Package: MARSS (via r-universe)

November 25, 2024

Type Package

Title Multivariate Autoregressive State-Space Modeling

Version 3.11.9

Date 2024-02-19

Depends R (>= 3.5.0)

Imports generics (>= 0.1.2), graphics, grDevices, KFAS (>= 1.0.1), mvtnorm, nlme, stats, utils

Suggests forecast, ggplot2, Hmisc, knitr, marssTMB

Description The MARSS package provides maximum-likelihood parameter estimation for constrained and unconstrained linear multivariate autoregressive state-space (MARSS) models, including partially deterministic models. MARSS models are a class of dynamic linear model (DLM) and vector autoregressive model (VAR) model. Fitting available via Expectation-Maximization (EM), BFGS (using optim), and 'TMB' (using the 'marssTMB' companion package). Functions are provided for parametric and innovations bootstrapping, Kalman filtering and smoothing, model selection criteria including bootstrap AICb, confidences intervals via the Hessian approximation or bootstrapping, and all conditional residual types. See the user guide for examples of dynamic factor analysis, dynamic linear models, outlier and shock detection, and multivariate AR-p models. Online workshops (lectures, eBook, and computer labs) at <https://atsa-es.github.io/>.

License GPL-2

LazyData yes

BuildVignettes yes

ByteCompile TRUE

URL https://atsa-es.github.io/MARSS/

BugReports https://github.com/atsa-es/MARSS/issues

RoxygenNote 7.3.1

Roxygen list(markdown = TRUE)

Encoding UTF-8

VignetteBuilder knitr, utils

Additional_repositories https://atsa-es.r-universe.dev

Repository https://nmfs-opensci.r-universe.dev

RemoteUrl https://github.com/atsa-es/MARSS

RemoteRef HEAD

RemoteSha a12ffb7d479968c524d71bc95b90e526cdee0440

Contents

`MARSS-package` *Multivariate Autoregressive State-Space Model Estimation*

Description

The MARSS package fits time-varying constrained and unconstrained multivariate autoregressive time-series models to multivariate time series data. To get started quickly, go to the Quick Start Guide (or at the command line, you can type `RShowDoc("Quick_Start", package="MARSS")`). To open the MARSS User Guide with many vignettes and examples, go to User Guide (or type `RShowDoc("UserGuide",package="MARSS")`).

The default MARSS model form is a MARXSS model: Multivariate Auto-Regressive(1) eXogenous inputs State-Space model. This model has the following form:

$$\mathbf{x}_t = \mathbf{B}\mathbf{x}_{t-1} + \mathbf{u} + \mathbf{C}\mathbf{c}_t + \mathbf{G}\mathbf{w}_t, \text{ where } \mathbf{W}_t \sim \text{MVN}(0, \mathbf{Q})$$

$$\mathbf{y}_t = \mathbf{Z}\mathbf{x}(t) + \mathbf{a} + \mathbf{D}\mathbf{d}_t + \mathbf{H}\mathbf{v}_t, \text{ where } \mathbf{V}_t \sim \text{MVN}(0, \mathbf{R})$$

$$\mathbf{X}_1 \sim \text{MVN}(\mathbf{x0}, \mathbf{V0}) \text{ or } \mathbf{X}_0 \sim \text{MVN}(\mathbf{x0}, \mathbf{V0})$$

All parameters can be time-varying; the t subscript is left off the parameters to remove clutter. Note, by default $\mathbf{V0}$ is a matrix of all zeros and thus $\mathbf{x}_1$ or $\mathbf{x}_0$ is treated as an estimated parameter not a diffuse prior.

The parameter matrices can have fixed values and linear constraints. This is an example of a 3x3 matrix with fixed values and linear constraints. In this example all the matrix elements can be written as a linear function of a, b, and c:

$$\begin{bmatrix} a+2b & 1 & a \\ 1+3a+b & 0 & b \\ 0 & -2 & c \end{bmatrix}$$

Values such as ab or a^2 or $ln(a)$ are not allowed as those would not be linear.

The MARSS model parameters, hidden state processes ($\mathbf{x}$), and observations ($\mathbf{y}$) are matrices:

- $\mathbf{x}_t$, $\mathbf{x0}$, and $\mathbf{u}$ are m x 1
- $\mathbf{y}_t$ and $\mathbf{a}$ are n x 1 (m<=n)
- $\mathbf{B}$ and $\mathbf{V0}$ are m x m
- $\mathbf{Z}$ is n x m
- $\mathbf{Q}$ is g x g (default m x m)
- $\mathbf{G}$ is m x g (default m x m identity matrix)
- $\mathbf{R}$ is h x h (default n x n)
- $\mathbf{H}$ is n x h (default n x n identity matrix)
- $\mathbf{C}$ is m x q
- $\mathbf{D}$ is n x p
- $\mathbf{c}_t$ is q x 1
- $\mathbf{d}_t$ is p x 1

If a parameter is time-varying then the time dimension is the 3rd dimension. Thus a time-varying $\mathbf{Z}$ would be n x m x T where T is the length of the data time series.

The main fitting function is `MARSS()` which is used to fit a specified model to data and estimate the model parameters. `MARSS()` estimates the model parameters using an EM algorithm (primarily but see `MARSSoptim()`). Functions are provided for parameter confidence intervals and the observed Fisher Information matrix, smoothed state estimates with confidence intervals, all the Kalman filter and smoother outputs, residuals and residual diagnostics, printing and plotting, and summaries.

Details

Main MARSS functions:

`MARSS()` Top-level function for specifying and fitting MARSS models.

`coef()` Returns the estimated parameters in a variety of formats.

`tidy()` Parameter estimates with confidence intervals

`tsSmooth()` $\mathbf{x}$ and $\mathbf{y}$ estimates output as a data frame. Output can be conditioned on all the data (T), data up to $t-1$, or data up to t. From the Kalman filter and smoother output.

`fitted()` Model $\mathbf{x}\mathbf{x}$ and $\mathbf{y}$ predictions as a data frame or matrices. Another user interface for model predictions is `predict.marssMLE`.

`residuals()` Model residuals as a data frame.

`MARSSresiduals()` Model residuals as a data frame or matrices. Normal user interface to residuals is `residuals.marssMLE`.

`predict()` Predictions and forecasts from a `marssMLE` object.

`plot for marssMLE` A series of plots of fits and residuals diagnostics.

`autoplot()` A series of plots using ggplot2 of fits and residuals diagnostics.

`glance()` Brief summary of fit.

`logLik()` Log-likelihood.

`print()` Prints a wide variety of output from a `marssMLE` object.

`print.marssMODEL()` Prints description of the MARSS model (`marssMODEL` object).

`plot.marssPredict()` Plot a prediction or forecast.

`toLatex.marssMODEL()` Outputs a LaTeX version of the model.

Other outputs for a fitted model:

`MARSSsimulate()` Produces simulated data from a MARSS model.

`MARSSkf()`, `MARSSkfas()`, `MARSSkfss()` Kalman filters and smoothers with extensive output of all the intermediate filter and smoother variances and expectations.

`MARSSaic()` Calculates AICc, AICc, and various bootstrap AICs.

`MARSSparamCIs()` Adds confidence intervals to a `marssMLE` object.

`MARSShessian()` Computes an estimate of the variance-covariance matrix for the MLE parameters.

`MARSSFisherI()` Returns the observed Fisher Information matrix.

Important internal MARSS functions (called by the above functions):

`MARSSkem()` Estimates MARSS parameters using an EM algorithm.

`MARSSoptim()` Estimates MARSS parameters using a quasi-Newton algorithm via `optim`.

`MARSShatyt()` Calculates the expectations involving Y.

`MARSSinnovationsboot()` Creates innovations bootstrapped data.

`MARSS.marss()` Discusses the form in which MARSS models are stored internally.

Use `help.search("internal", package="MARSS")` to see the documentation of all the internal functions in the MARSS R package.

Author(s)

Eli Holmes, Eric Ward and Kellie Wills, NOAA, Seattle, USA.

References

The MARSS User Guide: Holmes, E. E., E. J. Ward, and M. D. Scheuerell (2012) Analysis of multivariate time-series using the MARSS package. NOAA Fisheries, Northwest Fisheries Science Center, 2725 Montlake Blvd E., Seattle, WA 98112 User Guide or type `RShowDoc("UserGuide",package="MARSS")` to open a copy.

The MARSS Quick Start Guide: Quick Start Guide or type `RShowDoc("Quick_Start",package="MARSS")` to open a copy.

`accuracy` *Return accuracy metrics*

Description

This is a method for the generic `accuracy` function in the generics package. It is written to mimic the output from the accuracy function in the forecast package. See that package for details.

The measures calculated are:

- ME: Mean Error
- RMSE: Root Mean Squared Error
- MAE: Mean Absolute Error
- MPE: Mean Percentage Error
- MAPE: Mean Absolute Percentage Error
- MASE: Mean Absolute Scaled Error
- ACF1: Autocorrelation of errors at lag 1.

The MASE calculation is scaled using MAE of the training set naive forecasts which are simply $\mathbf{y}_{t-1}$.

For the training data, the metrics are shown for the one-step-ahead predictions by default (`type="ytt1"`). This is the prediction of $\mathbf{y}_t$ conditioned on the data up to $t-1$ (and the model estimated from all the data). With `type="ytT"`, you can compute the metrics for the fitted `ytT`, which is the expected value of new data at t conditioned on all the data. `type` does not affect test data (forecasts are past the end of the training data).

Usage

```
## S3 method for class 'marssPredict'
accuracy(object, x, test = NULL, type = "ytt1", verbose = FALSE, ...)
## S3 method for class 'marssMLE'
accuracy(object, x, test = NULL, type = "ytt1", verbose = FALSE, ...)
```

Arguments

object	A marssMLE or marssPredict object
x	A matrix or data frame with data to test against the h steps of a forecast.
test	Which time steps in training data (data model fit to) to compute accuracy for.
type	type="ytt1" is the one-step-ahead predictions. type="ytT" is the fitted ytT predictions. The former are standardly used for training data prediction metrics.
verbose	Show metrics for each time series of data.
...	Not used.

References

Hyndman, R.J. and Koehler, A.B. (2006) "Another look at measures of forecast accuracy". International Journal of Forecasting, 22(4), 679-688.

Hyndman, R.J. and Athanasopoulos, G. (2018) "Forecasting: principles and practice", 2nd ed., OTexts, Melbourne, Australia. Section 3.4 "Evaluating forecast accuracy". https://otexts.com/fpp2/accuracy.html.

Examples

```
dat <- t(harborSeal)
dat <- dat[c(2, 11, 12),]
train.dat <- dat[,1:12]
fit <- MARSS(train.dat, model = list(Z = factor(c("WA", "OR", "OR"))))

accuracy(fit)

# Compare to test data set
fr <- predict(fit, n.ahead=10)
test.dat <- dat[,13:22]
accuracy(fr, x=test.dat)
```

---

coef.marssMLE *Coefficient function for MARSS MLE objects*

Description

MARSS() outputs marssMLE objects. coef(object), where object is the output from a MARSS() call, will print out the estimated parameters. The default output is a list with values for each parameter, however the output can be altered using the type argument to output a vector of all the estimated values (type="vector") or a list with the full parameter matrix with the estimated and fixed elements (type="matrix"). For a summary of the parameter estimates with CIs from the estimated Hessian, use try tidy(object).

Usage

```
## S3 method for class 'marssMLE'
coef(object, ..., type = "list", form = NULL, what = "par")
```

Arguments

`object`	A `marssMLE` object.
`...`	Other arguments. Not used.
`type`	What to output. Default is "list". Options are
	"list" A list of only the estimated values in each matrix. Each model matrix has it's own list element.
	"vector" A vector of all the estimated values in each matrix.
	"matrix" A list of the parameter matrices each parameter with fixed values at their fixed values and the estimated values at their estimated values. Time-varying parameters, including d and c in a marxss form model, are returned as an array with time in the 3rd dimension.
	parameter name Returns the parameter matrix for that parameter with fixed values at their fixed values and the estimated values at their estimated values. Note, time-varying parameters, including d and c in a marxss form model, are returned as an array with time in the 3rd dimension.
`form`	This argument can be ignored. By default, the model form specified in the call to `MARSS()` is used to determine how to display the coefficients. This information is in `attr(object$model,"form")`. The default form is `"marxss"`; see `MARSS.marxss()`. However, the internal functions convert this to form `"marss"`; see `MARSS.marss()`. The marss form of the model is stored (in `object$marss`). You can look at the coefficients in marss form by passing in `form="marss"`.
`what`	By default, `coef()` shows the parameter estimates. Other options are "par.se", "par.lowCI", "par.upCI", "par.bias", and "start".

