- `"aicc"`: corrected AIC, `aicc = aic + (2k^2 + 2k) / (n - k - 1)`.
- `"hic"`: Hannan–Quinn information criterion, `hic = n * ln(SSE) + 2k * ln(ln(n))`.
- `"bic0.25"`: BIC-type approximation adopted from Kim et al. (2016) <doi:10.1016/j.jspi.2015.09.00 `bic0.25 = n * ln(SSE) + 0.25 k * ln(n)`, with a weaker penalty on model complexity.
- `"bic0.50"`: same as above but with penalty factor 0.50.
- `"bic1.5"`: same as above but with penalty factor 1.5.
- `"bic2"`: same as above but with penalty factor 2.0.

`detrend` Logical; if `TRUE`, a global trend is first fitted and removed from the series before running BEAST, and then added back to the final result. This can help when the global trend dominates other components.

`deseasonalize` Logical; if `TRUE`, a global seasonal model is first fitted and removed from the series before running BEAST, then added back to the final result. Ignored when `season = "none"`.

`mcmc.seed` Integer (`>= 0`); seed for the random number generator used in the MCMC sampler. If `mcmc.seed = 0`, an arbitrary seed is chosen and results vary across runs. Using a fixed non-zero seed improves reproducibility on the same machine; results may still differ across architectures because the underlying random number generator can depend on CPU instruction sets.

`mcmc.chains` Integer (`> 0`); number of parallel MCMC chains.

`mcmc.thin` Integer (`> 0`); thinning factor for the chains (e.g., if `mcmc.thin = 5`, every 5th iteration is retained).

`mcmc.burnin` Integer (`> 0`); number of burn-in samples discarded at the beginning of each chain.

`mcmc.samples` Integer (`>= 0`); number of post-burn-in samples collected per MCMC chain. The total number of iterations is `(mcmc.burnin + mcmc.samples * mcmc.thin) * mcmc.chains`.

`precValue` Numeric (`> 0`); hyperparameter for the precision prior. It is directly used only when `precPriorType = "constant"`; otherwise it serves as an initial value that is updated within MCMC (see below).

`precPriorType` Character; one of `"constant"`, `"uniform"`, `"componentwise"` (default), or `"orderwise"`. These options control how precision parameters for model coefficients are treated:

1. `"constant"`: a single precision parameter is fixed to `precValue`. The result may be sensitive to `precValue`.

2. "uniform": a single precision parameter is treated as a random variable; its initial value is precValue, but its posterior is inferred via MCMC, making results less sensitive to the initial choice.
3. "componentwise": separate precision parameters are used for different components (e.g., one for season, another for trend), initialized by precValue and updated via MCMC.
4. "orderwise": separate precision parameters are used not only for components but also for individual orders within each component, again initialized by precValue and inferred via MCMC.

hasOutlier — Logical; if TRUE, fits a model with an additional outlier component capturing spikes or dips at isolated data points:

- Y = trend + outlier + error when season = "none",
- Y = trend + season + outlier + error otherwise.

Outliers are modeled as changepoints that cannot be captured by trend or seasonal terms.

ocp.minmax — Integer vector of length 2 (>= 0); minimum and maximum numbers of *outlier-type* changepoints (OCPs) allowed in the series. OCPs correspond to isolated spikes/dips that are not captured by trend or seasonality.

print.param — Logical; if TRUE, the full list of internal BEAST parameters (organized as metadata, prior, mcmc, and extra) is printed before MCMC. The internal naming differs slightly from the beast.irreg interface, but there is a one-to-one mapping (e.g., prior$trendMinSepDist <- tseg.min). See beast123 or View(beast) for details.

print.progress — Logical; if TRUE, prints a progress bar during model fitting.

print.warning — Logical; if TRUE, prints warning messages.

quiet — Logical; if TRUE, suppresses all console output (overrides other printing options).

dump.ci — Logical; if TRUE, computes explicit credible intervals (i.e., out$season$CI, out$trend$CI) for the estimated seasonal and trend components. This requires sorting and can be time-consuming. If only symmetric intervals are needed, use the standard deviations (out$trend$SD, out$season$SD) and set dump.ci = FALSE.

dump.mcmc — Logical; if TRUE, dumps individual MCMC samples for further custom analysis.

gui — Logical; if TRUE, launches a GUI window that animates MCMC sampling in the model space (numbers and timings of seasonal and trend changepoints). This experimental GUI is currently available only on 64-bit Windows, not on 32-bit Windows, macOS, or Linux.

... — Additional implementation-level arguments passed to the underlying beast123() engine. For full control over advanced settings, use beast123 directly.

Value

An object of class "beast", i.e., a list with components similar to those returned by beast and beast123. Below we assume the input y is a single time series of length N:

time — Numeric vector of length N; the regularized time axis after aggregating the original irregular series. If no explicit time is available, it defaults to 1:N.

`data` A vector, matrix, or 3D array containing a copy of the regularized input data if `extra$dumpInputData = TRUE`. Otherwise `NULL`. If the original input was irregular, this field stores the aggregated regular series at the time interval specified by `metadata$deltaTime`.

`marg_lik` Numeric; the (average) model marginal likelihood. Larger values correspond to better fits for a given time series (e.g., `-1` is better than `-10`; `10` is better than `-1`).

`R2` Numeric; coefficient of determination (R^2) for the fitted model.

`RMSE` Numeric; root mean squared error of the fitted model.

`sig2` Numeric; estimated variance of the model residuals.

`trend` List containing summaries for the estimated trend component:

- `ncp`: Mean number of trend changepoints across sampled models. Because individual models have varying numbers of changepoints, additional summaries are provided: `ncp_mode`, `ncp_median`, `ncp_pct90`. For example, if `mcmc$samples = 10` yields changepoint counts `c(0, 2, 4, 1, 1, 2, 7, 6, 6, 1)`, then the mean is 3.1, median 2.5, mode 1, and 90th percentile 6.5.
- `ncp_mode`: Mode of the posterior distribution of the number of trend changepoints.
- `ncp_median`: Median of the posterior distribution of the number of trend changepoints.
- `ncp_pct90`: 90th percentile of the posterior distribution of the number of trend changepoints.
- `ncpPr`: Vector of length `tcp.max + 1` where `tcp.max = tcp.minmax[2]`; gives the probability of having 0, 1, ..., `tcp.max` trend changepoints. For example, `ncpPr[1]` is the probability of having no trend changepoint; `ncpPr[i]` is the probability of having `i - 1` changepoints.
- `cpOccPr`: Numeric vector of length `N`; the marginal probability at each time index of being a trend changepoint. Plotting `cpOccPr` yields a continuous curve of "probability of being a changepoint". *Note*: a higher peak in this curve reflects a higher probability at that specific time index, but not necessarily a higher probability of a changepoint in a neighborhood around that time. For instance, a window `c(0, 0, 0.5, 0, 0)` (peak 0.5, sum 0.5) can be less likely to contain a changepoint than `c(0.1, 0.2, 0.21, 0.2, 0.1)` (peak 0.21, sum 0.71).
- `order`: Numeric vector of length `N`; average polynomial order of the trend at each time index, averaged over sampled piecewise polynomial trends. It is not necessarily an integer.
- `cp`: Numeric vector of length `tcp.max`; the most probable locations of trend changepoints. These are derived by smoothing `cpOccPr` with a window of length `tseg.min` and picking up to `prior$MaxKnotNum` (subject to `tcp.max`) of the highest peaks in the smoothed curve. Entries may be `NaN` if insufficient changepoints are identified. Many entries may be false positives; they should not be treated as confirmed changepoints without further scrutiny.
- `cpPr`: Numeric vector of length `tcp.max`; the probabilities associated with the detected changepoints in `cp`. Remaining entries are `NaN` if `ncp < tcp.max`.

- cpCI: Numeric matrix of dimension tcp.max x 2; credible intervals for the detected changepoints in cp.
- cpAbruptChange: Numeric vector of length tcp.max; the jumps in the fitted trend at the detected changepoints.
- Y: Numeric vector of length N; posterior mean estimate of the trend component (Bayesian model average over sampled models).
- SD: Numeric vector of length N; posterior standard deviation of the trend estimate.
- CI: Numeric matrix of dimension N x 2; credible intervals (lower and upper envelopes) for the trend component.
- slp: Numeric vector of length N; time-varying slope of the fitted trend.
- slpSD: Numeric vector of length N; posterior standard deviation of the trend slope.
- slpSgnPosPr: Numeric vector of length N; probability that the trend slope is positive (i.e., increasing). For example, slpSgnPosPr[t] = 0.8 means that at time index t, 80% of sampled trend curves have positive slope.
- slpSgnZeroPr: Numeric vector of length N; probability that the trend slope is approximately zero (flat). The probability of a negative slope is given by 1 - slpSgnZeroPr - slpSgnPosPr.
- pos_ncp, neg_ncp, pos_ncpPr, neg_ncpPr, pos_cpOccPr, neg_cpOccPr, pos_cp, neg_cp, pos_cpPr, neg_cpPr, pos_cpAbruptChange, neg_cpAbruptChange, pos_cpCI, neg_cpCI:
  As above, but distinguishing changepoints where the trend *jumps up* (positive jump) versus *jumps down* (negative jump). For example, pos_ncp is the average number of changepoints with positive jumps.
- inc_ncp, dec_ncp, inc_ncpPr, dec_ncpPr, inc_cpOccPr, dec_cpOccPr, inc_cp, dec_cp, inc_cpPr, dec_cpPr, inc_cpAbruptChange, dec_cpAbruptChange, inc_cpCI, dec_cpCI:
  Analogous to the above, but distinguishing changepoints where the trend *slope* increases vs. decreases. For instance, if the slope changes from 0.4 to 2.5 at a changepoint, that changepoint contributes to inc_ncp.

season — List containing summaries for the estimated seasonal/periodic component (if present):

- ncp: Mean number of seasonal changepoints.
- ncpPr: Vector of length scp.max + 1, where scp.max = scp.minmax[2]; the probability of having 0, 1, ..., scp.max seasonal changepoints.
- cpOccPr: Numeric vector of length N; marginal probability that each time index is a seasonal changepoint. The same interpretation caveats as for the trend component apply.
- order: Numeric vector of length N; average harmonic order required to approximate the seasonal component. This is an average over sampled piecewise harmonic curves and is not necessarily an integer.
- cp: Numeric vector of length scp.max; the most probable seasonal changepoint locations, derived from the smoothed cpOccPr curve using a window size of sseg.min. If ncp < scp.max, the remaining entries are filled with NaN.

- cpPr: Numeric vector of length scp.max; probabilities associated with cp. Remaining entries are NaN if ncp < scp.max.
- cpCI: Numeric matrix of dimension scp.max x 2; credible intervals for the detected seasonal changepoints.
- cpAbruptChange: Numeric vector of length scp.max; jumps in the fitted seasonal component at the detected changepoints.
- Y: Numeric vector of length N; posterior mean estimate of the seasonal component.
- SD: Numeric vector of length N; posterior standard deviation of the seasonal estimate.
- CI: Numeric matrix of dimension N x 2; credible intervals (lower and upper envelopes) for the seasonal component.
- amp: Numeric vector of length N; time-varying amplitude of the seasonal component.
- ampSD: Numeric vector of length N; posterior standard deviation of the amplitude.
- pos_ncp, neg_ncp, pos_ncpPr, neg_ncpPr, pos_cpOccPr, neg_cpOccPr, pos_cp, neg_cp, pos_cpPr, neg_cpPr, pos_cpAbruptChange, neg_cpAbruptChange, pos_cpCI, neg_cpCI:
  Analogous to the trend component, but for seasonal changepoints that correspond to positive vs. negative jumps in the seasonal signal.

outlier — outlier is a list analogous to trend or season, but for the outlier component (present only if setting hasOutlier=TRUE)

Note

The three functions beast(), beast.irreg(), and beast123() implement the same BEAST algorithm but expose different APIs:

- beast() operates on regular (evenly spaced) time series.
- beast.irreg() accepts irregular/unordered data and internally aggregates it to regular intervals.
- beast123() exposes a low-level interface via four lists: metadata, prior, mcmc, and extra.