Value

A list of the estimated parameters for each model matrix.

Author(s)

Eli Holmes, NOAA, Seattle, USA.

See Also

`tidy()`, `print()`

Examples

```
dat <- t(harborSeal)
dat <- dat[c(2, 11), ]
fit <- MARSS(dat)

coef(fit)
coef(fit, type = "vector")
coef(fit, type = "matrix")
# to retrieve just the Q matrix
coef(fit, type = "matrix")$Q
```

CSEGriskfigure *Plot Extinction Risk Metrics*

Description

Generates a six-panel plot of extinction risk metrics used in Population Viability Analysis (PVA). This is a function used by one of the vignettes in the MARSS-package.

Usage

```
CSEGriskfigure(data, te = 100, absolutethresh = FALSE, threshold = 0.1,
  datalogged = FALSE, silent = FALSE, return.model = FALSE,
  CI.method = "hessian", CI.sim = 1000)
```

Arguments

data	A data matrix with 2 columns; time in first column and counts in second column. Note time is down rows, which is different than the base MARSS-package functions.
te	Length of forecast period (positive integer)
absolutethresh	Is extinction threshold an absolute number? (T/F)
threshold	Extinction threshold either as an absolute number, if absolutethresh=TRUE, or as a fraction of current population count, if absolutethresh=FALSE.
datalogged	Are the data already logged? (T/F)
silent	Suppress printed output? (T/F)
return.model	Return state-space model as marssMLE object? (T/F)
CI.method	Confidence interval method: "hessian", "parametrc", "innovations", or "none". See MARSSparamCIs.
CI.sim	Number of simulations for bootstrap confidence intervals (positive integer).

Details

Panel 1: Time-series plot of the data. Panel 2: CDF of extinction risk. Panel 3: PDF of time to reach threshold. Panel 4: Probability of reaching different thresholds during forecast period. Panel 5: Sample projections. Panel 6: TMU plot (uncertainty as a function of the forecast).

Value

If return.model=TRUE, an object of class marssMLE.

Author(s)

Eli Holmes, NOAA, Seattle, USA, and Steve Ellner, Cornell Univ.

References

Holmes, E. E., E. J. Ward, and M. D. Scheuerell (2012) Analysis of multivariate time-series using the MARSS package. NOAA Fisheries, Northwest Fisheries Science Center, 2725 Montlake Blvd E., Seattle, WA 98112 Type `RShowDoc("UserGuide",package="MARSS")` to open a copy.

(theory behind the figure) Holmes, E. E., J. L. Sabo, S. V. Viscido, and W. F. Fagan. (2007) A statistical approach to quasi-extinction forecasting. Ecology Letters 10:1182-1198.

(CDF and PDF calculations) Dennis, B., P. L. Munholland, and J. M. Scott. (1991) Estimation of growth and extinction parameters for endangered species. Ecological Monographs 61:115-143.

(TMU figure) Ellner, S. P. and E. E. Holmes. (2008) Resolving the debate on when extinction risk is predictable. Ecology Letters 11:E1-E5.

See Also

MARSSboot, marssMLE, CSEGtmufigure

Examples

```
d <- harborSeal[, 1:2]
kem <- CSEGriskfigure(d, datalogged = TRUE)
```

`CSEGtmufigure` *Plot Forecast Uncertainty*

Description

Plot the uncertainty in the probability of hitting a percent threshold (quasi-extinction) for a single random walk trajectory. This is the quasi-extinction probability used in Population Viability Analysis. The uncertainty is shown as a function of the forecast, where the forecast is defined in terms of the forecast length (number of time steps) and forecasted decline (percentage). This is a function used by one of the vignettes in the MARSS-package.

Usage

```
CSEGtmufigure(N = 20, u = -0.1, s2p = 0.01, make.legend = TRUE)
```

Arguments

`N`	Time steps between the first and last population data point (positive integer)
`u`	Per time-step decline (-0.1 means a 10% decline per time step; 1 means a doubling per time step.)
`s2p`	Process variance (Q). (a positive number)
`make.legend`	Add a legend to the plot? (T/F)

Details

This figure shows the region of high uncertainty in dark grey. In this region, the minimum 95 percent confidence intervals on the probability of quasi-extinction span 80 percent of the 0 to 1 probability. Green hashing indicates where the 95 percent upper bound does not exceed 5% probability of quasi-extinction. The red hashing indicates, where the 95 percent lower bound is above 95% probability of quasi-extinction. The light grey lies between these two certain/uncertain extremes. The extinction calculation is based on Dennis et al. (1991). The minimum theoretical confidence interval is based on Fieberg and Ellner (2000). This figure was developed in Ellner and Holmes (2008).

Examples using this figure are shown in the User Guide in the PVA chapter.

Author(s)

Eli Holmes, NOAA, Seattle, USA, and Steve Ellner, Cornell Univ.

References

Dennis, B., P. L. Munholland, and J. M. Scott. (1991) Estimation of growth and extinction parameters for endangered species. Ecological Monographs 61:115-143.

Fieberg, J. and Ellner, S.P. (2000) When is it meaningful to estimate an extinction probability? Ecology, 81, 2040-2047.

Ellner, S. P. and E. E. Holmes. (2008) Resolving the debate on when extinction risk is predictable. Ecology Letters 11:E1-E5.

See Also

CSEGriskfigure

Examples

```
CSEGtmufigure(N = 20, u = -0.1, s2p = 0.01)
```

`datasets` *Example Data Sets*

Description

Example data sets for use in MARSS vignettes for the MARSS-package.

- plankton Plankton data sets: Lake WA plankton 32-year time series and Ives et al data from West Long Lake.
- SalmonSurvCUI Snake River spring/summer chinook survival indices.
- isleRoyal Isle Royale wolf and moose data with temperature and precipitation covariates.
- population-count-data A variety of fish, mammal and bird population count data sets.
- loggerhead Loggerhead turtle tracking (location) data from ARGOS tags.
- harborSeal Harbor seal survey data (haul out counts) from Oregon, Washington and California, USA.

`fitted.marssMLE` *Return fitted values for X(t) and Y(t) in a MARSS model*

Description

`fitted()` returns the different types of fitted values for $\mathbf{x}_t$ and $\mathbf{y}_t$ in a MARSS model. The fitted values are the expected value of the right side of the MARSS equations without the error terms, thus are the model predictions of $\mathbf{y}_t$ or $\mathbf{x}_t$. `fitted.marssMLE` is a companion function to `tsSmooth()` which returns the expected value of the right side of the MARSS equations with the error terms (the Kalman filter and smoother output).

$$\mathbf{x}_t = \mathbf{B}\mathbf{x}_{t-1} + \mathbf{u} + \mathbf{C}\mathbf{c}_t + \mathbf{G}\mathbf{w}_t, \text{ where } \mathbf{W}_t \sim \text{MVN}(0, \mathbf{Q})$$

$$\mathbf{y}_t = \mathbf{Z}\mathbf{x}_t + \mathbf{a} + \mathbf{D}\mathbf{d}_t + \mathbf{H}\mathbf{v}_t, \text{ where } \mathbf{V}_t \sim \text{MVN}(0, \mathbf{R})$$

The data go from $t = 1$ to $t = T$. For brevity, the parameter matrices are shown without a time subscript, however all parameters can be time-varying.

Note that the prediction of $\mathbf{x}_t$ conditioned on the data up to time t is not provided since that would require the expected value of $\mathbf{X}_t$ conditioned on data from $t = 1$ to $t + 1$, which is not output from the Kalman filter or smoother.

Usage

```
## S3 method for class 'marssMLE'
fitted(object, ...,
    type = c("ytt1", "ytT", "xtT", "ytt", "xtt1"),
    interval = c("none", "confidence", "prediction"),
    level = 0.95,
    output = c("data.frame", "matrix"),
    fun.kf = c("MARSSkfas", "MARSSkfss"))
```

Arguments

`object`	A `marssMLE` object.
`type`	If `type="tT"`, then the predictions are conditioned on all the data. If `type="tt"`, then the predictions are conditioned on data up to time t. If `type="tt1"`, the predictions are conditioned on data up to time $t-1$. The latter are also known as one-step-ahead estimates. For $\mathbf{y}$, these are also known as the innovations.
`interval`	If `interval="confidence"`, then the standard error and confidence interval of the predicted value is returned. If `interval="prediction"`, then the standard deviation and prediction interval of new data or states are returned.
`level`	Level for the intervals if `interval` is not equal to `"none"`.
`output`	data frame or list of matrices
`fun.kf`	By default, `tsSmooth()` will use the Kalman filter/smoother function in `object$fun.kf` (either `MARSSkfas()` or `MARSSkfss()`). You can pass in `fun.kf` to force a particular Kalman filter/smoother function to be used.
`...`	Not used.

Details

In the state-space literature, the two most commonly used fitted values are `"ytt1"` and `"ytT"`. The former is the expected value of $\mathbf{Y}_t$ conditioned on the data 1 to time $t-1$. These are known as the innovations and they, plus their variance, are used in the calculation of the likelihood of a MARSS model via the innovations form of the likelihood. The latter, `"ytT"` are the model estimates of the $\mathbf{y}$ values using all the data; this is the expected value of $\mathbf{Z}\mathbf{X}_t + \mathbf{a} + \mathbf{D}\mathbf{d}_t$ conditioned on the data 1 to T. The `"xtT"` along with `"ytT"` are used for computing smoothation residuals used in outlier and shock detection. See `MARSSresiduals`. For completeness, `fitted.marssMLE` will also return the other possible model predictions with different conditioning on the data (1 to $t-1$, t, and T), however only `type="ytt1"` (innovations) and `"ytT"` and `"xtT"` (smoothations) are regularly used. Keep in mind that the fitted `"xtT"` is not the smoothed estimate of $\mathbf{x}$ (unlike `"ytT"`). If you want the smoothed estimate of $\mathbf{x}$ (i.e. the expected value of $\mathbf{X}_t$ conditioned on all the data), you want the Kalman smoother. See `tsSmooth`.

Fitted versus estimated values: The fitted states at time t are predictions from the estimated state at time $t-1$ conditioned on the data: expected value of $\mathbf{B}\mathbf{X}_{t-1} + \mathbf{u} + \mathbf{C}\mathbf{c}_t$ conditioned on the data. They are distinguished from the estimated states at time t which would includes the conditional expected values of the error terms: $\mathrm{E}[\mathbf{X}_t] = \mathbf{B}\mathbf{X}_{t-1} + \mathbf{u} + \mathbf{C}\mathbf{c}_t + \mathbf{w}_t$. The latter are returned by the Kalman filter and smoother. Analogously, the fitted observations at time t are model predictions from the estimated state at time t conditioned on the data: the expected value of the right side of the $\mathbf{y}$ equation without the error term. Like with the states, one can also compute the expected value of the observations at time t conditioned on the data: the expected value of the right side of the $\mathbf{y}$ equation with the error term. The latter would just be equal to the data if there are no missing data, but when there are missing data, this is what is used to estimate their values. The expected value of states and observations are provided via `tsSmooth`.

observation fitted values

The model predicted $\hat{\mathbf{y}}_t$ is $\mathbf{Z}\mathbf{x}_t^\tau + \mathbf{a} + \mathbf{D}\mathbf{d}_t$, where $\mathbf{x}_t^\tau$ is the expected value of the state at time t conditioned on the data from 1 to τ (τ will be $t-1$, t or T). Note, if you are using MARSS for estimating the values for missing data, then you want to use `tsSmooth()` with `type="ytT"` not `fitted(..., type="ytT")`.