There is a one-to-one correspondence between arguments of beast() / beast.irreg() and fields in metadata, prior, mcmc, and extra used by beast123(), Examples include:

```
start               <->  metadata$startTime
deltat              <->  metadata$deltaTime
deseasonalize       <->  metadata$deseasonalize
hasOutlier          <->  metadata$hasOutlier
scp.minmax[1:2]     <->  prior$seasonMinKnotNum, prior$seasonMaxKnotNum
sorder.minmax[1:2]  <->  prior$seasonMinOrder, prior$seasonMaxOrder
sseg.min            <->  prior$seasonMinSepDist
tcp.minmax[1:2]     <->  prior$trendMinKnotNum, prior$trendMaxKnotNum
torder.minmax[1:2]  <->  prior$trendMinOrder, prior$trendMaxOrder
```

```
tseg.leftmargin     <->  prior$trendLeftMargin
tseg.rightmargin    <->  prior$trendRightMargin
s.complexfct        <->  prior$seasonComplexityFactor
t.complexfct        <->  prior$trendComplexityFactor
mcmc.seed           <->  mcmc$seed
dump.ci             <->  extra$computeCredible.
```

Advanced users who need full control over all internal options are encouraged to use `beast123()` directly.

References

1. Zhao, K., Wulder, M. A., Hu, T., Bright, R., Wu, Q., Qin, H., Li, Y., Toman, E., Mallick, B., Zhang, X. and Brown, M. (2019). Detecting change-point, trend, and seasonality in satellite time series data to track abrupt changes and nonlinear dynamics: A Bayesian ensemble algorithm. *Remote Sensing of Environment*, 232, 111181. (The main BEAST algorithm paper.)
2. Zhao, K., Valle, D., Popescu, S., Zhang, X. and Mallick, B. (2013). Hyperspectral remote sensing of plant biochemistry using Bayesian model averaging with variable and band selection. *Remote Sensing of Environment*, 132, 102–119. (The Bayesian MCMC scheme used in BEAST.)
3. Hu, T., Toman, E. M., Chen, G., Shao, G., Zhou, Y., Li, Y., Zhao, K. and Feng, Y. (2021). Mapping fine-scale human disturbances in a working landscape with Landsat time series on Google Earth Engine. *ISPRS Journal of Photogrammetry and Remote Sensing*, 176, 250–261. (Example application of BEAST.)

See Also

`beast`, `beast123`, `minesweeper`, `tetris`, `geeLandsat`

Examples

```
library(Rbeast)



################################################################################
# Note: The BEAST algorithm is implemented for regular time series.
# \code{beast.irreg} accepts irregular data but internally aggregates it to a
# regular series before applying BEAST. For aggregation, both "time" and
# "deltat" are needed to specify the original timestamps via "time" and the desired
# regular interval via "deltat". If a cyclic component exists, "period" should also
# be provided; otherwise BEAST attempts to guess it via autocorrelation.
################################################################################

# 'ohio' is a data.frame containing an irregular Landsat time series of
# reflectances and NDVI at an Ohio site. It includes multiple alternative
# date formats: year (Y), month (M), day (D), day-of-year (doy), R "Date"
# (rdate), and fractional year (time).
```

```
data(ohio)
str(ohio)
plot(ohio$rdate, ohio$ndvi, type = "o")  # NDVI is irregular and unordered in time

################################################################################
# Example 1: 'time' as numeric values
# Below, 'time' is given as numeric values, which can be of any arbitray unit. Although
# here 1/12 can be interepreted as 1/12 year, BEAST itself doesn't care about the unit.
# So, the unit of 1/12 is irrelevant for BEAST. 'period' is missing
# and a guess of it is used.
################################################################################

o <- beast.irreg(ohio$ndvi, time = ohio$time, deltat = 1/12)
plot(o)
print(o)

################################################################################
# Example 2: Aggregate to a monthly interval (deltat = 1/12) and provide 'period'
################################################################################

o <- beast.irreg(ohio$ndvi, time = ohio$time, deltat = 1/12, period = 1.0)
# Alternative (deprecated argument 'freq'):
# o <- beast.irreg(ohio$ndvi, time = ohio$time, deltat = 1/12, freq = 12)



## Not run:

################################################################################
# Example 3: Aggregate at a half-monthly interval.
# Here period = 1: freq = period/deltat = 1/(1/24)=24 data points per period
################################################################################

o <- beast.irreg(ohio$ndvi, time = ohio$time, deltat = 1/24, period = 1)

################################################################################
# Example 4: 'time' as  R Dates (i.e, ohio$rdate). Unit is YEAR; 1/12 is ~1 month.
################################################################################

o <- beast.irreg(ohio$ndvi, time = ohio$rdate, deltat = 1/12)

################################################################################
# Example 5: 'time' as date strings. The unit is YEAR such that 1/12 is a month
################################################################################

o=beast.irreg(ohio$ndvi, time=ohio$datestr1,deltat=1/12) #"LT4-1984-03-27"  (YYYY-MM-DD)
o=beast.irreg(ohio$ndvi, time=ohio$datestr2,deltat=1/12) #"LT4-1984087ndvi" (YYYYDOY)
o=beast.irreg(ohio$ndvi, time=ohio$datestr3,deltat=1/12) #"1984,, 3/ 27"    (YYYY M D)

################################################################################
# Example 6: 'time' as date strings with explicit format specifiers
################################################################################
```

```
TIME <- list()
TIME$datestr <- ohio$datestr1
TIME$strfmt  <- "LT4-YYYY-MM-DD"    # e.g., "LT4-1984-03-27"
o <- beast.irreg(ohio$ndvi, time = TIME, deltat = 1/12)

TIME <- list()
TIME$datestr <- ohio$datestr2
TIME$strfmt  <- "LT4-YYYYDOYndvi"   # e.g., "LT4-1984087ndvi"
o <- beast.irreg(ohio$ndvi, time = TIME, deltat = 1/12)

################################################################################
# Example 7: 'time' as a list of date components
################################################################################

TIME <- list()
TIME$year  <- ohio$Y
TIME$month <- ohio$M
TIME$day   <- ohio$D
o <- beast.irreg(ohio$ndvi, time = TIME, deltat = 1/12)

TIME <- list()
TIME$year <- ohio$Y
TIME$doy  <- ohio$doy
o <- beast.irreg(ohio$ndvi, time = TIME, deltat = 1/12)


## End(Not run)
```

`beast123` *Bayesian time series decomposition into changepoints, trend, and seasonal/periodic components*

Description

BEAST is a Bayesian model averaging algorithm for decomposing time series or other 1D sequential data into individual components, including abrupt changes, trends, and periodic/seasonal variations. It is useful for changepoint detection (e.g., breakpoints or structural breaks), nonlinear trend analysis, time series decomposition, and time series segmentation. `beast123` is a low-level interface to BEAST.

Usage

```
beast123(
  Y,
  metadata = list(),
  prior    = list(),
  mcmc     = list(),
```

```
  extra    = list(),
  season   = c("harmonic","svd","dummy","none"),
  method   = c("bayes","bic","aic","aicc","hic",
               "bic0.25","bic0.5","bic1.5","bic2"),
  ...
)
```

Arguments

Y	Y is a numeric 1D vector, 2D matrix, or 3D array. Missing values are allowed and can be indicated by NA, NaN, or a customized value specified via the 2nd argument metadata (e.g., metadata$missingValue=-9999). • If Y is a vector of size N x 1 or 1 x N, it is treated as a single time series of length N. • If Y is a 2D matrix or 3D array of dimension N1 x N2 or N1 x N2 x N3 (e.g., stacked images of geospatial data), it contains multiple time series of equal length. The dimension corresponding to time must be specified in metadata$whichDimIsTime. For example, metadata$whichDimIsTime = 1 for a 190 x 35 matrix indicates 35 time series of length 190; metadata$whichDimIsTime = 2 for a 100 x 200 x 300 array indicates 100 * 300 = 30000 time series of length 200 each. Y can be either regular (evenly spaced in time) or irregular/unordered in time. • *Regular data:* Individual times are determined from the time of the first data point (startTime) and the time interval between consecutive points (deltaTime), specified via metadata$startTime and metadata$deltaTime. If not provided, both default to 1.0. • *Irregular or unordered data:* The times must be explicitly supplied via metadata$time. The BEAST model is currently formulated for regular data only, so beast123 internally aggregates/re-bins irregular data into a regular series. For this aggregation, metadata$deltaTime should also be provided to specify the desired bin size (time interval) of the regularized time series. Y may contain both a periodic component and a trend, or only a trend. Use the argument season to specify the case: • season = "none": Y is treated as trend-only; no periodic components are assumed. • season = "harmonic": Y has a periodic/seasonal component. The term "season" is used broadly here to refer to any periodic variation. The period is not a model parameter estimated by BEAST; instead it must be supplied by the user via metadata$period, the unit of which should be consistent with that of metadata$deltaTime (see metadata$period below). If season = "harmonic", The seasonal component is modeled as a harmonic curve (a combination of sines and cosines). • season = "dummy": As for "harmonic", except that the periodic/seasonal component is modeled as a non-parametric curve.

- `season = "svd"` (experimental): As for `"harmonic"`, except that the periodic/seasonal component is modeled as a linear combination of basis functions derived from a singular value decomposition (SVD). The SVD-based basis functions can be more parsimonious than harmonic bases in parameterizing seasonal variations and may facilitate the detection of more subtle seasonal changepoints.

`metadata` `metadata` is optional. If present, it can be:

- a scalar specifying the period of `Y`;
- a vector of numbers, character strings, or `Dates` specifying the times of `Y`; or
- more commonly, a `list` object specifying various parameters describing the first argument `Y`.

If missing, default values are used. However, when `Y` is a 2D matrix or 3D array, `metadata` should be supplied explicitly to avoid misinterpreting which dimension corresponds to time. `metadata` is not part of the Bayesian BEAST model itself; it provides additional information needed to correctly interpret `Y`.

When `metadata` is a list, the following fields are supported (not all are needed in every case; the relevant subset depends on the input type and regularity of the series):

- `metadata$whichDimIsTime`: Integer (`<= 3`). Specifies which dimension of `Y` corresponds to time for 2D or 3D inputs. Ignored if `Y` is a vector.
- `metadata$isRegularOrdered`: Logical. Obsolete and no longer used in this version. Regularity is now inferred from `metadata$time`; if `metadata$time` is missing, `Y` is assumed to be regular.
- `metadata$time`: A vector of the same length as the time dimension of `Y`, or a more complex object providing time information. It can be a vector of numbers, Dates, or date strings; it can also be a list of vectors of year, months, and days. Possible formats include:
  1. A vector of numeric values (e.g., `c(1984.23, 1984.27, 1984.36, ...)`). The time unit is arbitrary but must be consistent with the units used in `metadata$startTime`, `metadata$deltaTime`, and `metadata$period`.
  2. A vector of `Dates` (e.g., `as.Date(c("1984-03-27","1984-04-10","1984-05-12", ...))`).
  3. A vector of character strings. Examples are:
     - `c("1984-03-27","1984-04-10","1984-05-12")`
     - `c("1984/03/27","1984,04,10","1984 05 12")` (delimiters can differ as long as the Y-M-D order is consistent)
     - `c("LT4-1984-03-27","LT4-1984-04-10","LT4-1984+05,12")`
     - `c("LT4-1984087ndvi","LT4-1984101ndvi","LT4-1984133ndvi")`
     - `c("1984,,abc 3/ 27","1984,,ddxfdd 4/ 10","ggd1984,, 5/ ttt 12")`

     BEAST uses several heuristics to parse date strings without an explicit format specifier, but may fail when strings are ambiguous (e.g., `"LC8-2020-09-20-1984"`, where both 2020 and 1984 look like possible years). To ensure correct parsing, use a list with an explicit date format specifier (see next item).

4. A list object `time = list(datestr = ..., strfmt = "...")` consisting of:
   - `time$datestr`: a character vector of date strings, and
   - `time$strfmt`: a format string that specifies how to parse `datestr`.

   Three types of formats are supported:
   - (a) Fixed pattern of year, month, and day positions. For example, to extract 2001/12/02, 2002/11/08, etc. from `time$datestr = c("P23R34-2001.1202333xd","O93X94-2002.1108133fd","TP3R34-2009.0122333td")`, use `time$strfmt = "P23R34-yyyy.mmdd333xd"`, where `yyyy`, `mm`, and `dd` denote the year, month, and day. All other positions are treated as wildcards and nd can be filled with any other letters different from `yyyy`, `mm` and `dd`..
   - (b) Fixed pattern of year and day-of-year (DOY) positions. For example, to extract year and DOY from `"P23R342001888045"`, use `time$strfmt = "123123yyyy888doy"`, where `yyyy` and `doy` are specifiers and other characters are wildcards. `doy` must be three digits.
   - (c) Fixed pattern of separators between year, month, and day. For example, to get 2002/12/02 from `"2002,12/02"`, `" 2002 , 12/2"`, or `"2002,12 /02"`, use `time$strfmt = "Y,M/D"`, where whitespace is ignored. To get 2002/12/02 from `"2--12, 2012"`, use `time$strfmt = "D--M,Y"`.
5. A list object of separate vectors for dates. Use `time$year`, `time$month`, and `time$day` to specify dates, or alternatively `time$year` and `time$doy` (day of year, 1-365/366). Each vector must have the same length as the time dimension of `Y`.