$\mathbf{x}_t^\tau$ is the expected value of the states at time t conditioned on the data from time 1 to τ. If `type="ytT"`, the expected value is conditioned on all the data, i.e. `xtT` returned by `MARSSkf()`. If `type="ytt1"`, then the expected value uses only the data up to time $t-1$, i.e. `xtt1` returned by `MARSSkf()`. These are commonly known as the one step ahead predictions for a state-space model. If `type="ytt"`, then the expected value uses the data up to time t, i.e. `xtt` returned by `MARSSkf()`.

If `interval="confidence"`, the standard error and interval is for the predicted $\mathbf{y}$. The standard error is $\mathbf{Z}\mathbf{V}_t^\tau\mathbf{Z}^\top$. If `interval="prediction"`, the standard deviation of new iid $\mathbf{y}$ data sets is returned. The standard deviation of new $\mathbf{y}$ is $\mathbf{Z}\mathbf{V}_t^\tau\mathbf{Z}^\top + \mathbf{R}_t$. $\mathbf{V}_t^\tau$ is conditioned on the data from $t=1$ to n. τ will be either t, $t-1$ or T depending on the value of `type`.

Intervals returned by `predict()` are not for the data used in the model but rather new data sets. To evaluate the data used to fit the model for residuals analysis or analysis or model inadequacy, you want the model residuals (and residual se's). Use `residuals` for model residuals and their intervals. The intervals for model residuals are narrower because the predictions for $\mathbf{y}$ were estimated from the model data (so is closer to the data used to estimate the predictions than new independent data will be).

state fitted values

The model predicted $\mathbf{x}_t$ given $\mathbf{x}_{t-1}$ is $\mathbf{B}\mathbf{x}_{t-1}^{\tau} + \mathbf{u} + \mathbf{C}\mathbf{c}_t$. If you want estimates of the states, rather than the model predictions based on $\mathbf{x}_{t-1}$, go to `tsSmooth()`. Which function you want depends on your objective and application.

$\mathbf{x}_{t-1}^{\tau}$ used in the prediction is the expected value of the states at time $t-1$ conditioned on the data from $t=1$ to $t=\tau$. If `type="xtT"`, this is the expected value at time $t-1$ conditioned on all the data, i.e. `xtT[,t-1]` returned by `MARSSkf()`. If `type="xtt1"`, it is the expected value conditioned on the data up to time $t-1$, i.e. `xtt[,t-1]` returned by `MARSSkf()`. The predicted state values conditioned on data up to t are not provided. This would require the expected value of states at time t conditioned on data up to time $t+1$, and this is not output by the Kalman filter. Only conditioning on data up to $t-1$ and T are output.

If `interval="confidence"`, the standard error of the predicted values (meaning the standard error of the expected value of $\mathbf{X}_t$ given $\mathbf{X}_{t-1}$) is returned. The standard error of the predicted value is $\mathbf{B}\mathbf{V}_{t-1}^{\tau}\mathbf{B}^{\top}$. If `interval="prediction"`, the standard deviation of $\mathbf{X}_t$ given $\mathbf{X}_{t-1}$ is output. The latter is $\mathbf{B}\mathbf{V}_{t-1}^{\tau}\mathbf{B}^{\top} + \mathbf{Q}$. $\mathbf{V}_{t-1}^{\tau}$ is either conditioned on data 1 to $\tau = T$ or $\tau = t-1$ depending on `type`.

The intervals returned by `fitted.marssMLE()` are for the state predictions based on the state estimate at $t-1$. These are not typically what one uses or needs–however might be useful for simulation work. If you want confidence intervals for the state estimates, those are provided in `tsSmooth`. If you want to do residuals analysis (for outliers or model inadequacy), you want the residuals intervals provided in `residuals()`.

Value

If `output="data.frame"` (the default), a data frame with the following columns is returned. If `output="matrix"`, the columns are returned as matrices in a list. Information computed from the model has a leading "." in the column name.

If `interval="none"`, the following are returned (colnames with `.` in front are computed values):

`.rownames`	Names of the data or states.
`t`	Time step.
`y`	The data if `type` is `"ytT"`, `"ytt"` or `"ytt1"`.
`.x`	The expected value of $\mathbf{X}_t$ conditioned on all the data if `type="xtT"` or data up to time t if `type="xtt1"`. From `tsSmooth()`. This is the expected value of the right-side of the $\mathbf{x}_t$ equation with the errors terms while `.fitted` is the expected value of the right side without the error term $\mathbf{w}_t$.
`.fitted`	Predicted values of observations ($\mathbf{y}$) or the states ($\mathbf{x}$). See details.

If `interval = "confidence"`, the following are also returned:

`.se`	Standard errors of the predictions.
`.conf.low`	Lower confidence level at `alpha = 1-level`. The interval is approximated using qnorm(alpha/2)*.se + .fitted
`.conf.up`	Upper confidence level. The interval is approximated using qnorm(1-alpha/2)*.se + .fitted

The confidence interval is for the predicted value, i.e. $\mathbf{Z}\mathbf{x}_t^\tau + \mathbf{a}$ for $\mathbf{y}$ or $\mathbf{B}\mathbf{x}_{t-1}^\tau + \mathbf{u}$ for $\mathbf{x}$ where $\mathbf{x}_t^\tau$ is the expected value of $\mathbf{X}_t$ conditioned on the data from 1 to τ. (τ will be $t-1$, t or T).

If `interval = "prediction"`, the following are also returned:

`.sd`	Standard deviation of new $\mathbf{x}_t$ or $\mathbf{y}_t$ iid values.
`.lwr`	Lower range at `alpha = 1-level`. The interval is approximated using qnorm(alpha/2)*.sd + .fitted
`.upr`	Upper range at `level`. The interval is approximated using qnorm(1-alpha/2)*.sd + .fitted

The prediction interval is for new $\mathbf{x}_t$ or $\mathbf{y}_t$. If you want to evaluate the observed data or the states estimates for outliers then these are not the intervals that you want. For that you need the residuals intervals provided by `residuals()`.

Author(s)

Eli Holmes, NOAA, Seattle, USA.

See Also

`MARSSkf()`, `MARSSresiduals()`, `residuals()`, `predict()`, `tsSmooth()`

Examples

```
dat <- t(harborSeal)
dat <- dat[c(2, 11, 12), ]
fit <- MARSS(dat, model = list(Z = factor(c("WA", "OR", "OR"))))
fitted(fit)

# Example of fitting a stochastic level model to the Nile River flow data
# red line is smooothed level estimate
# grey line is one-step-ahead prediction using xtt1
nile <- as.vector(datasets::Nile)
mod.list <- list(
  Z = matrix(1), A = matrix(0), R = matrix("r"),
  B = matrix(1), U = matrix(0), Q = matrix("q"),
  x0 = matrix("pi")
)
fit <- MARSS(nile, model = mod.list, silent = TRUE)

# Plotting
# There are plot methods for marssMLE objects
library(ggplot2)
autoplot(fit)

# Below shows how to make plots manually but all of these can be made
# with autoplot(fit) or plot(fit)
plot(nile, type = "p", pch = 16, col = "blue")
lines(fitted(fit, type="ytT")$.fitted, col = "red", lwd = 2)
lines(fitted(fit, type="ytt1")$.fitted, col = "grey", lwd = 2)
```

```
# Make a plot of the model estimate of y(t), i.e., put a line through the points
# Intervals are for new data not the blue dots
# (which were used to fit the model so are not new)
library(ggplot2)
d <- fitted(fit, type = "ytT", interval="confidence", level=0.95)
d2 <- fitted(fit, type = "ytT", interval="prediction", level=0.95)
d$.lwr <- d2$.lwr
d$.upr <- d2$.upr
ggplot(data = d) +
  geom_line(aes(t, .fitted), linewidth=1) +
  geom_point(aes(t, y), color = "blue", na.rm=TRUE) +
  geom_ribbon(aes(x = t, ymin = .conf.low, ymax = .conf.up), alpha = 0.3) +
  geom_line(aes(t, .lwr), linetype = 2) +
  geom_line(aes(t, .upr), linetype = 2) +
  facet_grid(~.rownames) +
  xlab("Time Step") + ylab("Count") +
  ggtitle("Blue=data, Black=estimate, grey=CI, dash=prediction interval") +
  geom_text(x=15, y=7, label="The intervals are for \n new data not the blue dots")
```

`forecast.marssMLE` *forecast function for marssMLE objects*

Description

`MARSS()` outputs `marssMLE` objects. `forecast(object)`, where object is `marssMLE` object, will return the forecasts of $\mathbf{y}_t$ or $\mathbf{x}_t$ for `h` steps past the end of the model data. `forecast(object)` returns a `marssPredict` object which can be passed to `plot.marssPredict` for automatic plotting of the forecast. `forecast.marssMLE()` is used by `predict.marssMLE()` to generate forecasts.

This is a method for the generic `forecast` function in the generics package. It is written to mimic the behavior and look of the forecast package.

Usage

```
## S3 method for class 'marssMLE'
forecast(object, h = 10, level = c(0.8, 0.95),
     type = c("ytT","xtT", "ytt", "ytt1", "xtt", "xtt1"),
     newdata = list(y = NULL, c = NULL, d = NULL),
     interval = c("prediction", "confidence", "none"),
     fun.kf = c("MARSSkfas", "MARSSkfss"), ...)
```

Arguments

`object`	A `marssMLE` object.
`h`	Number of steps ahead to forecast. `newdata` is for the forecast, i.e. for the h time steps after the end of the model data. If there are covariates in the model, $\mathbf{c}_t$ or $\mathbf{d}_t$, then `newdata` is required. See details.

level	Level for the intervals if interval != "none".
type	The default for observations would be type="ytT" and for the states would be type="xtT", i.e. using all the data. Other possible forecasts are provided for completeness but would in most cases be identical (see details).
newdata	An optional list with matrices for new covariates $\mathbf{c}_t$ or $\mathbf{d}_t$ to use for the forecasts. $\mathbf{c}_t$ or $\mathbf{d}_t$ must be in the original model and have the same matrix rows and columns as used in the MARSS() call but the number of time steps can be different (and should be equal to h).
interval	If interval="confidence", then the standard error and confidence interval of the expected value of $\mathbf{y}_t$ (type="ytT") or $\mathbf{x}_t$ (type="xtT") is returned. interval="prediction" (default) returns prediction intervals which include the uncertainty in the expected value and due to observation error (the $\mathbf{R}$ in the $\mathbf{y}$ equation). Note, in the context of a MARSS model, only confidence intervals are available for the states (the $\mathbf{x}$).
fun.kf	Only if you want to change the default Kalman filter. Can be ignored.
...	Other arguments. Not used.

Details

The type="ytT" forecast for $T + i$ is

$$\mathbf{Z}\mathbf{x}_{T+i}^{T} + \mathbf{a} + \mathbf{D}\mathbf{d}_{T+i}$$

where $\mathbf{Z}$, $\mathbf{a}$ and $\mathbf{D}$ are estimated from the data from $t = 1$ to T. If the model includes $\mathbf{d}_t$ then newdata with d must be passed in. Either confidence or prediction intervals can be shown. Prediction intervals would be the norm for forecasts and show the intervals for new data which based on the conditional variance of $\mathbf{Z}\mathbf{X}_{T+i} + \mathbf{V}_{T+i}$. Confidence intervals would show the variance of the mean of the new data (such as if you ran a simulation multiple times and recorded only the mean observation time series). It is based on the conditional variance of $\mathbf{Z}\mathbf{X}_{T+i}$. The intervals shown are computed with fitted().

The type="xtT" forecast for $T + i$ is

$$\mathbf{B}\mathbf{x}_{T+i-1}^{T} + \mathbf{u} + \mathbf{C}\mathbf{c}_{T+i}$$

where $\mathbf{B}$ and $\mathbf{u}$ and $\mathbf{C}$ are estimated from the data from $t = 1$ to T (i.e. the estimates in the marssMLE object). If the model includes $\mathbf{c}_t$ then newdata with c must be passed in. The only intervals are confidence intervals which based on the conditional variance of $\mathbf{B}\mathbf{X}_{T+i-1} + \mathbf{W}_{T+i}$. If you pass in data for your forecast time steps, then the forecast will be computed conditioned on the original data plus the data in the forecast period. The intervals shown are computed with the Kalman smoother (or filter if type="xtt" or type="xtt1" specified) via tsSmooth().

If the model has time-varying parameters, the parameter estimates at time T will be used for the whole forecast. If new data c or d are passed in, it must have h time steps.