- `metadata$startTime`: Numeric (defaults to `1.0` if missing). The time of the first data point. It can be given as a scalar (e.g., `2021.23`) or as a vector of length 3 in the form of `c(year, month, day)` (e.g., `c(2021, 1, 24)`). `metadata$startTime` is needed for regular data; for irregular data it is optional and, if missing, may be derived from `metadata$time`.
- `metadata$deltaTime`: Numeric or character string. The time interval between consecutive data points. For regular data it is optional (default `1.0`); for irregular data it should be specified, as it determines the bin size when aggregating irregular times to regular intervals. If `metadata$time` is numeric, the unit of `deltaTime` is arbitrary but must be consistent with the numeric scale of `metadata$time`. If `metadata$time` is given as `Dates` or date strings, `deltaTime` is interpreted as a fraction of a year by default. Alternatively, use a string instead of a number to specify the interval with a unit, such as `deltaTime="7 days"`, `"7d"`, `"1/2 months"`, `"1mn"`, `"1 year"`, or `"1y"`.
- `metadata$period`: Numeric or character string. The period of the periodic/seasonal component in `Y`. This is needed only when a seasonal component is present (i.e., `season = "harmonic"` or `season = "dummy"`) and is ignored for trend-only data (`season = "none"`). The period must be in the same time unit as `deltaTime` and should satisfy `period = deltaTime * freq`, where `freq` is the number of data points per period. (The older argument `freq` is now obsolete and replaced by `period`.) The period is not a

BEAST model parameter and must be provided by the user. But if `period` is missing, BEAST first attempts to infer a plausible period via autocorrelation before fitting. If `period <= 0`, no seasonal component is assumed (equivalent to `season = "none"`). As with `deltaTime`, character strings such as `period="1 year"`, `"12 months"`, `"365d"`, or `"366 days"` can be used.

- `metadata$missingValue`: Numeric; a customized value indicating bad or missing values in addition to `NA` or `NaN`.
- `metadata$maxMissingRateAllowed`: Numeric in `[0, 1]`; the maximum proportion of missing values allowed. Time series with a missing rate higher than this threshold are skipped and not fitted by BEAST.
- `metadata$hasOutlier`: Logical; if `TRUE`, fit a model with an outlier component representing potential spikes or dips at isolated data points: `Y = trend + outlier + error` if `season = "none"`, and `Y = trend + season + outlier + error` otherwise.
- `metadata$deseasonalize`: Logical; if `TRUE`, remove a global seasonal component before fitting BEAST (see examples).
- `metadata$detrend`: Logical; if `TRUE`, remove a global trend before fitting BEAST.

`prior` `prior` (optional) is a `list` of hyperparameters defining the priors in the Bayesian BEAST model. Because these are part of the model specification, the BEAST results may be sensitive to their values. If `prior` is missing, default values are used and printed to the console at the start of the run. The following fields are supported:

- `prior$seasonMinOrder`: Integer (`>= 1`).
- `prior$seasonMaxOrder`: Integer (`>= 1`). The minimum and maximum harmonic orders considered for the seasonal component. These are used only when the series has a seasonal component (i.e., `season = "harmonic"` or `"svd"`) and ignored for trend-only data or when `season='none'` or `'dummy'`. If `seasonMinOrder == seasonMaxOrder`, the harmonic order is fixed and BEAST does not infer the posterior distribution of harmonic orders.
- `prior$seasonMinKnotNum`: Integer (`>= 0`).
- `prior$seasonMaxKnotNum`: Integer (`>= 0`). The minimum and maximum numbers of seasonal changepoints allowed in segementing the seasonal component. Used only when the data have a a seasonal component (e.g.,`season = "harmonic"`, `"svd"` or `"dummy"` and ignored for trend-only data. If `seasonMinKnotNum == seasonMaxKnotNum`, the number of seasonal changepoints is fixed but their possible locations and occurrence probabilities over time are still estimated. If `seasonMinKnotNum == seasonMaxKnotNum == 0`, no seasonal changepoints are allowed and a single global harmonic model is used with no changepoints.
- `prior$seasonMinSepDist`: Integer (`> 0`). The minimum separation, in number of time steps, between neighboring seasonal changepoints. That is, when fitting a piecewise seasonal model, no two changepoints may be closer than `seasonMinSepDist` time intervals. The corresponding time window in original time units is `seasonMinSepDist * metadata$deltaTime`.

- prior$seasonLeftMargin: Integer (>= 0). The number of leftmost observations excluded from seasonal changepoint detection. No seasonal changepoints are allowed in the initial window of length seasonLeftMargin. It is specified in units of time steps and thus corresponds to a time window of seasonLeftMargin * metadata$deltaTime. If missing, it defaults to seasonMinSepDist.
- prior$seasonRightMargin: Integer (>= 0). The number of rightmost observations excluded from seasonal changepoint detection. Similarly, no seasonal changepoints are allowed in the final window of length seasonRightMargin. Specified in time steps and corresponds to seasonRightMargin * metadata$deltaTime in the time unit. If missing, it defaults to seasonMinSepDist.
- prior$seasonComplexityFactor: Numeric (default 0.0). A hyperprior parameter–newly added in Version 1.02– controlling the complexity of the seasonal curve (i.e., the number of seasonal changepoints). A prior of the form $p(k) \propto \exp[\lambda(k+1)]$ is placed on the number of seasonal changepoints k, where λ is seasonComplexityFactor. Setting $\lambda = 0$ yields a non-informative prior $p(k) \propto 1.0$ where all model dimensions are equally likely *a priori*. Users may tune seasonComplexityFactor in the range [-20, 20] or an even wider range: negative values (e.g., $\lambda = -15.9$) favor fewer changepoints (simpler seasonal curves), whereas positive values (e.g., $\lambda = 5.76$) favor more changepoints (more complex curves).
- prior$trendMinOrder: Integer (>= 0).
- prior$trendMaxOrder: Integer (>= 0). The minimum and maximum polynomial orders considered to fit the trend component. Order 0 corresponds to a constant (flat) trend; order 1 to a linear trend. If trendMinOrder == trendMaxOrder, the polynomial order is fixed and BEAST does not infer the posterior distrubtion of polynomial orders.
- prior$trendMinKnotNum: Integer (>= 0).
- prior$trendMaxKnotNum: Integer (>= 0). The minimum and maximum numbers of trend changepoints allowed. If trendMinKnotNum == trendMaxKnotNum, the number of trend changepoints is fixed but ther locations and occurrence probabilities over time are still estimated. If trendMinKnotNum == trendMaxKnotNum == 0, no trend changepoints are allowed and a single global polynomial trend is fitted.
- prior$trendMinSepDist: Integer (> 0). The minimum separation, in number of time steps, between neighboring trend changepoints. That is, when fitting a piecewise tremd model, no two changepoints may be closer than trendMinSepDist time intervals. The corresponding time window in original time units is trendMinSepDist * metadata$deltaTime.
- prior$trendLeftMargin: Integer (>= 0). The number of leftmost observations excluded from trend changepoint detection. No trend changepoints are allowed in the initial window of length trendLeftMargin. Specified in time steps; the corresponding time window is trendLeftMargin * metadata$deltaTime. If missing, defaults to trendMinSepDist.
- prior$trendRightMargin: Integer (>= 0). The number of rightmost observations excluded from trend changepoint detection. No trend changepoints are allowed in the final window of length trendRightMargin. Spec-

ified in time steps; the corresponding time window is `trendRightMargin * metadata$deltaTime`. If missing, defaults to `trendMinSepDist`.

- `prior$trendComplexityFactor`: Numeric (default `0.0`). Analogous to `seasonComplexityFactor`, but for the trend component. A prior of the form $p(k) \propto \exp[\lambda(k+1)]$ is placed on the number of trend changepoints k, where λ is `trendComplexityFactor`. Setting $\lambda = 0$ yields a non-informative prior $p(k) \propto 1.0$ where all model dimensions are equally likely *a priori*. Users may tune `seasonComplexityFactor` in the range `[-20, 20]` or an even wider range: negative values (e.g., $\lambda = -15.9$) favor fewer changepoints (simpler trend curves), whereas positive values (e.g., $\lambda = 5.76$) favor more changepoints (more complex curves).
- `prior$precValue`: Numeric (`> 0`); default `10`. Useful only if `prior$precPriorType = "constant"`.
- `prior$precPriorType`: Character string, one of `"constant"`, `"uniform"` (default), `"componentwise"`, or `"orderwise"`. These options control how precision parameters (for model coefficients) are treated:
  1. `"constant"`: A single precision parameter is fixed at `prior$precValue`. The results may be sensitive to the choice of `prior$precValue`.
  2. `"uniform"`: A single precision parameter is treated as a random variable, with initial value `prior$precValue`. Its posterior is inferred via MCMC, making the results less sensitive to the initial choice of `prior$precValue`.
  3. `"componentwise"`: Separate precision parameters are used for different components (e.g., season vs. trend), starting from `prior$precValue`. Each precision parameter is inferred via MCMC.
  4. `"orderwise"`: Separate precision parameters are used for different components and different orders within components, starting from `prior$precValue`. All the precision parameters are inferred via MCMC.
- `prior$outlierMinKnotNum`: Integer (`>= 0`).
- `prior$outlierMaxKnotNum`: Integer (`>= 0`). These are used only if `metadata$hasOutlier = TRUE` to specify the minimum and maximum numbers of outlier-type changepoints (spikes or dips) allowed in the series.

`mcmc` `mcmc` (optional) is a `list` object of parameters configuring the reversible-jump MCMC procedure. These settings are not part of the Bayesian model itself but control the sampling process and chain length. In general, longer chains yield more stable estimates. Supported fields are:

- `mcmc$seed`: Integer (`>= 0`); the seed for the random number generator. If `mcmc$seed = 0`, an arbitrary seed is used and results can vary slightly across runs. Using the same non-zero seed makes results reproducible on the same machine, but differences across platforms/CPUs may still occur from the same seed due to differences in random number libraries or CPU instruction sets.
- `mcmc$chainNumber`: Integer (`> 0`); the number of parallel MCMC chains.
- `mcmc$samples`: Integer (`> 0`); the number of samples collected per MCMC chain.
- `mcmc$thinningFactor`: Integer (`> 0`); thinning factor. If `thinningFactor = k`, every `k`-th sample is retained and the others discarded.

- mcmc$burnin: Integer (> 0); the number of initial samples discarded as burn-in per chain.
- mcmc$maxMoveStepSize: Integer (> 0). In the RJMCMC sampler, a "move" proposal shifts changepoint locations. maxMoveStepSize specifies the maximum step size (in time steps) allowed in such moves.
- mcmc$seasonResamplingOrderProb: Numeric in (0, 1); probability of proposing a resampling of the seasonal harmonic order.
- mcmc$trendResamplingOrderProb: Numeric in (0, 1); probability of proposing a resampling of the trend polynomial order.
- mcmc$credIntervalAlphaLevel: Numeric in (0, 1) (default 0.95); the confidence level used to compute credible intervals (e.g., 0.95 for 95% credible intervals).

extra

extra (optional) is a list of flags and options controlling output content, console messages, and parallel computation. Supported fields include:

- extra$dumpInputData: Logical (default FALSE). If TRUE, copies of the input time series are included in the output. If the input Y is irregular the dumped copy is the aggregated regular series.
- extra$whichOutputDimIsTime: Integer (<= 3). For 2D or 3D inputs (multiple time series, e.g., stacked images), this specifies which output dimension corresponds to time. Defaults to 3 for 3D inputs. Ignored if Y is a single time series.
- extra$ncpStatMethod: Character string (deprecated). Formerly used to specify the statistic used to determine the number of changepoints (ncp) when computing the most likely changepoint locations (e.g., out$trend$cp and out$season$cp). Possible values were "mode", "mean", and "median" (default "mode"). Individual MCMC-sampled models have a varying number of changepoints. For example, if mcmc$samples=10 and assuming that the numbers of changepoints for the 10 sampled models are [0, 2, 4, 1, 1, 2, 7, 6, 6, 1], the mean ncp is 3.1 (rounded to 3), the median is 2.5 (2), and the mode is 1. ncpStatMethod is now deprecated: BEAST outputs all possible changepoints together with several summaries of ncp, including ncp, ncp_median, ncp_mode, and ncp_pct90. The choice of ncp for plotting is now controlled by the ncpStat argument in plot.beast.
- extra$computeCredible: Logical (default TRUE). If TRUE, credible intervals are computed and returned.
- extra$fastCIComputation: Logical (default TRUE). If TRUE, a faster approximate method is used to compute credible intervals. CI computation can be one of the most time-consuming steps, so fastCIComputation = TRUE is recommended unless more precise CI estimates are desired.
- extra$computeSeasonOrder: Logical (default TRUE). If TRUE, a posterior estimate of the seasonal harmonic order is returned. Only meaningful when season = "harmonic" or "svd" and prior$seasonMinOrder != prior$seasonMaxOrder.
- extra$computeTrendOrder: Logical (default TRUE). If TRUE, a posterior estimate of the trend polynomial order is returned. Only meaningful when prior$trendMinOrder != prior$trendMaxOrder.

- extra$computeSeasonChngpt: Logical (default TRUE). If TRUE, compute the most likely times/positions at which changepoints occur in the seasonal component. This is relevant only when a seasonal component is present (i.e., season = "harmonic", "svd", or "dummy") and prior$seasonMaxKnotNum > 0.
- extra$computeTrendChngpt: Logical (default TRUE). If TRUE, compute the most likely changepoint locations in the trend component. This is relevant only when prior$trendMaxKnotNum > 0.
- extra$computeSeasonAmp: Logical (default FALSE). If TRUE, output the time-varying amplitude of the seasonal component.
- extra$computeTrendSlope: Logical (default FALSE). If TRUE, output the time-varying slope of the trend component.
- extra$tallyPosNegSeasonJump: Logical (default FALSE). If TRUE, classify seasonal changepoints by the sign of the jump (positive vs. negative) and output separate summaries for each group.
- extra$tallyPosNegTrendJump: Logical (default FALSE). If TRUE, classify trend changepoints by whether the trend level jumps up or down at the changepoint and output separate summaries.
- extra$tallyIncDecTrendJump: Logical (default FALSE). If TRUE, classify trend changepoints by changes in slope (increasing vs. decreasing) and output separate summaries for those changepoints with increased slopes and those with decreased slopes.
- extra$printProgress: Logical (default FALSE). If TRUE, a progress bar is displayed showing the progress of the run. For multiple time series (e.g., stacked images), the progress bar also reports an estimate of remaining time.
- extra$consoleWidth: Integer (default 0). The width (in characters) of each status line printed when extra$printProgress = TRUE. If 0, the width of the current console is used.
- extra$printParameter: Logical (default TRUE). If TRUE, the values used in metadata, prior, mcmc, and extra are printed at the start of the run.
- extra$printWarning: Logical (default TRUE). If TRUE, warning messages are printed.
- extra$quiet: Logical. If TRUE, suppress most console output.
- extra$dumpMCMCSamples: Logical. If TRUE, dump individual MCMC samples.
- extra$numThreadsPerCPU: Integer (default 2); the number of threads scheduled per CPU core.
- extra$numParThreads: Integer (default 0). Total number of parallel threads used when fitting many time series. If 0, the actual number of threads is set to extra$numThreadsPerCPU * (number of CPU cores). That is, each CPU core will generate a number 'numThreadsPerCPU' of thread. On 64-bit Windows, BEAST is group-aware and distributes threads across NUMA nodes. Currently, up to 256 CPU cores are supported.

season — Character (default "harmonic"). Specifies whether Y has a periodic component and how it is modeled:

- `"none"`: `Y` is trend-only; no periodic components are modeled. Seasonal prior parameters (e.g., seasonMaxKnotNum and seasonMinSepDist) are ignored.
- `"harmonic"`: `Y` has a periodic/seasonal component modeled as a harmonic series (e.g., combination of sines and cosines). The term `season` is used broadly to refer to any periodic variation. The period is not estimated but must be specified by the user (via `metadata$period`); see `metadata$period`.
- `"dummy"`: As `"harmonic"`, except that the seasonal component is modeled as a non-parametric curve. The harmonic-order prior settings (e.g., `prior$seasonMinOrder` and `prior$seasonMaxOrder`) are irrelevant and ignored if `season="dummy"`.
- `"svd"`: (Experimental) As `"harmonic"`, except that the seasonal component is expressed as a linear combination of basis functions derived from a Single-value decomposition. These basis functions can be more parsimonious than pure harmonic bases, and may improve detection of subtle seasonal changepoints.

`method` Character (default `"bayes"`). Specifies how the model posterior probability is formulated or approximated:

- `"bayes"`: Full Bayesian formulation as described in Zhao et al. (2019).
- `"bic"`: Approximate posterior probabilities using the Bayesian Information Criterion (BIC), $\mathrm{BIC} = n\log(\mathrm{SSE}) + k\log(n)$, where k is the number of parameters and n the number of observations.
- `"aic"`: Approximate posterior probabilities using Akaike's Information Criterion (AIC), $\mathrm{AIC} = n\log(\mathrm{SSE}) + 2k$.
- `"aicc"`: Approximate posterior probabilities using the corrected AIC (AICc), $\mathrm{AICc} = \mathrm{AIC} + \frac{2k^2+2k}{n-k-1}$.
- `"hic"`: Approximate posterior probabilities using the Hannan-Quinn Information Criterion (HIC), $\mathrm{HIC} = n\log(\mathrm{SSE}) + 2k\log(\log(n))$.
- `"bic0.25"`: BIC-type approximation adopted from Kim et al. (2016) doi:10.1016/j.jspi.2015.09.008 with reduced complexity penalty, $\mathrm{BIC}_{0.25} = n\log(\mathrm{SSE}) + 0.25k\log(n)$.
- `"bic0.5"`: As above but with penalty factor `0.5`.
- `"bic1.5"`: As above but with penalty factor `1.5`.
- `"bic2"`: As above but with penalty factor `2.0`.

`...` Additional parameters reserved for future extensions; currently unused.

Value

The output is a list object of class `"beast"` containing the following fields. Exact dimensions depend on the input `Y` and `extra$whichOutputDimIsTime`. The descriptions below assume a single time series input of length `N`; for 2D/3D inputs, interpret the results according to the 2D/3D dimensions.

`time` A numeric vector of length `N`: the time associated with each sampled observation. By default, this is simply `1:N` if `metadata$time`, `metadata$startTime`, or `metadata$deltaTime` are missing.

`data` A vector, matrix, or 3D array; a copy of the input `Y` if `extra$dumpInputData = TRUE`. If `extra$dumpInputData = FALSE`, this is `NULL`. For irregular inputs in `Y` , this field stores the aggregated regular series at time intervals spaced by `metadata$deltaTime`.

`marg_lik` Numeric; the average marginal likelihood of the sampled models. Larger values indicate better fit for a given time series.

`R2` Numeric; coefficient of determination (R-squared) of the fitted model.

`RMSE` Numeric; root mean squared error of the fitted model.

`sig2` Numeric; estimated variance of the residual error.

`trend` `trend` is a `list` of outputs related to the estimated trend component:

- `ncp`: Numeric scalar; mean number of trend changepoints. Because individual models sampled by BEAST can have varying numbers of changepoints, several alternative summaries (`ncp_mode`, `ncp_median`, `ncp_pct90`) are also provided. For example, if `mcmc$samples = 10` and the numbers of changepoints across models are `c(0, 2, 4, 1, 1, 2, 7, 6, 6, 1)`, then the mean number of changepoints (ncp) is `3.1`, the median `2.5`, the mode `1`, and the 90th percentile (`ncp_pct90`) `6.5`.
- `ncp_mode`: Mode of the number of trend changepoints. See the above for explanations.
- `ncp_median`: Median of the number of trend changepoints. See the above for explanations.
- `ncp_pct90`: 90th percentile of the number of trend changepoints. See the above for explanations.
- `ncpPr`: Numeric vector of length `prior$trendMaxKnotNum + 1`. Probability distribution over the number of trend changepoints in the range `0:prior$trendMaxKnotNum`. For example, `ncpPr[1]` is the probability of having no trend changepoint; `ncpPr[i]` is the probability of having `i - 1` changepoints.
- `cpOccPr`: Numeric vector of length `N`. The changepoint occurrence probability at each time index for the trend component. Plotting `cpOccPr` yields a continuous probability-of-being-a-changepoint curve. A peak with a high value at a single time step does not necessarily imply a higher *overall* probability of a changepoint in a neighborhood, compared to a broader peak with a lower maximum but higher total probability. Specifically, a higher peak indicates a higher chance of being a changepoint only at that particular SINGLE point in time and but not necessarily mean a higher chance of observing a changepoint AROUND that time. For example, a window of cpOccPr values `c(0,0,0.5,0,0)` (i.e., the peak prob is 0.5 and the summed probability over the window is 0.5) is less likely to be a changepoint compared to another window `c(0.1,0.2,0.21,0.2,0.1)` where the peak of 0.21 is lower but the summed probablity of 0.71 is larger.
- `order`: Numeric vector of length `N`. The average polynomial order of the trend across sampled models at each time step. Because this is a posterior average, it need not be an integer.
- `cp`: Numeric vector (up to length `prior$trendMaxKnotNum`). Estimated trend changepoint locations. These are obtained by smoothing `cpOccPr` with a window of size `prior$trendMinSepDist` and then selecting up to

prior$trendMaxKnotNum peaks in the smoothed curve. Some entries may be NaN if the number of detected changepoints is fewer than trendMaxKnotNum. Not all listed changepoints should be treated as "true" changepoints; some may be false positives. Check the associated posterior probabiliites (given in the next field trend$cpPr for the confidence levels.

- cpPr: Numeric vector of length prior$trendMaxKnotNum; the posterior probabilities associated with each changepoint in cp. Trailing entries are NaN if fewer than prior$trendMaxKnotNum changepoints are identified.
- cpCI: Numeric matrix of dimension prior$trendMaxKnotNum x 2; credible intervals for the changepoint locations in cp.
- cpAbruptChange: Numeric vector of length prior$trendMaxKnotNum; the estimated magnitude of the jump in the trend at each detected changepoint in cp.
- Y: Numeric vector of length N; the estimated trend component (Bayesian model average over all sampled trends).
- SD: Numeric vector of length N; posterior standard deviation of the estimated trend.
- CI: Numeric matrix of dimension N x 2; credible interval for the trend, with lower and upper envelopes.
- slp: Numeric vector of length N; time-varying slope of the trend component.
- slpSD: Numeric vector of length N; posterior standard deviation of the slope.
- slpSgnPosPr: Numeric vector of length N; posterior probability that the slope of the trend is positive at each time point. For example, slpSgnPosPr = 0.80 at a given time means that 80% of sampled trend models have a positive slope there.
- slpSgnZeroPr: Numeric vector of length N; posterior probability that the slope of the trend is effectively zero (flat) at each time point. The probability of a negative slope can be derived as 1 - slpSgnZeroPr - slpSgnPosPr.
- pos_ncp, neg_ncp, pos_ncpPr, neg_ncpPr, pos_cpOccPr, neg_cpOccPr, pos_cp, neg_cp, pos_cpPr, neg_cpPr, pos_cpAbruptChange, neg_cpAbruptChange, pos_cpCI, neg_cpCI: As above, but restricted to trend changepoints with positive jumps (pos_*) or negative jumps (neg_*) in the trend. For example, pos_ncp is the mean number of changepoints at which the trend level jumps up.
- inc_ncp, dec_ncp, inc_ncpPr, dec_ncpPr, inc_cpOccPr, dec_cpOccPr, inc_cp, dec_cp, inc_cpPr, dec_cpPr, inc_cpAbruptChange, dec_cpAbruptChange, inc_cpCI, dec_cpCI: As above, but restricted to changepoints where the *trend slope* increases (inc_*) or decreases (dec_*). For example, if the trend slope changes from 0.4 to 2.5 at a changepoint, that changepoint is counted toward inc_ncp.

season

season is a list analogous to trend, but for the seasonal/periodic component (when present):

- ncp: Mean number of seasonal changepoints. Because individual models sampled by BEAST can have varying numbers of changepoints, several alternative summaries (ncp_mode, ncp_median, ncp_pct90) are also

provided. For example, if `mcmc$samples = 10` and the numbers of changepoints across models are `c(0, 2, 4, 1, 1, 2, 7, 6, 6, 1)`, then the mean number of changepoints (ncp) is `3.1`, the median `2.5`, the mode `1`, and the 90th percentile (`ncp_pct90`) `6.5`.