Note: y in newdata. Data along with covariates can be passed into newdata. In this case, the data in newdata ($T + 1$ to $T + h$) are conditioned on for the expected value of $\mathbf{X}_t$ but parameters used are only estimated using the data in the marssMLE object ($t = 1$ to T). If you include data in newdata, you need to decide how to condition on that new data for the forecast. type="ytT" would mean that the $t = T + i$ forecast is conditioned on all the data, $t = 1$ to $T + h$, type="ytt" would mean

that the $t = T + i$ forecast is conditioned on the data, $t = 1$ to $T + i$, and `type="ytt1"` would mean that the $t = T + i$ forecast is conditioned on the data, $t = 1$ to $T + i - 1$. Because MARSS models can be used in all sorts of systems, the $\mathbf{y}$ part of the MARSS model might not be "data" in the traditional sense. In some cases, one of the $\mathbf{y}$ (in a multivariate model) might be a known deterministic process or it might be a simulated future $\mathbf{y}$ that you want to include. In this case the $\mathbf{y}$ rows that are being forecasted are NAs and the $\mathbf{y}$ rows that are known are passed in with `newdata`.

Value

A list with the following components:

`method`	The method used for fitting, e.g. "kem".
`model`	The `marssMLE` object passed into `forecast.marssMLE()`.
`newdata`	The `newdata` list if passed into `forecast.marssMLE()`.
`level`	The confidence `level` passed into `forecast.marssMLE()`.
`pred`	A data frame the forecasts along with the intervals.
`type`	The `type` ("ytT" or "xtT") passed into `forecast.marssMLE()`.
`t`	The time steps used to fit the model (used for plotting).
`h`	The number of forecast time steps (used for plotting).

Author(s)

Eli Holmes, NOAA, Seattle, USA.

See Also

`plot.marssPredict()`, `predict.marssMLE()`

Examples

```
# More examples are in ?predict.marssMLE

dat <- t(harborSealWA)
dat <- dat[2:4,] #remove the year row
fit <- MARSS(dat, model=list(R="diagonal and equal"))

# 2 steps ahead forecast
fr <- forecast(fit, type="ytT", h=2)
plot(fr)

# forecast and only show last 10 steps of original data
fr <- forecast(fit, h=10)
plot(fr, include=10)
```

glance *Return brief summary information on a MARSS fit*

Description

This returns a data frame with brief summary information.

coef.det The coefficient of determination. This is the squared correlation between the fitted values and the original data points. This is simply a metric for the difference between the data points and the fitted values and should not be used for formal model comparison.

sigma The sample variance (unbiased) of the data residuals (fitted minus data). This is another simple metric of the difference between the data and fitted values. This is different than the sigma returned by an `arima()` call for example. That sigma would be akin to `sqrt(Q)` in the MARSS parameters; 'akin' because MARSS models are multivariate and the sigma returned by `arima()` is for a univariate model.

df The number of estimated parameters. Denoted `num.params` in a `marssMLE` object.

logLik The log-likelihood.

AIC Akaike information criterion.

AICc Akaike information criterion corrected for small sample size.

AICbb Non-parametric bootstrap Akaike information criterion if in the `marssMLE` object.

AICbp Parametric bootstrap Akaike information criterion if in the `marssMLE` object.

convergence 0 if converged according to the convergence criteria set. Note the default convergence criteria are high in order to speed up fitting. A number other than 0 means the model did not meet the convergence criteria.

errors 0 if no errors. 1 if some type of error or warning returned.

Usage

```
## S3 method for class 'marssMLE'
glance(x, ...)
```

Arguments

`x`	A `marssMLE` object
`...`	Not used.

Examples

```
dat <- t(harborSeal)
dat <- dat[c(2, 11, 12), ]
fit <- MARSS(dat, model = list(Z = factor(c("WA", "OR", "OR"))))

glance(fit)
```

harborSeal *Harbor Seal Population Count Data (Log counts)*

Description

Data sets used in MARSS vignettes in the MARSS-package. These are data sets based on logged count data from Oregon, Washington and California sites where harbor seals were censused while hauled out on land. "harborSealnomiss" is an extrapolated data set where missing values in the original data set have been extrapolated so that the data set can be used to demonstrate fitting population models with different underlying structures.

Usage

```
data(harborSeal)
data(harborSealWA)
```

Format

Matrix "harborSeal" contains columns "Years", "StraitJuanDeFuca", "SanJuanIslands", "EasternBays", "PugetSound", "HoodCanal", "CoastalEstuaries", "OlympicPeninsula", "CA.Mainland", "OR.NorthCoast", "CA.ChannelIslands", and "OR.SouthCoast".

Matrix "harborSealnomiss" contains columns "Years", "StraitJuanDeFuca", "SanJuanIslands", "EasternBays", "PugetSound", "HoodCanal", "CoastalEstuaries", "OlympicPeninsula", "OR.NorthCoast", and "OR.SouthCoast". Matrix "harborSealWA" contains columns "Years", "SJF", "SJI", "EBays", "PSnd", and "HC", representing the same five sites as the first five columns of "harborSeal".

Details

Matrix "harborSealWA" contains the original 1978-1999 logged count data for five inland WA sites. Matrix "harborSealnomiss" contains 1975-2003 data for the same sites as well as four coastal sites, where missing values have been replaced with extrapolated values. Matrix "harborSeal" contains the original 1975-2003 LOGGED data (with missing values) for the WA and OR sites as well as a CA Mainland and CA ChannelIslands time series.

Source

Jeffries et al. 2003. Trends and status of harbor seals in Washington State: 1978-1999. Journal of Wildlife Management 67(1):208-219.

Brown, R. F., Wright, B. E., Riemer, S. D. and Laake, J. 2005. Trends in abundance and current status of harbor seals in Oregon: 1977-2003. Marine Mammal Science, 21: 657-670.

Lowry, M. S., Carretta, J. V., and Forney, K. A. 2008. Pacific harbor seal census in California during May-July 2002 and 2004. California Fish and Game 94:180-193.

Hanan, D. A. 1996. Dynamics of Abundance and Distribution for Pacific Harbor Seal, Phoca vitulina richardsi, on the Coast of California. Ph.D. Dissertation, University of California, Los Angeles. 158pp. DFO. 2010. Population Assessment Pacific Harbour Seal (Phoca vitulina richardsi). DFO Can. Sci. Advis. Sec. Sci. Advis. Rep. 2009/011.

Examples

```
str(harborSealWA)
str(harborSeal)
```

`isleRoyal` *Isle Royale Wolf and Moose Data*

Description

Example data set for estimation of species interaction strengths. These are data on the number of wolves and moose on Isle Royal, Michigan. The data are unlogged. The covariate data are the average Jan-Feb, average Apr-May and average July-Sept temperature (Fahrenheit) and precipitation (inches). Also included are 3-year running means of these covariates. The choice of covariates is based on those presented in the Isle Royale 2012 annual report.

Usage

```
data(isleRoyal)
```

Format

The data are supplied as a matrix with years in the first column.

Source

Peterson R. O., Thomas N. J., Thurber J. M., Vucetich J. A. and Waite T. A. (1998) Population limitation and the wolves of Isle Royale. In: Biology and Conservation of Wild Canids (eds. D. Macdonald and C. Sillero-Zubiri). Oxford University Press, Oxford, pp. 281-292.

Vucetich, J. A. and R. O. Peterson. (2010) Ecological studies of wolves on Isle Royale. Annual Report 2009-10. School of Forest Resources and Environmental Science, Michigan Technological University, Houghton, Michigan USA 49931-1295

The source for the covariate data is the Western Regional Climate Center (http://www.wrcc.dri.edu) using their data for the NE Minnesota division. The website used was http://www.wrcc.dri.edu/cgi-bin/divplot1_form.pl?2103 and www.wrcc.dri.edu/spi/divplot1map.html.

Examples

```
str(isleRoyal)
```

ldiag *Return a diagonal list matrix*

Description

Creates a list diagonal matrix where the diagonal can be a combination of numbers and characters. Characters are names of parameters to be estimated.

Usage

```
ldiag(x, nrow = NA)
```

Arguments

x	A vector or list of single values
nrow	Rows in square matrix

Details

A diagonal list matrix is returned. The off-diagonals will be 0 and the diagonal will be x . x can be a combination of numbers and characters. If x is numeric, the diagonal will still be list type so that later the diagonal can be replace with characters. See examples.

Value

a square list matrix

Author(s)

Eli Holmes, NOAA, Seattle, USA.

Examples

```
ldiag(list(0, "b"))
ldiag("a", nrow=3)

# This works
a <- ldiag(1:3)
diag(a) <- "a"
diag(a) <- list("a", 0, 0)

# ldiag() is a convenience function to replace having to
# write code like this
a <- matrix(list(0), 3, 3)
diag(a) <- list("a", 0, 0)

# diag() alone won't work because it cannot handle mixed number/char lists

# This turns the off-diagonals to character "0"
```

```
a <- diag(1:3)
diag(a) <- "a"

# This would turn our matrix into a list only (not a matrix anymore)
a <- diag(1:3)
diag(a) <- list("a", 0, 0)

# This would return NA on the diagonal
a <- diag("a", 3)
```

`loggerhead` *Loggerhead Turtle Tracking Data*

Description

Data used in MARSS vignettes in the MARSS-package. Tracking data from ARGOS tags on eight individual loggerhead turtles, 1997-2006.

Usage

```
data(loggerhead)
data(loggerheadNoisy)
```

Format

Data frames "loggerhead" and "loggerheadNoisy" contain the following columns:

turtle Turtle name.

day Day of the month (character).

month Month number (character).

year Year (character).

lon Longitude of observation.

lat Latitude of observation.

Details

Data frame "loggerhead" contains the original latitude and longitude data. Data frame "loggerhead-Noisy" has noise added to the lat and lon data to represent data corrupted by errors.

Source

Gray's Reef National Marine Sanctuary (Georgia) and WhaleNet: http://whale.wheelock.edu/whalenet-stuff/stop_cover_archive.html

Examples

```
str(loggerhead)
str(loggerheadNoisy)
```

`logLik.marssMLE` *logLik method for MARSS MLE objects*

Description

Returns a logLik class object with attributes nobs and df.

Usage

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

Arguments

`object`	A `marssMLE` object.
`...`	Other arguments. Not used.

Value

An object of class logLik.

Author(s)

Eli Holmes, NOAA, Seattle, USA.

See Also

`MARSSkf()`

Examples

```
dat <- t(harborSeal)
dat <- dat[c(2, 11, 12), ]
MLEobj <- MARSS(dat, model = list(Z = factor(c("WA", "OR", "OR"))))
logLik(MLEobj)
```

MARSS *Fit a MARSS Model via Maximum-Likelihood Estimation*

Description

This is the main function for fitting multivariate autoregressive state-space (MARSS) models with linear constraints. Scroll down to the bottom to see some short examples. To open a guide to show you how to get started quickly, type `RShowDoc("Quick_Start",package="MARSS")`. To open the MARSS User Guide from the command line, type `RShowDoc("UserGuide",package="MARSS")`. To get an overview of the package and all its main functions and how to get output (parameter estimates, fitted values, residuals, Kalmin filter or smoother output, or plots), go to `MARSS-package`. If `MARSS()` is throwing errors or warnings that you don't understand, try the Troubleshooting section of the user guide or type `MARSSinfo()` at the command line.

The default MARSS model form is "marxss", which is Multivariate Auto-Regressive(1) eXogenous inputs State-Space model:

$$\mathbf{x}_t = \mathbf{B}_t\mathbf{x}_{t-1} + \mathbf{u}_t + \mathbf{C}_t\mathbf{c}_t + \mathbf{G}_t\mathbf{w}_t, \text{ where } \mathbf{W}_t \sim \text{MVN}(0, \mathbf{Q}_t)$$

$$\mathbf{y}_t = \mathbf{Z}_t\mathbf{x}_t + \mathbf{a}_t + \mathbf{D}_t\mathbf{d}_t + \mathbf{H}_t\mathbf{v}_t, \text{ where } \mathbf{V}_t \sim \text{MVN}(0, \mathbf{R}_t)$$

$$\mathbf{X}_1 \sim \text{MVN}(\mathbf{x0}, \mathbf{V0}) \text{ or } \mathbf{X}_0 \sim \text{MVN}(\mathbf{x0}, \mathbf{V0})$$

The parameters are everything except $\mathbf{x}$, $\mathbf{y}$, $\mathbf{v}$, $\mathbf{w}$, $\mathbf{c}$ and $\mathbf{d}$. $\mathbf{y}$ are data (missing values allowed). $\mathbf{c}$ and $\mathbf{d}$ are inputs (no missing values allowed). All parameters (except $\mathbf{x0}$ and $\mathbf{V0}$) can be time-varying but by default, all are time-constant (and the MARSS equation is generally written without the t subscripts on the parameter matrices). All parameters can be zero, including the variance matrices.