- `ncp_mode`: Mode of the number of seasonal changepoints. See the above for explanations.
- `ncp_median`: Median of the number of seasonal changepoints. See the above for explanations.
- `ncp_pct90`: 90th percentile of the number of seasonal changepoints. See the above for explanations.
- `ncpPr`: Numeric vector of length `prior$seasonMaxKnotNum + 1`; probability distribution over the number of seasonal changepoints in `0:prior$seasonMaxKnotNum`. For example, `ncpPr[1]` is the probability of having no seasonal changepoint; `ncpPr[i]` is the probability of having `i - 1` changepoints.
- `cpOccPr`: Numeric vector of length `N`; seasonal changepoint occurrence probability over time. Plotting `cpOccPr` yields a continuous probability-of-being-a-changepoint curve. A peak with a high value at a single time step does not necessarily imply a higher *overall* probability of a changepoint in a neighborhood, compared to a broader peak with a lower maximum but higher total probability. Specifically, a higher peak indicates a higher chance of being a changepoint only at that particular SINGLE point in time and does not necessarily mean a higher chance of observing a changepoint AROUND that time. For example, a window of cpOccPr values `c(0,0,0.5,0,0)` (i.e., the peak prob is 0.5 and the summed probability over the window is 0.5) is less likely to be a changepoint compared to another window `c(0.1,0.2,0.21,0.2,0.1)` where the peak of 0.21 is lower but the summed probablity of 0.71 is larger.
- `order`: Numeric vector of length `N`; average harmonic order needed to approximate the seasonal component. Because this is a posterior average, it need not be an integer
- `cp`: Numeric vector (up to length `prior$seasonMaxKnotNum`); estimated seasonal changepoint locations. These are obtained by smoothing `cpOccPr` with a window of size `prior$seasonMinSepDist` and then selecting up to `prior$seasonMaxKnotNum` peaks in the smoothed curve. Some entries may be `NaN` if the number of detected changepoints is fewer than `seasonMaxKnotNum`. Not all listed changepoints should be treated as "true" changepoints; some may be false positives. Check the associated posterior probabiliites (given in the next field `season$cpPr` for the confidence levels.
- `cpPr`: Numeric vector of length `prior$seasonMaxKnotNum`; the posterior probabilities associated with each changepoint in `cp`. Trailing entries are `NaN` if fewer than `prior$seasonMaxKnotNum` changepoints are identified.
- `cpCI`: Numeric matrix of dimension `prior$seasonMaxKnotNum x 2`; credible intervals for seasonal changepoint locations.
- `cpAbruptChange`: Numeric vector of length `prior$seasonMaxKnotNum`; magnitude of jumps in the seasonal curve at each changepoint.
- `Y`: Numeric vector of length `N`; estimated seasonal component (Bayesian model average).

- SD: Numeric vector of length N; posterior standard deviation of the seasonal component.
- CI: Numeric matrix of dimension N x 2; credible interval for the seasonal component.
- amp: Numeric vector of length N; time-varying amplitude of the seasonality.
- ampSD: Numeric vector of length N; posterior standard deviation of the amplitude.
- pos_ncp, neg_ncp, pos_ncpPr, neg_ncpPr, pos_cpOccPr, neg_cpOccPr, pos_cp, neg_cp, pos_cpPr, neg_cpPr, pos_cpAbruptChange, neg_cpAbruptChange, pos_cpCI, neg_cpCI: Seasonal analogues of the trend outputs with the same names, but restricted to seasonal changepoints with positive (pos_*) or negative (neg_*) jumps in the seasonal component. For example, pos_ncp refers to the average number of seasonal changepoints that jump up (i.e., positively) in the seasonal curve.

outlier — outlier is a list analogous to trend or season, but for the outlier component (present only if setting hasOutlier=TRUE)

References

1. Zhao, K., Wulder, M.A., Hu, T., Bright, R., Wu, Q., Qin, H., Li, Y., Toman, E., Mallick, B., Zhang, X. and Brown, M., 2019. Detecting change-point, trend, and seasonality in satellite time series data to track abrupt changes and nonlinear dynamics: A Bayesian ensemble algorithm. Remote Sensing of Environment, 232, p.111181 (the beast algorithm paper).
2. Zhao, K., Valle, D., Popescu, S., Zhang, X. and Mallick, B., 2013. Hyperspectral remote sensing of plant biochemistry using Bayesian model averaging with variable and band selection. Remote Sensing of Environment, 132, pp.102-119 (the Bayesian MCMC scheme used in beast).
3. Hu, T., Toman, E.M., Chen, G., Shao, G., Zhou, Y., Li, Y., Zhao, K. and Feng, Y., 2021. Mapping fine-scale human disturbances in a working landscape with Landsat time series on Google Earth Engine. ISPRS Journal of Photogrammetry and Remote Sensing, 176, pp.250-261(a beast application paper).

See Also

beast, beast.irreg, minesweeper, tetris, geeLandsat

Examples

```
#------------------------------- NOTE ---------------------------------------------#
# beast123() is an all-inclusive function that unifies the functionality of beast()
# and beast.irreg(). It can handle single, multiple, or 3D stacked time series, either
# regular or irregular. It allows customization via four list arguments:
#   metadata -- additional information about the input Y
#   prior    -- prior parameters for the BEAST model
#   mcmc     -- MCMC settings
#   extra    -- miscellaneous options controlling output and parallel computation
#
# Although functionally similar to beast() and beast.irreg(), beast123() is designed
# to efficiently handle multiple time series concurrently (e.g., stacked satellite
```

```
# images) via internal multithreading. When processing stacks of raster layers,
# DO NOT iterate pixel-by-pixel using beast() or beast.irreg() wrapped by external
# parallel tools (e.g., doParallel, foreach). Instead, use beast123(), which
# implements parallelism internally.

#------------------------------ Example 1: seasonal time series ----------------------#
# Yellowstone is a half-monthly NDVI time series of length 774 for a site in Yellowstone
# starting from July 1-15, 1981 (start ~ c(1981, 7, 7); 24 observations per year).

library(Rbeast)
data(Yellowstone)
plot(Yellowstone)

# Below, the four optional arguments are omitted, so default values are used.
# By default, Y is assumed to be regular with a seasonal component. The actual
# default values are printed at the start of the run and can be used as templates
# for customization.

  o <- beast123(Yellowstone)
  plot(o)


#------------------------------ Example 2: trend-only series -------------------------#
# Nile is an annual river flow time series (no periodic variation). Set season = "none"
# to indicate trend-only analysis. Default values are used for other options.
# Unlike beast(), beast123() does NOT use the time attributes of a 'ts' object.
# For example, Nile is treated as numeric data only with the default times
# 1:length(Nile); its (start=1871, end=1970, freq=1) attributes are ignored
# unless specified through 'metadata'.


  o <- beast123(Nile, season = "none")
  o <- beast123(Nile, extra = list(dumpInputData = TRUE, quiet = TRUE),
                season = "none")
  o <- beast123(Nile, metadata = list(startTime = 1871, hasOutlier = TRUE),
                season = "none")
  plot(o)


#------------------------------ Example 3: full API usage ----------------------------#
# Explicitly specify metadata, prior, mcmc, and extra. Among these, 'prior' contains
# the true statistical model parameters; the others configure input, output, and
# computation.

## Not run:

  # metadata: extra information describing the input time series Y.
  metadata <- list()
  # metadata$isRegularOrdered <- TRUE    # no longer used in this version
  metadata$whichDimIsTime      <- 1      # which dimension is time for 2D/3D inputs
  metadata$startTime           <- c(1981, 7, 7)
  # alternatively: metadata$startTime <- 1981.5137
  #                metadata$startTime <- as.Date("1981-07-07")
```

```
metadata$deltaTime           <- 1/24   # half-monthly: 24 obs/year
metadata$period              <- 1.0    # 1-year period: 24 * (1/24) = 1
metadata$omissionValue       <- NaN    # NaNs are omitted by default
metadata$maxMissingRateAllowed <- 0.75 # skip series with > 75
metadata$deseasonalize       <- FALSE  # do not pre-remove global seasonality
metadata$detrend             <- FALSE  # do not pre-remove global trend

# prior: only true model parameters specifying the Bayesian formulation
prior <- list()
prior$seasonMinOrder    <- 1     # min harmonic order allowed to fit seasonal cmpnt
prior$seasonMaxOrder    <- 5     # max harmonic order allowed to fit seasonal cmpnt
prior$seasonMinKnotNum  <- 0     # min number of changepnts in seasonal cmpnt
prior$seasonMaxKnotNum  <- 3     # max number of changepnts in seasonal cmpnt
prior$seasonMinSepDist  <- 10    # min inter-chngpts separation for seasonal cmpnt
prior$trendMinOrder     <- 0     # min polynomial order allowed to fit trend cmpnt
prior$trendMaxOrder     <- 1     # max polynomial order allowed to fit trend cmpnt
prior$trendMinKnotNum   <- 0     # min number of changepnts in trend cmpnt
prior$trendMaxKnotNum   <- 15    # max number of changepnts in trend cmpnt
prior$trendMinSepDist   <- 5     # min inter-chngpts separation for trend cmpnt
prior$precValue         <- 10.0  # Initial value of precision parameter(no need
                                 # to change it unless for precPrioType='const')
prior$precPriorType   <- "uniform"  # "constant", "uniform", "componentwise", "orderwise"

# mcmc: settings for the MCMC sampler
mcmc <- list()
mcmc$seed                     <- 9543434  # arbitray seed for random number generator
mcmc$samples                  <- 3000     # samples collected per chain
mcmc$thinningFactor           <- 3        # take every 3rd sample and discard others
mcmc$burnin                   <- 150      # discard the initial 150 samples per chain
mcmc$chainNumber              <- 3        # number of chains
mcmc$maxMoveStepSize         <- 4       # max random jump step when proposing new chngpts
mcmc$trendResamplingOrderProb  <- 0.10    # prob of choosing to resample polynomial order
mcmc$seasonResamplingOrderProb <- 0.10     # prob of choosing to resample harmonic order
mcmc$credIntervalAlphaLevel   <- 0.95    # the significance level for credible interval

# extra: output and computation options
extra <- list()
extra$dumpInputData        <- FALSE  # If true, a copy of input time series is outputted
extra$whichOutputDimIsTime  <- 1     # For 2D or 3D inputs, which dim of the output refers
                                  # to time? Ignored if the input is a single time series
extra$computeCredible     <- FALSE  # If true, compute CI: computing CI is time-intensive.
extra$fastCIComputation    <- TRUE   # If true, a faster way is used to get CI, but it is
                                  # still time-intensive. That is why the function beast()
                                  #       # is slow because it always compute CI.
extra$computeSeasonOrder   <- FALSE  # If true, dump the estimated harmonic order over time
extra$computeTrendOrder    <- FALSE  # If true, dump the estimated polynomial order over time
extra$computeSeasonChngpt  <- TRUE   # If true, get the most likely locations of s chgnpts
extra$computeTrendChngpt   <- TRUE   # If true, get the most likely locations of t chgnpts
extra$computeSeasonAmp     <- FALSE  # If true, get time-varying amplitude of seasonality
 extra$computeTrendSlope     <- FALSE  # If true, get time-varying slope of trend
extra$tallyPosNegSeasonJump <- FALSE  # If true, get those changpts with +/- jumps in season
extra$tallyPosNegTrendJump  <- FALSE  # If true, get those changpts with +/- jumps in trend
 extra$tallyIncDecTrendJump  <- FALSE  # If true, get those changpts with increasing/
```

```
                                    # decreasing trend slopes
  extra$printProgress         <- TRUE
  extra$printParameter        <- TRUE
  extra$quiet                 <- FALSE  # if TRUE, print warning messages, if any
  extra$consoleWidth        <- 0     # If 0, the console width is from the current console
  extra$numThreadsPerCPU     <- 2     # 'numThreadsPerCPU' and 'numParThreads' are used to
  extra$numParThreads        <- 0     #   configure multithreading runs; they're used only
                                  #   if Y has multiple time series (e.g.,stacked images)

  o <- beast123(Yellowstone, metadata, prior, mcmc, extra, season = "harmonic")
  plot(o)

## End(Not run)

#------------------------------ Example 4: irregular time series --------------------#
# ohio is a data frame of Landsat NDVI observations at unevenly spaced times.


## Not run:

  data(ohio)
  str(ohio)

  ###############################################################################
  # Individual times are provided as fractional years via ohio$time

  metadata <- list()
  metadata$time      <- ohio$time  # Must supply individual times for irregular inputs
  metadata$deltaTime <- 1/12      # Must supply the desired time interval for aggregation
  metadata$period    <- 1.0
  o <- beast123(ohio$ndvi, metadata)   # defaults used for missing parameters

  ###############################################################################
  # Another accepted time format: Ohio dates as Date objects
  class(ohio$rdate)

  metadata <- list()
  metadata$time      <- ohio$rdate # Must supply individual times for irregular inputs
  metadata$deltaTime <- 1/12      # Must supply the desired time interval for aggregation
  o <- beast123(ohio$ndvi, metadata)  # Default values used for those missing parameters

  ###############################################################################
  # Another accepted time format: separate Y, M, D columns
  ohio$Y      # Year
  ohio$M      # Month
  ohio$D      # Day

  metadata <- list()
  metadata$time$year   <- ohio$Y   # Must supply individual times for irregular inputs
  metadata$time$month  <- ohio$M
  metadata$time$day    <- ohio$D
  metadata$deltaTime   <- 1/12    # Must supply the desired time interval for aggregation
  o <- beast123(ohio$ndvi, metadata) # Default values used for those missing parameters
```

```
  ##############################################################################
  # Another accepted time format: year and DOY
  ohio$Y        # Year
  ohio$doy      # Day of year

  metadata <- list()
  metadata$time$year  <- ohio$Y   # Must supply individual times for irregular inputs
  metadata$time$doy   <- ohio$doy
  metadata$deltaTime  <- 1/12     # Must supply the desired time interval for aggregation
  o <- beast123(ohio$ndvi, metadata) # Default values used for those missing parameters


  ##############################################################################
  # Another accepted time format: date strings with custom strfmt
  ohio$datestr1

  metadata <- list()
  metadata$time$datestr  <- ohio$datestr1
  metadata$time$strfmt   <- "????yyyy?mm?dd"
  metadata$deltaTime     <- 1/12
  o <- beast123(ohio$ndvi, metadata)

 ##############################################################################
 # Another accepted time format: date strings with custom strfmt
  ohio$datestr2

  metadata <- list()
  metadata$deltaTime     <- 1/12
  metadata$time$datestr  <- ohio$datestr2
  metadata$time$strfmt   <- "????yyyydoy????"
  o <- beast123(ohio$ndvi, metadata)

 ##############################################################################
 # Another accepted time format: date strings with custom strfmt
  ohio$datestr3

  metadata <- list()
  metadata$deltaTime     <- 1/12
  metadata$time$datestr  <- ohio$datestr3
  metadata$time$strfmt   <- "Y,,M/D"
  o <- beast123(ohio$ndvi, metadata)

## End(Not run)


#------------------ Example 5: multiple time series (matrix input) -------------------#
# simdata is a 2D matrix of dimension 300 x 3, representing 3 time series of length 300.

## Not run:
  data(simdata)  # dim of simdata: 300 x 3 (time x num_of_time_series)
  dim(simdata)   # the first dimenion refer to time (i.e, 300)
```

```
  metadata <- list()
  metadata$whichDimIsTime <- 1  # first dimension is time
  metadata$period         <- 24 # assume startTime = 1, deltaTime = 1 by default

  extra <- list()
  extra$whichOutputDimIsTime <- 2  # Which dim of the output arrays refers to  time?

  o <- beast123(simdata, metadata, extra = extra)

  # Lists of inputs args can also be provided inline:
  o <- beast123(simdata,
                metadata = list(whichDimIsTime = 1, period = 24),
                extra    = list(whichOutputDimIsTime = 2))

  # Field names can be shortened, provided no ambiguity arises:
  o <- beast123(simdata,
                metadata = list(whichDim = 1, per = 24),
                extra    = list(whichOut = 2))

  #------------------ Example 5b: transposed simdata ---------------------------------#
  simdata1 <- t(simdata)   # 3 (series) x 300 (time)
                           # dim of simdata1: 3 x 300 (num of ts  x time )

  metadata <- list()
  metadata$whichDimIsTime <- 2  # time is second dimension
  metadata$period         <- 24 # assume startTime = 1, deltaTime = 1 by default
  o <- beast123(simdata1, metadata)

  o <- beast123(simdata1, metadata = list(whichDim = 2, per = 24))

## End(Not run)

#------------------ Example 6: 3D stacked image time series -------------------------#
# imagestack$ndvi is a 3D array of size 12 x 9 x 1066 (row x col x time). Each pixel
# corresponds to a time series of length 1066.

## Not run:
  data(imagestack)
  dim(imagestack$ndvi)   # 12 x 9 x 1066
  imagestack$datestr     # A character vector of 1066 date strings

  metadata <- list()
  metadata$whichDimIsTime <- 3  # Which dim of the input refer to time for 3D inputs?
                                # 1066 is the ts length, so dim is set to '3' here.
                                # In this example, this arg is not needed because
                                # the time$datestr can also help to match and pick up
                                # the right time dimesion of imagestack$ndvi.
  metadata$time$datestr   <- imagestack$datestr
  metadata$time$strfmt    <- "LT05_018032_20080311.yyyy-mm-dd"
  metadata$deltaTime      <- 1/12   # aggregate to monthly (i.e., 1/12 yr)
  metadata$period         <- 1.0    # 1-year period

  extra <- list()
```

```
  extra$dumpInputData    <- TRUE   # Dump a copy of aggregated input ts
  extra$numThreadsPerCPU <- 2      # Each cpu core will be assigned 2 threads
 extra$numParThreads   <- 0    # If 0, total_num_threads=numThreadsPerCPU*num_of_cpu_core
                                  # if >0, used to specify the total number of threads

  # Default values for missing parameters
  o <- beast123(imagestack$ndvi, metadata = metadata, extra = extra)

  print(o, c(5, 3))     # result for pixel at row 5, col 3
  plot(o,  c(5, 3))     # plot result for pixel at row 5, col 3
  image(o$trend$ncp)    # map number of trend changepoints over space

## End(Not run)

#------------------ Example 7: GeoTIFF stacks via raster package --------------------#
# Handle 3D stacked images from a series of NDVI GeoTIFF files (e.g., ndvi.zip with
# 437 NDVI TIFF files of dimension 12 x 9). Example code is available at:
#   https://github.com/zhaokg/Rbeast/blob/master/R/beast123_raster_example.txt
```