The parameter matrices can have fixed values and linear constraints. This is an example of a 3x3 matrix with linear constraints. All matrix elements can be written as a linear function of a, b, and c:

$$\begin{bmatrix} a+2b & 1 & a \\ 1+3a+b & 0 & b \\ 0 & -2 & c \end{bmatrix}$$

Values such as ab or a^2 or $log(a)$ are not linear constraints.

Usage

```
MARSS(y,
    model = NULL,
    inits = NULL,
    miss.value = as.numeric(NA),
    method = c("kem", "BFGS", "TMB", "BFGS_TMB", "nlminb_TMB"),
    form = c("marxss", "dfa", "marss"),
    fit = TRUE,
    silent = FALSE,
    control = NULL,
    fun.kf = c("MARSSkfas", "MARSSkfss"),
    ...)
```

Arguments

The default settings for the optional arguments are set in MARSSsettings.R and are given below in the details section. For form specific defaults see the form help file (e.g. MARSS.marxss or MARSS.dfa).

y	A n x T matrix of n time series over T time steps. Only y is required for the function. A ts object (univariate or multivariate) can be used and this will be converted to a matrix with time in the columns.
inits	A list with the same form as the list outputted by coef(fit) that specifies initial values for the parameters. See also MARSS.marxss.
model	Model specification using a list of parameter matrix text shortcuts or matrices. See Details and MARSS.marxss for the default form. Or better yet open the Quick Start Guide RShowDoc("Quick_Start",package="MARSS").
miss.value	Deprecated. Denote missing values by NAs in your data.
method	Estimation method. MARSS provides an EM algorithm (method="kem") (see MARSSkem) and the BFGS algorithm (method="BFGS") (see MARSSoptim).
form	The equation form used in the MARSS() call. The default is "marxss". See MARSS.marxss or MARSS.dfa.
fit	TRUE/FALSE Whether to fit the model to the data. If FALSE, a marssMLE object with only the model is returned.
silent	Setting to TRUE(1) suppresses printing of full error messages, warnings, progress bars and convergence information. Setting to FALSE(0) produces error output. Setting silent=2 will produce more verbose error messages and progress information.
fun.kf	What Kalman filter function to use. MARSS has two: MARSSkfas() which is based on the Kalman filter in the KFAS package based on Koopman and Durbin and MARSSkfss() which is a native R implementation of the Kalman filter and smoother in Shumway and Stoffer. The KFAS filter is much faster. MARSSkfas() modifies the input and output in order to output the lag-one covariance smoother needed for the EM algorithm (per page 321 in Shumway and Stoffer (2000).
control	Estimation options for the maximization algorithm. The typically used control options for method="kem" are below but see marssMLE for the full list of control options. Note many of these are not allowed if method="BFGS"; see MARSSoptim for the allowed control options for this method. minit The minimum number of iterations to do in the maximization routine (if needed by method). If method="kem", this is an easy way to up the iterations and see how your estimates are converging. (positive integer) maxit Maximum number of iterations to be used in the maximization routine (if needed by method) (positive integer). min.iter.conv.test Minimum iterations to run before testing convergence via the slope of the log parameter versus log iterations. conv.test.deltaT=9 Number of iterations to use for the testing convergence via the slope of the log parameter versus log iterations.

`conv.test.slope.tol` The slope of the log parameter versus log iteration to use as the cut-off for convergence. The default is 0.5 which is a bit high. For final analyses, this should be set lower. If you want to only use abstol as your convergence test, then to something very large, for example `conv.test.slope.tol=1000`. Type `MARSSinfo(11)` to see some comments on when you might want to do this.

`abstol` The logLik.(iter-1)-logLik.(iter) convergence tolerance for the maximization routine. To meet convergence both the abstol and slope tests must be passed.

`allow.degen` Whether to try setting $\mathbf{Q}$ or $\mathbf{R}$ elements to zero if they appear to be going to zero.

`trace` An integer specifying the level of information recorded and error-checking run during the algorithms. `trace=0`, specifies basic error-checking and brief error-messages; `trace>0` will print full error messages. In addition if trace>0, the Kalman filter output will be added to the outputted `marssMLE` object. Additional information recorded depends on the method of maximization. For the EM algorithm, a record of each parameter estimate for each EM iteration will be added. See `optim` for trace output details for the BFGS method. `trace=-1` will turn off most internal error-checking and most error messages. The internal error checks are time expensive so this can speed up model fitting. This is particularly useful for bootstrapping and simulation studies. It is also useful if you get an error saying that `MARSS()` stops in `MARSSkfss()` due to a `chol()` call. `MARSSkfss()` uses matrix inversions and for some models these are unstable (high condition value). `MARSSkfss()` is used for error-checks and does not need to be called normally.

`safe` Setting `safe=TRUE` runs the Kalman smoother after each parameter update rather than running the smoother only once after updated all parameters. The latter is faster but is not a strictly correct EM algorithm. In most cases, `safe=FALSE` (default) will not change the fits. If this setting does cause problems, you will know because you will see an error regarding the log-likelihood dropping and it will direct you to set `safe=TRUE`.

`...` Optional arguments passed to function specified by form.

Details

The `model` argument specifies the structure of your model. There is a one-to-one correspondence between how you would write your model in matrix form on the whiteboard and how you specify the model for `MARSS()`. Many different types of multivariate time-series models can be converted to the MARSS form. See the User Guide and Quick Start Guide for examples.

The MARSS package has two forms for standard users: marxss and dfa.

`MARSS.marxss` This is the default form. This is a MARSS model with (optional) inputs $\mathbf{c}_t$ or $\mathbf{d}_t$. Most users will want this help page.

`MARSS.dfa` This is a model form to allow easier specification of models for Dynamic Factor Analysis. The $\mathbf{Z}$ parameters has a specific form and the $\mathbf{Q}$ is set at i.i.d (diagonal) with variance of 1.

Those looking to modify or understand the base code, should look at MARSS.marss and MARSS.vectorized. These describe the forms used by the base functions. The EM algorithm uses the MARSS model written in vectorized form. This form is what allows linear constraints.

The likelihood surface for MARSS models can be multimodal or with strong ridges. It is recommended that for final analyses the estimates are checked by using a Monte Carlo initial conditions search; see the chapter on initial conditions searches in the User Guide. This requires more computation time, but reduces the chance of the algorithm terminating at a local maximum and not reaching the true MLEs. Also it is wise to check the EM results against the BFGS results (if possible) if there are strong ridges in the likelihood. Such ridges seems to slow down the EM algorithm considerably and can cause the algorithm to report convergence far from the maximum-likelihood values. EM steps up the likelihood and the convergence test is based on the rate of change of the log-likelihood in each step. Once on a strong ridge, the steps can slow dramatically. You can force the algorithm to keep working by setting minit. BFGS seems less hindered by the ridges but can be prodigiously slow for some multivariate problems. BFGS tends to work better if you give it good initial conditions (see Examples below for how to do this).

If you are working with models with time-varying parameters, it is important to notice the time-index for the parameters in the process equation (the $\mathbf{x}$ equation). In some formulations (e.g. in KFAS), the process equation is $\mathbf{x}_t = \mathbf{B}_{t-1}\mathbf{x}_{t-1} + \mathbf{w}_{t-1}$ so $\mathbf{B}_{t-1}$ goes with $\mathbf{x}_t$ not $\mathbf{B}_t$. Thus one needs to be careful to line up the time indices when passing in time-varying parameters to MARSS(). See the User Guide for examples.

Value

An object of class marssMLE. The structure of this object is discussed below, but if you want to know how to get specific output (like residuals, coefficients, smoothed states, confidence intervals, etc), see print.marssMLE(), tidy.marssMLE(), MARSSresiduals() and plot.marssMLE().

The outputted marssMLE object has the following components:

model	MARSS model specification. It is a marssMODEL object in the form specified by the user in the MARSS() call. This is used by print functions so that the user sees the expected form.
marss	The marssMODEL object in marss form. This form is needed for all the internal algorithms, thus is a required part of a marssMLE object.
call	All the information passed in in the MARSS() call.
start	List with specifying initial values that were used for each parameter matrix.
control	A list of estimation options, as specified by arguments control.
method	Estimation method.

If fit=TRUE, the following are also added to the marssMLE object. If fit=FALSE, a marssMLE object ready for fitting via the specified method is returned.

par	A list of estimated parameter values in marss form. Use print(), tidy() or coef() for outputing the model estimates in the MARSS() call (e.g. the default "marxss" form).
states	The expected value of $\mathbf{X}$ conditioned on all the data, i.e. smoothed states.
states.se	The standard errors of the expected value of $\mathbf{X}$.

ytT	The expected value of $\mathbf{Y}$ conditioned on all the data. Note this is just y for those y that are not missing.
ytT.se	The standard errors of the expected value of $\mathbf{Y}$. Note this is 0 for any non-missing y.
numIter	Number of iterations required for convergence.
convergence	Convergence status. 0 means converged successfully, 3 means all parameters were fixed (so model did not need to be fit) and -1 means call was made with fit=FALSE and parameters were not fixed (thus no $par element and Kalman filter/smoother cannot be run). Anything else is a warning or error. 2 means the marssMLE object has an error; the object is returned so you can debug it. The other numbers are errors during fitting. The error code depends on the fitting method. See MARSSkem and MARSSoptim.
logLik	Log-likelihood.
AIC	Akaike's Information Criterion.
AICc	Sample size corrected AIC.

If control$trace is set to 1 or greater, the following are also added to the marssMLE object.

kf	A list containing Kalman filter/smoother output from MARSSkf(). This is not normally added to a marssMLE object since it is verbose, but can be added using MARSSkf().
Ey	A list containing output from MARSShatyt. This isn't normally added to a marssMLE object since it is verbose, but can be computed using MARSShatyt().

Author(s)

Eli Holmes, Eric Ward and Kellie Wills, NOAA, Seattle, USA.

References

The MARSS User Guide: Holmes, E. E., E. J. Ward, and M. D. Scheuerell (2012) Analysis of multivariate time-series using the MARSS package. NOAA Fisheries, Northwest Fisheries Science Center, 2725 Montlake Blvd E., Seattle, WA 98112 Type RShowDoc("UserGuide",package="MARSS") to open a copy.

Holmes, E. E. (2012). Derivation of the EM algorithm for constrained and unconstrained multivariate autoregressive state-space (MARSS) models. Technical Report. arXiv:1302.3919 [stat.ME]

Holmes, E. E., E. J. Ward and K. Wills. (2012) MARSS: Multivariate autoregressive state-space models for analyzing time-series data. R Journal 4: 11-19.