CNAchrom11 *DNA copy number alteration data in array-based CGH data for Chromesome 11*

Description

CNAchrom11 is a vector of the log2 intensity ratios for cell line GM03576 for Chromosome 11, obtained from Snijders et al. (2001).

Usage

```
data(CNAchrom11)
```

Source

Snijders et al. (2001), Assembly of microarrays for genome-wide measurement of DNA copy number, Nature Genetics, 29, 263-264 (http://www.nature.com/ng/journal/v29/n3/full/ng754.html).

References

1. Zhao, K., Wulder, M.A., Hu, T., Bright, R., Wu, Q., Qin, H., Li, Y., Toman, E., Mallick, B., Zhang, X. and Brown, M., 2019. Detecting change-point, trend, and seasonality in satellite time series data to track abrupt changes and nonlinear dynamics: A Bayesian ensemble algorithm. Remote Sensing of Environment, 232, p.111181 (the beast algorithm paper).
2. Zhao, K., Valle, D., Popescu, S., Zhang, X. and Mallick, B., 2013. Hyperspectral remote sensing of plant biochemistry using Bayesian model averaging with variable and band selection. Remote Sensing of Environment, 132, pp.102-119 (the Bayesian MCMC scheme used in beast).

3. Hu, T., Toman, E.M., Chen, G., Shao, G., Zhou, Y., Li, Y., Zhao, K. and Feng, Y., 2021. Mapping fine-scale human disturbances in a working landscape with Landsat time series on Google Earth Engine. ISPRS Journal of Photogrammetry and Remote Sensing, 176, pp.250-261(a beast application paper).

Examples

```
library(Rbeast)
data(CNAchrom11)


o = beast(CNAchrom11, season='none') # no periodic component
plot(o)
```

`covid19` *Daily confirmed COVID19 cases and deaths in the world*

Description

`covid19` is a data frame consisting of daily confirmed COVID19 cases and deaths in the world from Jan 22, 2020 to Dec 16, 2021.

Usage

```
data(covid19)
```

Source

https://ourworldindata.org/grapher/daily-covid-cases-deaths?country=~OWID_WRL (last accessed on Dec 16, 2021)

References

1. Zhao, K., Wulder, M.A., Hu, T., Bright, R., Wu, Q., Qin, H., Li, Y., Toman, E., Mallick, B., Zhang, X. and Brown, M., 2019. Detecting change-point, trend, and seasonality in satellite time series data to track abrupt changes and nonlinear dynamics: A Bayesian ensemble algorithm. Remote Sensing of Environment, 232, p.111181 (the beast algorithm paper).
2. Zhao, K., Valle, D., Popescu, S., Zhang, X. and Mallick, B., 2013. Hyperspectral remote sensing of plant biochemistry using Bayesian model averaging with variable and band selection. Remote Sensing of Environment, 132, pp.102-119 (the Bayesian MCMC scheme used in beast).
3. Hu, T., Toman, E.M., Chen, G., Shao, G., Zhou, Y., Li, Y., Zhao, K. and Feng, Y., 2021. Mapping fine-scale human disturbances in a working landscape with Landsat time series on Google Earth Engine. ISPRS Journal of Photogrammetry and Remote Sensing, 176, pp.250-261(a beast application paper).

Examples

```
library(Rbeast)
data(covid19)
plot(covid19$date, covid19$newcases, type='l')

## Not run:

# Apply a square root-transformation
newcases = sqrt( covid19$newcases )

# This time series varies periodically every 7 days. 7 days can't be precisely
# represented in the unit of year bcz some years has 365 days and others has 366.
# BEAST can hanlde this in two ways.


#(1) Use the date number as the time unit--the num of days lapsed since 1970-01-01.

 datenum  = as.numeric(covid19$date)
 o        = beast(newcases, start=min(datenum), deltat=1, period=7)
 o$time   = as.Date(o$time, origin='1970-01-01') # Convert from integers to Date.
 plot(o)

#(2) Use strings to explicitly specify deltat and period with a unit.

 startdate = covid19$date[1]
 o         = beast(newcases, start=startdate, deltat='1day', period='7days')
 plot(o)


## End(Not run)
```

`geeLandsat` *Landsat reflectance and NDVI time series from Google Earth Engine*

Description

Get Landsat reflectance and NDVI time series from Google Earth Engine given longitude and latitude

Usage

```
geeLandsat(lon=NA, lat=NA, radius=100, stat='mean',timeout=700)
```

Arguments

`lon`	numeric within [-180,180]
`lat`	numeric within [-90, 90]

radius	a positive number (<=500 meters); the radius of a buffer around the given latitude and longitude for aggregation. If radius=0, the single pixel at the lat and lon will be retrieved
stat	character; if radius>0, used to specify the spatial aggregation method for pixels in the buffer. Possible values are 'mean','min','max', or 'median'.
timeout	integer; the seconds elapsed to wait for connection timeout. See the note for an explanation.

Value

a data.frame object consisting of dates, sensor type, reflectances, and NDVI for the requested location. It contains only valid and clear-sky values as obtained by referring to the standard clouds flags.

Note

As a poor man's scheme to interact with Google Earth Engine, geeLandsat should be used only for occasional retrieval of Landsat time series at a few sites, NOT for batch downloading for thousands of sites in a R loop. This procedure is provided to get example time series for testing BEAST. Behind the scene, this function calls to a free Python-based server using my own GEE credential. Normally it takes several seconds to retrieve one time series, but as a free cloud service, the Python server only offers 100 seconds of free CPU time per day, with throttling applied. So it may take up to a few mins to get a time series on your end. It may fail due to connection timeout; if so, give it a few tries. If you need to retrieve data for thousands or millions of sites, please contact the author.

References

1. Zhao, K., Wulder, M.A., Hu, T., Bright, R., Wu, Q., Qin, H., Li, Y., Toman, E., Mallick, B., Zhang, X. and Brown, M., 2019. Detecting change-point, trend, and seasonality in satellite time series data to track abrupt changes and nonlinear dynamics: A Bayesian ensemble algorithm. Remote Sensing of Environment, 232, p.111181 (the beast algorithm paper).
2. Zhao, K., Valle, D., Popescu, S., Zhang, X. and Mallick, B., 2013. Hyperspectral remote sensing of plant biochemistry using Bayesian model averaging with variable and band selection. Remote Sensing of Environment, 132, pp.102-119 (the Bayesian MCMC scheme used in beast).
3. Hu, T., Toman, E.M., Chen, G., Shao, G., Zhou, Y., Li, Y., Zhao, K. and Feng, Y., 2021. Mapping fine-scale human disturbances in a working landscape with Landsat time series on Google Earth Engine. ISPRS Journal of Photogrammetry and Remote Sensing, 176, pp.250-261(a beast application paper).