See Also

marssMLE, MARSSkem(), MARSSoptim(), MARSSkf(), MARSS-package, print.marssMLE(), plot.marssMLE(), print.marssMODEL(), MARSS.marxss(), MARSS.dfa(), fitted(), residuals(), MARSSresiduals(), predict(), tsSmooth(), tidy(), coef()

Examples

```
dat <- t(harborSealWA)
dat <- dat[2:4, ] # remove the year row
# fit a model with 1 hidden state and 3 observation time series
kemfit <- MARSS(dat, model = list(
  Z = matrix(1, 3, 1),
  R = "diagonal and equal"
))
kemfit$model # This gives a description of the model
print(kemfit$model) # same as kemfit$model
summary(kemfit$model) # This shows the model structure

# add CIs to a marssMLE object
# default uses an estimated Hessian matrix
kem.with.hess.CIs <- MARSSparamCIs(kemfit)
kem.with.hess.CIs

# fit a model with 3 hidden states (default)
kemfit <- MARSS(dat, silent = TRUE) # suppress printing
kemfit

# Fit the above model with BFGS using a short EM fit as initial conditions
kemfit <- MARSS(dat, control=list(minit=5, maxit=5))
bffit <- MARSS(dat, method="BFGS", inits=kemfit)

# fit a model with 3 correlated hidden states
# with one variance and one  covariance
# maxit set low to speed up example, but more iters are needed for convergence
kemfit <- MARSS(dat, model = list(Q = "equalvarcov"), control = list(maxit = 50))
# use Q="unconstrained" to allow different variances and covariances

# fit a model with 3 independent hidden states
# where each observation time series is independent
# the hidden trajectories 2-3 share their U parameter
kemfit <- MARSS(dat, model = list(U = matrix(c("N", "S", "S"), 3, 1)))

# same model, but with fixed independent observation errors
# and the 3rd x processes are forced to have a U=0
# Notice how a list matrix is used to combine fixed and estimated elements
# all parameters can be specified in this way using list matrices
kemfit <- MARSS(dat, model = list(U = matrix(list("N", "N", 0), 3, 1), R = diag(0.01, 3)))

# fit a model with 2 hidden states (north and south)
# where observation time series 1-2 are north and 3 is south
# Make the hidden state process independent with same process var
# Make the observation errors different but independent
# Make the growth parameters (U) the same
# Create a Z matrix as a design matrix that assigns the "N" state to the first 2 rows of dat
# and the "S" state to the 3rd row of data
Z <- matrix(c(1, 1, 0, 0, 0, 1), 3, 2)
# You can use factor is a shortcut making the above design matrix for Z
# Z <- factor(c("N","N","S"))
```

```
# name the state vectors
colnames(Z) <- c("N", "S")
kemfit <- MARSS(dat, model = list(
  Z = Z,
  Q = "diagonal and equal", R = "diagonal and unequal", U = "equal"
))

# print the model followed by the marssMLE object
kemfit$model

## Not run:
# simulate some new data from our fitted model
sim.data <- MARSSsimulate(kemfit, nsim = 10, tSteps = 10)

# Compute bootstrap AIC for the model; this takes a long, long time
kemfit.with.AICb <- MARSSaic(kemfit, output = "AICbp")
kemfit.with.AICb

## End(Not run)

## Not run:
# Many more short examples can be found in the
# Quick Examples chapter in the User Guide
RShowDoc("UserGuide", package = "MARSS")

# You can find the R scripts from the chapters by
# going to the index page
RShowDoc("index", package = "MARSS")

## End(Not run)
```

MARSS.dfa *Multivariate Dynamic Factor Analysis*

Description

The Dynamic Factor Analysis model in MARSS is The argument `form="marxss"` in a `MARSS()` function call specifies a MAR-1 model with eXogenous variables model. This is a MARSS(1) model of the form:

$$\mathbf{x}_t = \mathbf{x}_{t-1} + \mathbf{w}_t, \text{ where } \mathbf{W}_t \sim \text{MVN}(0, \mathbf{I})$$
$$\mathbf{y}_t = \mathbf{Z}_t\mathbf{x}_t + \mathbf{D}_t\mathbf{d}_t + \mathbf{v}_t, \text{ where } \mathbf{V}_t \sim \text{MVN}(0, \mathbf{R}_t)$$
$$\mathbf{X}_1 \sim \text{MVN}(\mathbf{x0}, 5\mathbf{I})$$

Note, by default $\mathbf{x}_1$ is treated as a diffuse prior.

Passing in `form="dfa"` to `MARSS()` invokes a helper function to create that model and creates the $\mathbf{Z}$ matrix for the user. $\mathbf{Q}$ is by definition identity, $\mathbf{x}_0$ is zero and $\mathbf{V_0}$ is diagonal with large variance (5). $\mathbf{u}$ is zero, $\mathbf{a}$ is zero, and covariates only enter the $\mathbf{y}$ equation. Because $\mathbf{u}$ and $\mathbf{a}$ are 0, the data should have mean 0 (demeaned) otherwise one is likely to be creating a structurally inadequate model (i.e. the model implies that the data have mean = 0, yet data do not have mean = 0).

Arguments

Some arguments are common to all forms: "y" (data), "inits", "control", "method", "form", "fit", "silent", "fun.kf". See MARSS for information on these arguments.

In addition to these, form="dfa" has some special arguments that can be passed in:

- `demean` Logical. Default is TRUE, which means the data will be demeaned.
- `z.score` Logical. Default is TRUE, which means the data will be z-scored (demeaned and variance standardized to 1).
- `covariates` Covariates (d) for the y equation. No missing values allowed and must be a matrix with the same number of time steps as the data. An unconstrained D matrix will estimated.

The `model` argument of the `MARSS()` call is constrained in terms of what parameters can be changed and how they can be changed. See details below. An additional element, `m`, can be passed into the `model` argument that specifies the number of hidden state variables. It is not necessarily for the user to specify `Z` as the helper function will create a `Z` appropriate for a DFA model.

Details

The `model` argument is a list. The following details what list elements can be passed in:

- `B` "Identity". The standard (and default) DFA model has B="identity". However it can be "identity", "diagonal and equal", "diagonal and unequal" or a time-varying fixed or estimated diagonal matrix.
- `U` "Zero". Cannot be changed or passed in via model argument.
- `Q` "Identity". The standard (and default) DFA model has Q="identity". However, it can be "identity", "diagonal and equal", "diagonal and unequal" or a time-varying fixed or estimated diagonal matrix.
- `Z` Can be passed in as a (list) matrix if the user does not want a default DFA `Z` matrix. There are many equivalent ways to construct a DFA `Z` matrix. The default is Zuur et al.'s form (see User Guide).
- `A` Default="zero". Can be "unequal", "zero" or a matrix.
- `R` Default="diagonal and equal". Can be set to "identity", "zero", "unconstrained", "diagonal and unequal", "diagonal and equal", "equalvarcov", or a (list) matrix to specify general forms.
- `x0` Default="zero". Can be "unconstrained", "unequal", "zero", or a (list) matrix.
- `V0` Default=diagonal matrix with 5 on the diagonal. Can be "identity", "zero", or a matrix.
- `tinitx` Default=0. Can be 0 or 1. Tells MARSS whether x0 is at t=0 or t=1.
- `m` Default=1. Can be 1 to n (the number of y time-series). Must be integer.

See the User Guide chapter on Dynamic Factor Analysis for examples of of using `form="dfa"`.

Value

A object of class `marssMLE`. See `print()` for a discussion of the various output available for `marssMLE` objects (coefficients, residuals, Kalman filter and smoother output, imputed values for missing data, etc.). See `MARSSsimulate()` for simulating from `marssMLE` objects. `MARSSboot()` for bootstrapping, `MARSSaic()` for calculation of various AIC related model selection metrics, and `MARSSparamCIs()` for calculation of confidence intervals and bias.

Usage

```
MARSS(y, inits = NULL, model = NULL, miss.value = as.numeric(NA), method = "kem", form
= "dfa", fit = TRUE, silent = FALSE, control = NULL, fun.kf = "MARSSkfas", demean = TRUE,
z.score = TRUE)
```

Author(s)

Eli Holmes, NOAA, Seattle, USA.

References

The MARSS User Guide: Holmes, E. E., E. J. Ward, and M. D. Scheuerell (2012) Analysis of multivariate time-series using the MARSS package. NOAA Fisheries, Northwest Fisheries Science Center, 2725 Montlake Blvd E., Seattle, WA 98112 Type `RShowDoc("UserGuide",package="MARSS")` to open a copy.

See Also

`MARSS()`, `MARSS.marxss()`

Examples

```
## Not run:
dat <- t(harborSealWA[,-1])
# DFA with 3 states; used BFGS because it fits much faster for this model
fit <- MARSS(dat, model = list(m=3), form="dfa", method="BFGS")

# See the Dynamic Factor Analysis chapter in the User Guide
RShowDoc("UserGuide", package = "MARSS")

## End(Not run)
```

MARSS.marss *Multivariate AR-1 State-space Model*

Description

The form of MARSS models for users is "marxss", the MARSS models with inputs. See `MARSS.marxss`. In the internal algorithms (e.g. `MARSSkem`), the "marss" form is used and the $\mathbf{D}\mathbf{d}_t$ are incorporated into the $\mathbf{a}_t$ matrix and $\mathbf{C}\mathbf{c}_t$ are incorporated into the $\mathbf{u}_t$. The $\mathbf{a}$ and $\mathbf{u}$ matrices then become time-varying if the model includes $\mathbf{d}_t$ and $\mathbf{c}_t$.

This is a MARSS(1) model of the marss form:

$$\mathbf{x}_t = \mathbf{B}\mathbf{x}_{t-1} + \mathbf{u}_t + \mathbf{G}\mathbf{w}_t, \text{ where } \mathbf{W}_t \sim \text{MVN}(0, \mathbf{Q})$$

$$\mathbf{y}_t = \mathbf{Z}\mathbf{x}_t + \mathbf{a}_t + \mathbf{H}\mathbf{v}_t, \text{ where } \mathbf{V}_t \sim \text{MVN}(0, \mathbf{R})$$

$$\mathbf{X}_1 \sim \text{MVN}(\mathbf{x0}, \mathbf{V0}) \text{ or } \mathbf{X}_0 \sim \text{MVN}(\mathbf{x0}, \mathbf{V0})$$

Note, by default $\mathbf{V0}$ is a matrix of all zeros and thus $\mathbf{x}_1$ or $\mathbf{x}_0$ is treated as an estimated parameter not a diffuse prior. To remove clutter, the rest of the parameters are shown as time-constant (no t subscript) but all parameters can be time-varying.

Note, "marss" is a model form. A model form is defined by a collection of form functions discussed in `marssMODEL`. These functions are not exported to the user, but are called by `MARSS()` using the argument `form`. These internal functions convert the users model list into the vec form of a MARSS model and do extensive error-checking.

Details

See the help page for the `MARSS.marxss` form for details.

Value

A object of class `marssMLE`.

Usage

```
MARSS(y, inits = NULL, model = NULL, miss.value = as.numeric(NA), method = "kem", form
= "marxss", fit = TRUE, silent = FALSE, control = NULL, fun.kf = "MARSSkfas", ...)
```

Author(s)

Eli Holmes, NOAA, Seattle, USA.

See Also

`marssMODEL`, `MARSS.marxss()`

Examples

```
## Not run:
# See the MARSS man page for examples
?MARSS

# and the Quick Examples chapter in the User Guide
RShowDoc("UserGuide", package = "MARSS")

## End(Not run)
```

`MARSS.marxss` *Multivariate AR-1 State-space Model with Inputs*

Description

The argument `form="marxss"` in a `MARSS()` function call specifies a MAR-1 model with eXogenous variables model. This is a MARSS(1) model of the form:

$$\mathbf{x}_t = \mathbf{B}_t\mathbf{x}_{t-1} + \mathbf{u}_t + \mathbf{C}_t\mathbf{c}_t + \mathbf{G}_t\mathbf{w}_t, \text{ where } \mathbf{W}_t \sim \text{MVN}(0, \mathbf{Q}_t)$$

$$\mathbf{y}_t = \mathbf{Z}_t\mathbf{x}_t + \mathbf{a}_t + \mathbf{D}_t\mathbf{d}_t + \mathbf{H}_t\mathbf{v}_t, \text{ where } \mathbf{V}_t \sim \text{MVN}(0, \mathbf{R}_t)$$

$$\mathbf{X}_1 \sim \text{MVN}(\mathbf{x0}, \mathbf{V0}) \text{ or } \mathbf{X}_0 \sim \text{MVN}(\mathbf{x0}, \mathbf{V0})$$

Note, by default $\mathbf{V0}$ is a matrix of all zeros and thus $\mathbf{x}_1$ or $\mathbf{x}_0$ is treated as an estimated parameter not a diffuse prior.

Note, "marxss" is a model form. A model form is defined by a collection of form functions discussed in `marssMODEL`. These functions are not exported to the user, but are called by `MARSS()` using the argument `form`.