See Also

beast, beast.irreg, beast123, minesweeper, tetris

Examples

```
 library(Rbeast)
## Not run:
```

```
df = geeLandsat(lon=-80.983877,lat= 40.476882) #if it fails, try a few more times before giving up
print(df)

## End(Not run)
```

---

googletrend_beach *A monthly Google Trend time series of the US search interest in the word "beach"*

Description

googletrend_beach is a ts object comprising monthly search interest in "beach" from the United States, as reported from Google Trends. Sudden changes in the search trend are attributed to extreme weather events or the covid19 outbreak

Usage

```
data(googletrend_beach)
```

Source

https://trends.google.com/trends/explore?date=all&geo=US&q=beach

References

1. Zhao, K., Wulder, M.A., Hu, T., Bright, R., Wu, Q., Qin, H., Li, Y., Toman, E., Mallick, B., Zhang, X. and Brown, M., 2019. Detecting change-point, trend, and seasonality in satellite time series data to track abrupt changes and nonlinear dynamics: A Bayesian ensemble algorithm. Remote Sensing of Environment, 232, p.111181 (the beast algorithm paper).
2. Zhao, K., Valle, D., Popescu, S., Zhang, X. and Mallick, B., 2013. Hyperspectral remote sensing of plant biochemistry using Bayesian model averaging with variable and band selection. Remote Sensing of Environment, 132, pp.102-119 (the Bayesian MCMC scheme used in beast).
3. Hu, T., Toman, E.M., Chen, G., Shao, G., Zhou, Y., Li, Y., Zhao, K. and Feng, Y., 2021. Mapping fine-scale human disturbances in a working landscape with Landsat time series on Google Earth Engine. ISPRS Journal of Photogrammetry and Remote Sensing, 176, pp.250-261(a beast application paper).

Examples

```
library(Rbeast)
data(googletrend_beach) # A monthly ts starting from Jan 2004

o = beast(googletrend_beach )
plot(o)
```

imagestack *Decades of Landsat NDVI time series over a small area in Ohio*

Description

imagestack is a LIST containing Landsat-derived NDVI image chips at an Ohio site

Usage

```
data(imagestack)
```

Source

Landsat images courtesy of the U.S. Geological Survey

References

1. Zhao, K., Wulder, M.A., Hu, T., Bright, R., Wu, Q., Qin, H., Li, Y., Toman, E., Mallick, B., Zhang, X. and Brown, M., 2019. Detecting change-point, trend, and seasonality in satellite time series data to track abrupt changes and nonlinear dynamics: A Bayesian ensemble algorithm. Remote Sensing of Environment, 232, p.111181 (the beast algorithm paper).
2. Zhao, K., Valle, D., Popescu, S., Zhang, X. and Mallick, B., 2013. Hyperspectral remote sensing of plant biochemistry using Bayesian model averaging with variable and band selection. Remote Sensing of Environment, 132, pp.102-119 (the Bayesian MCMC scheme used in beast).
3. Hu, T., Toman, E.M., Chen, G., Shao, G., Zhou, Y., Li, Y., Zhao, K. and Feng, Y., 2021. Mapping fine-scale human disturbances in a working landscape with Landsat time series on Google Earth Engine. ISPRS Journal of Photogrammetry and Remote Sensing, 176, pp.250-261(a beast application paper).

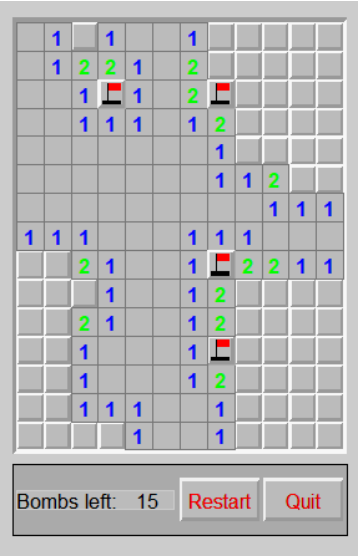
Examples

```
data(imagestack)
imagestack$datestr # A string vector containg the observation dates of individual ndvi images
## Not run:
imagestack$ndvi    # NDVI images collected over the past several deccades

## End(Not run)
plot(imagestack$ndvi[3,4,],type='l') # Plot the raw data at a pixel
```

`minesweeper` *The Minesweeper game in R*

Description

A poor man's implementation of the minesweeper game in R. Yes, you are right: it has nothing to do with time series decomposition, changepoint detection, and time series segmentation. Its only remote connection to Rbeast is that this is a practice script I wrote to learn R graphics for implementing Rbeast.



Usage

```
minesweeper(height=15, width=12, prob=0.1)
```

Arguments

`height`	integer; number of rows of the mine grid along the vertical direction.
`width`	integer; number of columns of the mine grid along the horizontal direction.
`prob`	numeric; a fraction between 0 and 1 to specify the probability of mine occurrence in the mine grid.

Value

Instructions:

- LEFT-click to clear a spot.

- RIGHT-click to flag a spot.
- MIDDLE-click(wheel) a cleared and numbered spot to open neighbor spots, if flagged correctly.
- Click Restart for a new game

Note

An interactive graphics window is needed to run this function correctly. So it won't run in RStudio's plot pane. The function will use the x11() or x11(type='Xlib') graphic device to open a pop-up window.

References

1. Zhao, K., Wulder, M.A., Hu, T., Bright, R., Wu, Q., Qin, H., Li, Y., Toman, E., Mallick, B., Zhang, X. and Brown, M., 2019. Detecting change-point, trend, and seasonality in satellite time series data to track abrupt changes and nonlinear dynamics: A Bayesian ensemble algorithm. Remote Sensing of Environment, 232, p.111181 (the beast algorithm paper).
2. Zhao, K., Valle, D., Popescu, S., Zhang, X. and Mallick, B., 2013. Hyperspectral remote sensing of plant biochemistry using Bayesian model averaging with variable and band selection. Remote Sensing of Environment, 132, pp.102-119 (the Bayesian MCMC scheme used in beast).
3. Hu, T., Toman, E.M., Chen, G., Shao, G., Zhou, Y., Li, Y., Zhao, K. and Feng, Y., 2021. Mapping fine-scale human disturbances in a working landscape with Landsat time series on Google Earth Engine. ISPRS Journal of Photogrammetry and Remote Sensing, 176, pp.250-261(a beast application paper).

See Also

beast, beast.irreg, beast123, tetris, geeLandsat

Examples

```
library(Rbeast)

## Not run:
minesweeper()

# A mine field of size 20x25 with rougly a 15
minesweeper(20,25,0.15)

## End(Not run)
```

ohio *An irregular Landsat NDVI time series at an Ohio site*

Description

ohio is a data.frame object comprising decades of Landsat-observed surface reflectances and NDVI at an Ohio site

Usage

```
data(ohio)
```

Source

Landsat images courtesy of the U.S. Geological Survey

References

1. Zhao, K., Wulder, M.A., Hu, T., Bright, R., Wu, Q., Qin, H., Li, Y., Toman, E., Mallick, B., Zhang, X. and Brown, M., 2019. Detecting change-point, trend, and seasonality in satellite time series data to track abrupt changes and nonlinear dynamics: A Bayesian ensemble algorithm. Remote Sensing of Environment, 232, p.111181 (the beast algorithm paper).
2. Zhao, K., Valle, D., Popescu, S., Zhang, X. and Mallick, B., 2013. Hyperspectral remote sensing of plant biochemistry using Bayesian model averaging with variable and band selection. Remote Sensing of Environment, 132, pp.102-119 (the Bayesian MCMC scheme used in beast).
3. Hu, T., Toman, E.M., Chen, G., Shao, G., Zhou, Y., Li, Y., Zhao, K. and Feng, Y., 2021. Mapping fine-scale human disturbances in a working landscape with Landsat time series on Google Earth Engine. ISPRS Journal of Photogrammetry and Remote Sensing, 176, pp.250-261(a beast application paper).

Examples

```
library(Rbeast)
data(ohio) # Landsat surface references and NDVI at a single pixel observed over time
str(ohio)

## Not run:
 # ohio$ndvi is a single irregular time series
 y = ohio$ndvi
 o = beast.irreg(y, time=ohio$time,deltat=1/12)
 plot(o)
 print(o)

 # ohio also contains irregular time series of individual spectral bands
 # Below, run the multivariate version of the BEAST algorithm to decompose
 # the 5 time series and detect common changepoints altogether
```

```
y = list(ohio$blue, ohio$green, ohio$red, ohio$nir, ohio$swir1);
o = beast.irreg(y, time=ohio$time,deltat=1/12, freq=12)
plot(o)
print(o)

## End(Not run)
```

---

plot.beast *Bayesian changepoint detection and time series decomposition*

Description

Plot the result obtained from the beast function.

Usage

```
## S3 method for class 'beast'
plot(
     x,
     index = 1,
   vars  = c('y','s','scp','sorder','t','tcp','torder','slpsgn','o','ocp','error'),
     col         = NULL,
     main        = "BEAST decomposition and changepoint detection",
     xlab        = 'Time',
     ylab        = NULL,
     cex.main    = 1,
     cex.lab     = 1,
     relative.heights = NULL,
     interactive = FALSE,
     ncpStat     = c('median','mode','mean','pct90','max'),
     ...
   )
```

Arguments

x	a "beast" object returned by beast,beast.irreg, or beast123. It may contain one or many time series.
index	an integer (default to 1) or a vector of two integers to specify the index of the time series to plot if x contains results for multiple time series. index is always 1 if x has 1 time series. If x is returned by beast123 with a 2D input,index should be a single integer. If x is from beast123 applied to 3D arrays of time series (e.g., stacked satellite images), index can be a linear index or two subscripts to specify the row and column of the pixel/grid.
vars	a vector of strings indicating the elements or variables of x to plot. Possible vars strings include 'y' (season plus trend), 's' (season component), 't' (trend

component), 'o' (outliers), 'scp', 'tcp', 'ocp' (occurrence probability of seasonal/trend/outlier changepoint), 'sorder' (seasonal harmonic order), 'torder' (trend polynomial order), 'samp' (amplitude of seasonality), 'tslp' (slope of trend), 'slpsgn' (probabilities of the slope being positive, zero, and negative) and 'error' (remainder).

`relative.heights`
: a numeric vector of the same length as that of `vars` to specify the relative heights of subplots of individual variables in `vars`.

`col`
: a string vector of the same length as that of `vars` to specify the colors of individual subplots associated with `vars`.

`main`
: a string; the main title.

`xlab`
: a string: the x axis title.

`ylab`
: a string vector of the same length as that of `vars` to specify the y axis names of individual subplots associated with `vars`

`cex.main`
: cex for the main title

`cex.lab`
: cex for the axis title

`interactive`
: a bool scalar. If TRUE, an interactive GUI is used for examining individual elements of `x`.

`ncpStat`
: character. A string to specify which statistic is used for the Number of Change-Point (ncp). Five values are possible: 'mean', 'mode', 'median','pct90', and 'max'; the default is 'median'. Individual models sampled by BEAST has a varying dimension (e.g., number of changepoints or knots). For example, if mcmc$samples=10, the numbers of changepoints for the 10 sampled models are assumed to be c(0, 2, 4, 1, 1, 2, 7, 6, 6, 1). The mean ncp will be 3.1 (rounded to 3), the median is 2.5 (2), the mode is 1, and the maximum is 7. The 'max' option plots all the changepoints recorded in `out$trend$cp`, `out$season$cp`, or `out$outlier$cp`; many of these changepoints are bound to be false positives, so do not treat all of them as actual changepoints.

`...`
: additional parameters to be implemented.

Value

This function creates various plots to demonstrate the results of a beast decomposition. .

References

1. Zhao, K., Wulder, M.A., Hu, T., Bright, R., Wu, Q., Qin, H., Li, Y., Toman, E., Mallick, B., Zhang, X. and Brown, M., 2019. Detecting change-point, trend, and seasonality in satellite time series data to track abrupt changes and nonlinear dynamics: A Bayesian ensemble algorithm. Remote Sensing of Environment, 232, p.111181 (the beast algorithm paper).

2. Zhao, K., Valle, D., Popescu, S., Zhang, X. and Mallick, B., 2013. Hyperspectral remote sensing of plant biochemistry using Bayesian model averaging with variable and band selection. Remote Sensing of Environment, 132, pp.102-119 (the Bayesian MCMC scheme used in beast).

3. Hu, T., Toman, E.M., Chen, G., Shao, G., Zhou, Y., Li, Y., Zhao, K. and Feng, Y., 2021. Mapping fine-scale human disturbances in a working landscape with Landsat time series on Google Earth Engine. ISPRS Journal of Photogrammetry and Remote Sensing, 176, pp.250-261(a beast application paper).

See Also

beast, beast.irreg, beast123, plot.beast,minesweeper, tetris, geeLandsat

Examples

```
 library(Rbeast)
 data(simdata)
## Not run:
 result=beast123(simdata, metadata=list(whichDimIsTime=1))
 plot(result,1)
 plot(result,2)

## End(Not run)
```

---

print.beast *Bayesian changepoint detection and time series decomposition*

Description

Summarize and print the results obtained from the BEAST time series decomposition and segmentation.