Details

The allowed arguments when `form="marxss"` are 1) the arguments common to all forms: "y" (data), "inits", "control", "method", "form", "fit", "silent", "fun.kf" (see `MARSS` for information on these arguments) and 2) the argument "model" which is a list describing the MARXSS model (the model list is described below). See the Quick Start Guide guide or the User Guide for examples.

The argument `model` must be a list. The elements in the list specify the structure for the $\mathbf{B}$, $\mathbf{u}$, $\mathbf{C}$, $\mathbf{c}$, $\mathbf{Q}$, $\mathbf{Z}$, $\mathbf{a}$, $\mathbf{D}$, $\mathbf{d}$, $\mathbf{R}$, $\mathbf{x}_0$, and $\mathbf{V}_0$ in the MARXSS model (above). The list elements can have the following values:

`Z` Default="identity". A text string, "identity","unconstrained", "diagonal and unequal", "diagonal and equal", "equalvarcov", or "onestate", or a length n vector of factors specifying which of the m hidden state time series correspond to which of the n observation time series. May be specified as a n x m list matrix for general specification of both fixed and shared elements within the matrix. May also be specified as a numeric n x m matrix to use a custom fixed $\mathbf{Z}$. "onestate" gives a n x 1 matrix of 1s. "identity","unconstrained", "diagonal and unequal", "diagonal and equal", and "equalvarcov" all specify n x n matrices.

`B` Default="identity". A text string, "identity", "unconstrained", "diagonal and unequal", "diagonal and equal", "equalvarcov", "zero". Can also be specified as a list matrix for general specification of both fixed and shared elements within the matrix. May also be specified as a numeric m x m matrix to use custom fixed $\mathbf{B}$, but in this case all the eigenvalues of $\mathbf{B}$ must fall in the unit circle.

`U, x0` Default="unconstrained". A text string, "unconstrained", "equal", "unequal" or "zero". May be specified as a m x 1 list matrix for general specification of both fixed and shared elements within the matrix. May also be specified as a numeric m x 1 matrix to use a custom fixed $\mathbf{u}$ or $\mathbf{x}_0$. Notice that `U` is capitalized in the `model` argument and output lists.

`A` Default="scaling". A text string, "scaling","unconstrained", "equal", "unequal" or "zero". May be specified as a n x 1 list matrix for general specification of both fixed and shared elements within the matrix. May also be specified as a numeric n x 1 matrix to use a custom fixed $\mathbf{a}$. Care must be taken when specifying `A` so that the model is not under-constrained and unsolvable model. The default, "scaling", only applies to $\mathbf{Z}$ matrices that are design matrices (only 1s and 0s and all rows sum to 1). When a column in $\mathbf{Z}$ has multiple 1s, the first row

in the $\mathbf{a}$ matrix associated with those $\mathbf{Z}$ rows is 0 and the other associated $\mathbf{a}$ rows have an estimated value. This is used to treat $\mathbf{a}$ as an intercept where one intercept for each $\mathbf{x}$ (hidden state) is fixed at 0 and any other intercepts associated with that $\mathbf{x}$ have an estimated intercept. This ensures a solvable model when $\mathbf{Z}$ is a design matrix. Note in the model argument and output, `A` is capitalized.

`Q` Default="diagonal and unequal". A text string, "identity", "unconstrained", "diagonal and unequal", "diagonal and equal", "equalvarcov", "zero". May be specified as a list matrix for general specification of both fixed and shared elements within the matrix. May also be specified as a numeric g x g matrix to use a custom fixed matrix. Default value of g is m, so $\mathbf{Q}$ is a m x m matrix. g is the number of columns in $\mathbf{G}$ (below).

`R` Default="diagonal and equal". A text string, "identity", "unconstrained", "diagonal and unequal", "diagonal and equal", "equalvarcov", "zero". May be specified as a list matrix for general specification of both fixed and shared elements within the matrix. May also be specified as a numeric h x h matrix to use a custom fixed matrix. Default value of h is n, so $\mathbf{R}$ is a n x n matrix. h is the num of columns in $\mathbf{H}$ (below).

`V0` Default="zero". A text string, "identity", "unconstrained", "diagonal and unequal", "diagonal and equal", "equalvarcov", "zero". May be specified as a list matrix for general specification of both fixed and shared elements within the matrix. May also be specified as a numeric m x m matrix to use a custom fixed matrix.

`D` **and** `C` Default="zero". A text string, "identity", "unconstrained", "diagonal and unequal", "diagonal and equal", "equalvarcov", "zero". Can be specified as a list matrix for general specification of both fixed and shared elements within the matrix. May also be specified as a numeric matrix to use custom fixed values. Must have n rows ($\mathbf{D}$) or m rows ($\mathbf{C}$).

`d` **and** `c` Default="zero". Numeric matrix. No missing values allowed. Must have 1 column or the same number of columns as the data, $\mathbf{y}$. The numbers of rows in $\mathbf{d}$ must be the same as number of columns in $\mathbf{D}$; similarly for $\mathbf{c}$ and $\mathbf{C}$.

`G` **and** `H` Default="identity". A text string, "identity". Can be specified as a numeric matrix or array for time-varying cases. Must have m rows and g columns ($\mathbf{G}$) or n rows and h columns ($\mathbf{H}$). g is the dim of $\mathbf{Q}$ and h is the dim of $\mathbf{R}$.

`tinitx` Default=0. Whether the initial state is specified at t=0 (default) or t=1.

All parameters except $\mathbf{x}_0$ and $\mathbf{V}_0$ may be time-varying. If time-varying, then text shortcuts cannot be used. Enter as an array with the 3rd dimension being time. Time dimension must be 1 or equal to the number of time-steps in the data. See Quick Start guide (`RShowDoc("Quick_Start",package="MARSS")`) or the User Guide (`RShowDoc("UserGuide",package="MARSS")`) for examples.Valid model structures for `method="BFGS"` are the same as for `method="kem"`. See `MARSSoptim()` for the allowed options for this method.

The default estimation method, `method="kem"`, is the EM algorithm described in the MARSS User Guide. The default settings for the control and inits arguments are set via `MARSS:::alldefaults$kem` in `MARSSsettings.R`. The defaults for the model argument are set in `MARSS_marxss.R` For this method, they are:

- inits = list(B=1, U=0, Q=0.05, Z=1, A=0, R=0.05, x0=-99, V0=0.05, G=0, H=0, L=0, C=0, D=0, c=0, d=0)
- model = list(Z="identity", A="scaling", R="diagonal and equal", B="identity", U="unconstrained", Q="diagonal and unequal", x0="unconstrained", V0="zero", C="zero",D="zero",c=matrix(0,0,1), d=matrix(0,0,1), tinitx=0, diffuse=FALSE)

- control=list(minit=15, maxit=500, abstol=0.001, trace=0, sparse=FALSE, safe=FALSE, allow.degen=TRUE, min.degen.iter=50, degen.lim=1.0e-04, min.iter.conv.test=15, conv.test.deltaT=9, conv.test.slope.tol= 0.5, demean.states=FALSE) You can read about these in `MARSS()`. If you want to speed up your fits, you can turn off most of the model checking using `trace=-1`.
- fun.kf = "MARSSkfas"; This sets the Kalman filter function to use. `MARSSkfas()` is generally more stable as it uses Durban & Koopman's algorithm. But it may dramatically slow down when the data set is large (more than 10 rows of data). Try the classic Kalman filter algorithm to see if it runs faster by setting `fun.kf="MARSSkfss"`. You can read about the two algorithms in `MARSSkf`.

For `method="BFGS"`, type `MARSS:::alldefaults$BFGS` to see the defaults.

Value

A object of class `marssMLE`. See `print.marssMLE` for a discussion of the various output available for `marssMLE` objects (coefficients, residuals, Kalman filter and smoother output, imputed values for missing data, etc.). See `MARSSsimulate` for simulating from `marssMLE` objects. `MARSSboot` for bootstrapping, `MARSSaic` for calculation of various AIC related model selection metrics, and `MARSSparamCIs` for calculation of confidence intervals and bias. See `plot.marssMLE` for some default plots of a model fit.

Usage

```
MARSS(y, inits = NULL, model = NULL, miss.value = as.numeric(NA), method = "kem", form
= "marxss", fit = TRUE, silent = FALSE, control = NULL, fun.kf = "MARSSkfas", ...)
```

Author(s)

Eli Holmes, NOAA, Seattle, USA.

See Also

`marssMODEL`, `MARSS.dfa()`

Examples

```
## Not run:
#See the MARSS man page for examples
?MARSS

#and the Quick Examples chapter in the User Guide
RShowDoc("UserGuide",package="MARSS")

## End(Not run)
```

MARSS.vectorized *Vectorized Multivariate AR-1 State-space Model*

Description

The EM algorithm (MARSSkem) in the MARSS package works by converting the more familiar MARSS model in matrix form into the vectorized form which allows general linear constraints (Holmes 2012). The vectorized form is:

$$\mathbf{x}(t) = (\mathbf{x}(t-1)^\top \otimes \mathbf{I}_m)(\mathbf{f}_b(t)+\mathbf{D}_b(t)\beta)+(\mathbf{f}_u(t)+\mathbf{D}_u(t)\upsilon)+\mathbf{w}(t), \text{ where } \mathbf{W}(t) \sim \text{MVN}(0, \mathbf{Q}(t))$$

$$\mathbf{y}(t) = (\mathbf{x}(t)^\top \otimes \mathbf{I}_n)(\mathbf{f}_z(t) + \mathbf{D}_z(t)\zeta) + (\mathbf{f}_a(t) + \mathbf{D}_a(t)\alpha) + \mathbf{v}(t), \text{ where } \mathbf{V}(t) \sim \text{MVN}(0, \mathbf{R}(t))$$

$$\mathbf{x}(1) \sim \text{MVN}(x0, V0) \text{ or } \mathbf{x}(0) \sim \text{MVN}(x0, V0)$$

where β, υ, ζ, and α are column vectors of estimated values, the $\mathbf{f}$ are column vectors of inputs (fixed values), and the $\mathbf{D}$ are perturbation matrices that align the estimated values into the right rows. The $\mathbf{f}$ and $\mathbf{D}$ are potentially time-varying. $\otimes$ means kronecker product and $\mathbf{I}_p$ is a p x p identity matrix.

Normally the user will specify their model in "marxss" form, perhaps with text short-cuts. The "marxss" form is then converted to "marss" form using the conversion function `marxss_to_marss()`. In "marss" form, the D, d, C, and c information is put in A and U respectively. If there are inputs (d and c), then this will make A and U time-varying. This is unfortunate, because this slows down the EM algorithm considerably due to the unfortunate decision (early on) to store time-varying parameters as 3-dimensional. The functions for the "marss" form (in the file `MARSS_marss.R`) convert the "marss" form model into vectorized form and prepares the f (fixed) and D (free) matrices that are at the heart of the model specification.

Note, "marss" is a model form. A model form is defined by a collection of form functions discussed in marssMODEL. These functions are not exported to the user, but are called by `MARSS()` using the argument `form`. These internal functions convert the users model list into the vectorized form of a MARSS model and do extensive error-checking. "marxss" is also a model form and these models are also stored in vectorized form (See examples below).

Details

See Holmes (2012) for a discussion of MARSS models in vectorized form.

Author(s)

Eli Holmes, NOAA, Seattle, USA.

References

Holmes, E. E. (2012). Derivation of the EM algorithm for constrained and unconstrained multivariate autoregressive state-space (MARSS) models. Technical Report. arXiv:1302.3919 [stat.ME]

See Also

marssMODEL, `MARSS.marss()`, `MARSS.marxss()`

Examples

```
dat <- t(harborSealWA)
dat <- dat[2:4, ]
MLEobj <- MARSS(dat)

# free (D) and fixed (f) matrices
names(MLEobj$model$free)
names(MLEobj$model$fixed)
# In marss form, the D, C, d, and c matrices are found in A and U
# If there are inputs, this makes U time-varying
names(MLEobj$marss$free)
names(MLEobj$marss$fixed)

# par is in marss form so does not have values for D, C, d, or c
names(MLEobj$par)
# if you need the par in marxss form, you can use print
tmp <- print(MLEobj, what="par", form="marxss", silent=TRUE)
names(tmp)
```

MARSSaic *AIC for MARSS Models*

Description

Calculates AIC, AICc, a parametric bootstrap AIC (AICbp) and a non-parametric bootstrap AIC (AICbb). If you simply want the AIC value for a `marssMLE` object, you can use `AIC(fit)`.