Usage

```
## S3 method for class 'beast'
print(
     x,
     index = 1,
     ...
 )
```

Arguments

x	a "beast" object returned by beast, beast.irreg, or beast123. It may contain one or many time series.
index	an integer (default to 1) or a vector of two integers to specify the index of the time series to print if x contains results for multiple time series. If x has 1 time series, index should be always 1. If x is returned by beast123 applied to a 2D input,index should be a single index. If x is from beast123 applied to 3D arrays of time series (e.g., stacked satellite images), index can be a linear index or two subscripts to specify the row and column of the desired pixel/grid.
...	additional parameters to be implemented.

Value

Print a summary of changepoints detected for the seasonal or trend component.

References

1. Zhao, K., Wulder, M.A., Hu, T., Bright, R., Wu, Q., Qin, H., Li, Y., Toman, E., Mallick, B., Zhang, X. and Brown, M., 2019. Detecting change-point, trend, and seasonality in satellite time series data to track abrupt changes and nonlinear dynamics: A Bayesian ensemble algorithm. Remote Sensing of Environment, 232, p.111181 (the beast algorithm paper).
2. Zhao, K., Valle, D., Popescu, S., Zhang, X. and Mallick, B., 2013. Hyperspectral remote sensing of plant biochemistry using Bayesian model averaging with variable and band selection. Remote Sensing of Environment, 132, pp.102-119 (the Bayesian MCMC scheme used in beast).
3. Hu, T., Toman, E.M., Chen, G., Shao, G., Zhou, Y., Li, Y., Zhao, K. and Feng, Y., 2021. Mapping fine-scale human disturbances in a working landscape with Landsat time series on Google Earth Engine. ISPRS Journal of Photogrammetry and Remote Sensing, 176, pp.250-261(a beast application paper).

See Also

`beast`, `beast.irreg`, `beast123`, `minesweeper`, `tetris`, `geeLandsat`

Examples

```
 library(Rbeast)
 data(simdata)

## Not run:
#out=beast123(simdata) #Error: whichDimIsTime has to be specified to
                       # tell which dim of simdata refers to time.
                       # See below.
 out=beast123(simdata, metadata=list(whichDimIsTime=1))
 print(out, 1)
 print(out, 2)

## End(Not run)
```

`simdata` *Simulated time series to test BEAST*

Description

`simdata` is a 300 x 3 matrix, consisting three time series of length 300. Currently, the three time series are the same. It is used to illustrate BEAST can handle multiple time series at a single function call. of BEAST.

Usage

```
data(simdata)
```

Source

Rbeast v0.9.2

References

1. Zhao, K., Wulder, M.A., Hu, T., Bright, R., Wu, Q., Qin, H., Li, Y., Toman, E., Mallick, B., Zhang, X. and Brown, M., 2019. Detecting change-point, trend, and seasonality in satellite time series data to track abrupt changes and nonlinear dynamics: A Bayesian ensemble algorithm. Remote Sensing of Environment, 232, p.111181 (the beast algorithm paper).
2. Zhao, K., Valle, D., Popescu, S., Zhang, X. and Mallick, B., 2013. Hyperspectral remote sensing of plant biochemistry using Bayesian model averaging with variable and band selection. Remote Sensing of Environment, 132, pp.102-119 (the Bayesian MCMC scheme used in beast).
3. Hu, T., Toman, E.M., Chen, G., Shao, G., Zhou, Y., Li, Y., Zhao, K. and Feng, Y., 2021. Mapping fine-scale human disturbances in a working landscape with Landsat time series on Google Earth Engine. ISPRS Journal of Photogrammetry and Remote Sensing, 176, pp.250-261(a beast application paper).

Examples

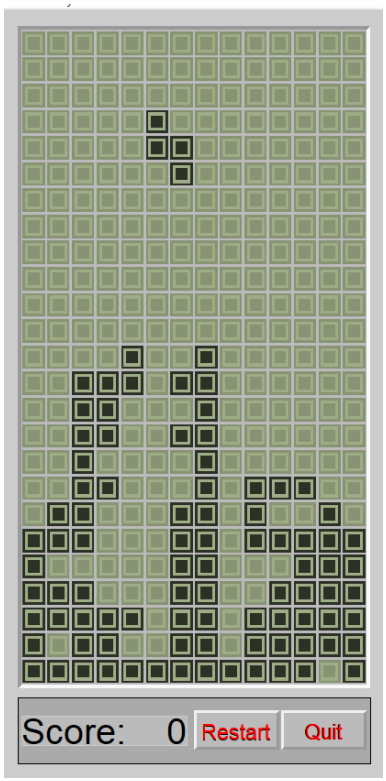
```
library(Rbeast)
data(simdata)
plot(simdata[,1],type='l')

## Not run:
#out=beast123(simdata) # Error: whichDimIsTime has to be specified. See below
out=beast123(simdata, metadata=list(whichDimIsTime=1))

plot(out,1)
plot(out,2)
plot(out,3)

## End(Not run)
```

---

tetris *The Tetris game in R*

Description

A poor man's implementation of the Tetris game in R. Yes, you are right again: it has nothing to do with time series decomposition, changepoint detection, and time series segmentation. Its only remote connection to Rbeast is that this is a practice script I wrote to learn R graphics for implementing Rbeast.



Usage

```
tetris(height=25, width=14, speed=0.6)
```

Arguments

height	integer; number of rows of the mine grid along the vertical direction.
width	integer; number of columns of the mine grid along the horizontal direction.
speed	numeric; a time interval between 0.05 and 2 seconds, specifying how fast the tetriminos moves down. The smaller, the faster.

Value

Instructions:

- Left arrow to move left.
- Right arrow to move right.
- Up arrow to rotate.
- Down arrow to speed up.
- Space key to sink to the bottom.

Note

This function works only under the Windows OS not Linux or Mac. An interactive graphics window is needed to run this function correctly. So it won't run in RStudio's plot pane. The function will use the x11() or x11(type='Xlib') graphic device to open a pop-up window.

References

1. Zhao, K., Wulder, M.A., Hu, T., Bright, R., Wu, Q., Qin, H., Li, Y., Toman, E., Mallick, B., Zhang, X. and Brown, M., 2019. Detecting change-point, trend, and seasonality in satellite time series data to track abrupt changes and nonlinear dynamics: A Bayesian ensemble algorithm. Remote Sensing of Environment, 232, p.111181 (the beast algorithm paper).
2. Zhao, K., Valle, D., Popescu, S., Zhang, X. and Mallick, B., 2013. Hyperspectral remote sensing of plant biochemistry using Bayesian model averaging with variable and band selection. Remote Sensing of Environment, 132, pp.102-119 (the Bayesian MCMC scheme used in beast).
3. Hu, T., Toman, E.M., Chen, G., Shao, G., Zhou, Y., Li, Y., Zhao, K. and Feng, Y., 2021. Mapping fine-scale human disturbances in a working landscape with Landsat time series on Google Earth Engine. ISPRS Journal of Photogrammetry and Remote Sensing, 176, pp.250-261(a beast application paper).

See Also

beast, beast.irreg, beast123, minesweeper, geeLandsat

Examples

```
library(Rbeast)

## Not run:
tetris()

# A  field of size 20x25 with blocks moving down every 0.1 sec.
tetris(20,25,0.1)

## End(Not run)
```

`tsextract` *Bayesian changepoint detection and time series decomposition*

Description

Extract the result of a single time series from an object of class beast

Usage

```
tsextract( x, index = 1 )
```

Arguments

x	a "beast" object returned by beast, beast.irreg, or beast123. It may contain one or many time series.
index	an integer (default to 1) or a vector of two integers to specify the index of the time series to extract if x contains results for multiple time series. If x has 1 time series, index should be always 1. If x is returned by beast123 applied to a 2D input,index should be a single index. If x is from beast123 applied to 3D arrays of time series (e.g., stacked satellite images), index can be a linear index or two subscripts to specify the row and column of the desired pixel/grid.

Value

A LIST object of the result for the chosen time series, which contains the same field as x.

Note

Use this function only to manually and interactively examine individual times series. If the purpose is to loop through x, the use of direct indexing is much faster. For example, if x is a beast object for a 300x200x1000 3D array (row x col x time), use x$trend$Y[20,40,] to get the fitted trend at the pixel of row 20 and col 40.

References

1. Zhao, K., Wulder, M.A., Hu, T., Bright, R., Wu, Q., Qin, H., Li, Y., Toman, E., Mallick, B., Zhang, X. and Brown, M., 2019. Detecting change-point, trend, and seasonality in satellite time series data to track abrupt changes and nonlinear dynamics: A Bayesian ensemble algorithm. Remote Sensing of Environment, 232, p.111181 (the beast algorithm paper).
2. Zhao, K., Valle, D., Popescu, S., Zhang, X. and Mallick, B., 2013. Hyperspectral remote sensing of plant biochemistry using Bayesian model averaging with variable and band selection. Remote Sensing of Environment, 132, pp.102-119 (the Bayesian MCMC scheme used in beast).
3. Hu, T., Toman, E.M., Chen, G., Shao, G., Zhou, Y., Li, Y., Zhao, K. and Feng, Y., 2021. Mapping fine-scale human disturbances in a working landscape with Landsat time series on Google Earth Engine. ISPRS Journal of Photogrammetry and Remote Sensing, 176, pp.250-261(a beast application paper).

See Also

beast, beast.irreg, beast123, minesweeper, tetris, geeLandsat

Examples

```
library(Rbeast)
data(simdata)


# handle only the 1st ts
out=beast(simdata[,1])


## Not run:
# handle all the ts
out=beast123(simdata, metadata=list(whichDimIsTime=1))

plot(out,1)
plot(out,2)

## End(Not run)
```

`Yellowstone` *30 years' AVHRR NDVI data at a Yellostone site*

Description

`Yellowstone` is a vector comprising 30 years' AVHRR NDVI data at a Yellostone site

Usage

```
data(Yellowstone)
```

Source

Rbeast v0.9.2

References

1. Zhao, K., Wulder, M.A., Hu, T., Bright, R., Wu, Q., Qin, H., Li, Y., Toman, E., Mallick, B., Zhang, X. and Brown, M., 2019. Detecting change-point, trend, and seasonality in satellite time series data to track abrupt changes and nonlinear dynamics: A Bayesian ensemble algorithm. Remote Sensing of Environment, 232, p.111181 (the beast algorithm paper).
2. Zhao, K., Valle, D., Popescu, S., Zhang, X. and Mallick, B., 2013. Hyperspectral remote sensing of plant biochemistry using Bayesian model averaging with variable and band selection. Remote Sensing of Environment, 132, pp.102-119 (the Bayesian MCMC scheme used in beast).
3. Hu, T., Toman, E.M., Chen, G., Shao, G., Zhou, Y., Li, Y., Zhao, K. and Feng, Y., 2021. Mapping fine-scale human disturbances in a working landscape with Landsat time series on Google Earth Engine. ISPRS Journal of Photogrammetry and Remote Sensing, 176, pp.250-261(a beast application paper).

Examples

```
library(Rbeast)
data(Yellowstone)
plot(Yellowstone,type='l')

result=beast(Yellowstone)
plot(result)
```

Index

∗ **misc**
  beast, 2
  beast.irreg, 20
  beast123, 32
  CNAchrom11, 51
  covid19, 52
  geeLandsat, 53
  googletrend_beach, 55
  imagestack, 56
  minesweeper, 57
  ohio, 59
  plot.beast, 60
  print.beast, 62
  simdata, 63
  tetris, 64
  tsextract, 66
  Yellowstone, 68

BEAST *(*beast*)*, 2
Beast *(*beast*)*, 2
beast, 2, *21*, *26*, *30*, *45*, *54*, *58*, *60*, *62, 63*, *66, 67*
beast.123 *(*beast123*)*, 32
BEAST.irreg *(*beast.irreg*)*, 20
beast.irreg, *4*, *9*, *12*, 20, *45*, *54*, *58*, *60*, *62, 63*, *66, 67*
BEAST123 *(*beast123*)*, 32
beast123, *4*, *8, 9*, *12*, *21*, *26*, *30*, 32, *54*, *58*, *60*, *62, 63*, *66, 67*

CNAchrom11, 51
Covid19 *(*covid19*)*, 52
covid19, 52

geeLandsat, *12*, *30*, *45*, 53, *58*, *62, 63*, *66, 67*
googletrend_beach, 55

imagestack, 56

minesweeper, *12*, *30*, *45*, *54*, 57, *62, 63*, *66, 67*

Ohio *(*ohio*)*, 59
ohio, *4*, *21*, 59

plot.beast, *39*, 60, *62*
print.beast, 62

RBEAST *(*beast*)*, 2
Rbeast *(*beast*)*, 2
rbeast *(*beast*)*, 2

simdata, 63

Tetris *(*tetris*)*, 64
tetris, *12*, *30*, *45*, *54*, *58*, *62, 63*, 64, *67*
tsextract, 66

Yellowstone, 68