Usage

```
MARSSaic(MLEobj, output = c("AIC", "AICc"),
    Options = list(nboot = 1000, return.logL.star = FALSE,
    silent = FALSE))
```

Arguments

`MLEobj`	An object of class `marssMLE`. This object must have a `$par` element containing MLE parameter estimates from e.g. `MARSSkem()`.
`output`	A vector containing one or more of the following: "AIC", "AICc", "AICbp", "AICbb", "AICi", "boot.params". See Details.
`Options`	A list containing: • `nboot` Number of bootstraps (positive integer) • `return.logL.star` Return the log-likelihoods for each bootstrap? (T/F) • `silent` Suppress printing of the progress bar during AIC bootstraps? (T/F)

Details

When sample size is small, Akaike's Information Criterion (AIC) under-penalizes more complex models. The most commonly used small sample size corrector is AICc, which uses a penalty term of $Kn/(n-K-1)$, where K is the number of estimated parameters. However, for time series models, AICc still under-penalizes complex models; this is especially true for MARSS models.

Two small-sample estimators specific for MARSS models have been developed. Cavanaugh and Shumway (1997) developed a variant of bootstrapped AIC using Stoffer and Wall's (1991) bootstrap algorithm ("AICbb"). Holmes and Ward (2010) developed a variant on AICb ("AICbp") using a parametric bootstrap. The parametric bootstrap permits AICb calculation when there are missing values in the data, which Cavanaugh and Shumway's algorithm does not allow. More recently, Bengtsson and Cavanaugh (2006) developed another small-sample AIC estimator, AICi, based on fitting candidate models to multivariate white noise.

When the `output` argument passed in includes both `"AICbp"` and `"boot.params"`, the bootstrapped parameters from `"AICbp"` will be added to `MLEobj`.

Value

Returns the `marssMLE` object that was passed in with additional AIC components added on top as specified in the 'output' argument.

Author(s)

Eli Holmes, NOAA, Seattle, USA.

References

Holmes, E. E., E. J. Ward, and M. D. Scheuerell (2012) Analysis of multivariate time-series using the MARSS package. NOAA Fisheries, Northwest Fisheries Science Center, 2725 Montlake Blvd E., Seattle, WA 98112 Type `RShowDoc("UserGuide",package="MARSS")` to open a copy.

Bengtsson, T., and J. E. Cavanaugh. 2006. An improved Akaike information criterion for state-space model selection. Computational Statistics & Data Analysis 50:2635-2654.

Cavanaugh, J. E., and R. H. Shumway. 1997. A bootstrap variant of AIC for state-space model selection. Statistica Sinica 7:473-496.

See Also

`MARSSboot()`

Examples

```
dat <- t(harborSealWA)
dat <- dat[2:3, ]
kem <- MARSS(dat, model = list(
  Z = matrix(1, 2, 1),
  R = "diagonal and equal"
))
kemAIC <- MARSSaic(kem, output = c("AIC", "AICc"))
```

MARSSboot *Bootstrap MARSS Parameter Estimates*

Description

Creates bootstrap parameter estimates and simulated (or bootstrapped) data (if appropriate). This is a base function in the MARSS-package.

Usage

```
MARSSboot(MLEobj, nboot = 1000,
    output = "parameters", sim = "parametric",
    param.gen = "MLE", control = NULL, silent = FALSE)
```

Arguments

MLEobj	An object of class marssMLE. Must have a $par element containing MLE parameter estimates.
nboot	Number of bootstraps to perform.
output	Output to be returned: "data", "parameters" or "all".
sim	Type of bootstrap: "parametric" or "innovations". See Details.
param.gen	Parameter generation method: "hessian" or "MLE".
control	The options in MLEobj$control are used by default. If supplied here, must contain all of the following: max.iter Maximum number of EM iterations. tol Optional tolerance for log-likelihood change. If log-likelihood decreases less than this amount relative to the previous iteration, the EM algorithm exits. allow.degen Whether to try setting $\mathbf{Q}$ or $\mathbf{R}$ elements to zero if they appear to be going to zero.
silent	Suppresses printing of progress bar.

Details

Approximate confidence intervals (CIs) on the model parameters can be calculated by the observed Fisher Information matrix (the Hessian of the negative log-likelihood function). The Hessian CIs (param.gen="hessian") are based on the asymptotic normality of ML estimates under a large-sample approximation. CIs that are not based on asymptotic theory can be calculated using parametric and non-parametric bootstrapping (param.gen="MLE"). In this case, parameter estimates are generated by the ML estimates from each bootstrapped data set. The MLE method (kem or BFGS) is determined by MLEobj$method.

Stoffer and Wall (1991) present an algorithm for generating CIs via a non-parametric bootstrap for state-space models (sim = "innovations"). The basic idea is that the Kalman filter can be used to generate estimates of the residuals of the model fit. These residuals are then standardized

and resampled and used to generate bootstrapped data using the MARSS model and its maximum-likelihood parameter estimates. One of the limitations of the Stoffer and Wall algorithm is that it cannot be used when there are missing data, unless all data at time t are missing. An alternative approach is a parametric bootstrap (`sim = "parametric"`), in which the ML parameter estimates are used to produce bootstrapped data directly from the state-space model.

Value

A list with the following components:

`boot.params`	Matrix (number of params x nboot) of parameter estimates from the bootstrap.
`boot.data`	Array (n x t x nboot) of simulated (or bootstrapped) data (if requested and appropriate).
`marss`	The `marssMODEL` object (form="marss") that was passed in via `MLEobj$marss`.
`nboot`	Number of bootstraps performed.
`output`	Type of output returned.
`sim`	Type of bootstrap.
`param.gen`	Parameter generation method: "hessian" or "KalmanEM".

Author(s)

Eli Holmes and Eric Ward, NOAA, Seattle, USA.

References

Holmes, E. E., E. J. Ward, and M. D. Scheuerell (2012) Analysis of multivariate time-series using the MARSS package. NOAA Fisheries, Northwest Fisheries Science Center, 2725 Montlake Blvd E., Seattle, WA 98112 Type `RShowDoc("UserGuide",package="MARSS")` to open a copy.

Stoffer, D. S., and K. D. Wall. 1991. Bootstrapping state-space models: Gaussian maximum likelihood estimation and the Kalman filter. Journal of the American Statistical Association 86:1024-1033.

Cavanaugh, J. E., and R. H. Shumway. 1997. A bootstrap variant of AIC for state-space model selection. Statistica Sinica 7:473-496.

See Also

`marssMLE`, `marssMODEL`, `MARSSaic()`, `MARSShessian()`, `MARSSFisherI()`

Examples

```
# nboot is set low in these examples in order to run quickly
# normally nboot would be >1000 at least
dat <- t(kestrel)
dat <- dat[2:3, ]
# maxit set low to speed up the example
kem <- MARSS(dat,
  model = list(U = "equal", Q = diag(.01, 2)),
  control = list(maxit = 50)
```

```
)
# bootstrap parameters from a Hessian matrix
hess.list <- MARSSboot(kem, param.gen = "hessian", nboot = 4)

# from resampling the innovations (no missing values allowed)
boot.innov.list <- MARSSboot(kem, output = "all", sim = "innovations", nboot = 4)

# bootstrapped parameter estimates
hess.list$boot.params
```

MARSScv *MARSScv is a wrapper for MARSS that re-fits the model with cross validated data.*

Description

MARSScv is a wrapper for MARSS that re-fits the model with cross validated data.

Usage

```
MARSScv(
  y,
  model = NULL,
  inits = NULL,
  method = "kem",
  form = "marxss",
  silent = FALSE,
  control = NULL,
  fun.kf = c("MARSSkfas", "MARSSkfss"),
  fold_ids = NULL,
  future_cv = FALSE,
  n_future_cv = floor(ncol(y)/3),
  interval = "confidence",
  ...
)
```

Arguments

y	A n x T matrix of n time series over T time steps. Only y is required for the function. A ts object (univariate or multivariate) can be used and this will be converted to a matrix with time in the columns.
model	Model specification using a list of parameter matrix text shortcuts or matrices. See Details and MARSS.marxss() for the default form. Or better yet open the Quick Start Guide RShowDoc("Quick_Start",package="MARSS")
inits	A list with the same form as the list output by coef(fit) that specifies initial values for the parameters. See also MARSS.marxss().

method	Estimation method. MARSS provides an EM algorithm (method="kem") (see MARSSkem()) and the BFGS algorithm (method="BFGS") (see MARSSoptim()). Fast TMB fitting provided by the companion package marssTMB.
form	The equation form used in the MARSS() call. The default is "marxss". See MARSS.marxss() or MARSS.dfa()
silent	Setting to TRUE(1) suppresses printing of full error messages, warnings, progress bars and convergence information. Setting to FALSE(0) produces error output. Setting silent=2 will produce more verbose error messages and progress information.
control	Estimation options for the maximization algorithm. The typically used control options for method="kem" are below but see marssMLE for the full list of control options. Note many of these are not allowed if method="BFGS"; see MARSSoptim() for the allowed control options for this method.
fun.kf	What Kalman filter function to use. MARSS has two: MARSSkfas() which is based on the Kalman filter in the KFAS package based on Koopman and Durbin and MARSSkfss() which is a native R implementation of the Kalman filter and smoother in Shumway and Stoffer. The KFAS filter is much faster. MARSSkfas() modifies the input and output in order to output the lag-one covariance smoother needed for the EM algorithm (per page 321 in Shumway and Stoffer (2000).
fold_ids	A n x T matrix of integers, with values assigned by the user to folds. If not included, data are randomly assigned to one of 10 folds
future_cv	Whether or not to use future cross validation (defaults to FALSE), where data up to time T-1 are used to predict data at time T. Data are held out by time slices, and the fold_ids argument is ignored.
n_future_cv	Number of time slices to hold out for future cross validation. Defaults to floor(n_future_cv/3). Predictions are made for the last n_future_cv time steps
interval	uncertainty interval for prediction. Can be one of "confidence" or "prediction", and defaults to "confidence"
...	not used

Value

A list object, containing cv_pred (a matrix of predictions), cv_se (a matrix of SEs), fold_ids (a matrix of fold ids used as data), and df (a dataframe containing the original data, predictions, SEs, and folds)

Examples

```
dat <- t(harborSealWA)
dat <- dat[2:4, ] # remove the year row
# fit a model with 1 hidden state and 3 observation time series
# cross validation here is random, 10 folds
fit <- MARSScv(dat, model = list(
  Z = matrix(1, 3, 1),
  R = "diagonal and equal"
))
```

```
# second, demonstrate passing in pre-specified folds
fold_ids <- matrix(
  sample(1:5, size = nrow(dat) * ncol(dat), replace = TRUE),
  nrow(dat), ncol(dat)
)
fit <- MARSScv(dat, model = list(
  Z = matrix(1, 3, 1),
  R = "diagonal and equal"
), fold_ids = fold_ids)

# third, illustrate future cross validation
fit <- MARSScv(dat, model = list(
  Z = matrix(1, 3, 1),
  R = "diagonal and equal"
), future_cv = TRUE, n_future_cv = 5)
```

MARSSFisherI *Observed Fisher Information Matrix at the MLE*

Description

Returns the observed Fisher Information matrix for a marssMLE object (a fitted MARSS model) via either the analytical algorithm of Harvey (1989) or a numerical estimate.

The observed Fisher Information is the negative of the second-order partial derivatives of the log-likelihood function evaluated at the MLE. The derivatives being with respect to the parameters. The Hessian matrix is the second-order partial derivatives of a scalar-valued function. Thus the observed Fisher Information matrix is the Hessian of the negative log-likelihood function evaluated at the MLE (or equivalently the negative of the Hessian of the log-likelihood function). The inverse of the observed Fisher Information matrix is an estimate of the asymptotic variance-covariance matrix for the estimated parameters. Use MARSShessian() (which calls MARSSFisherI()) to return the parameter variance-covariance matrix computed from the observed Fisher Information matrix.

Note for the numerically estimated Hessian, we pass in the negative log-likelihood function to a minimization function. As a result, the numerical functions return the Hessian of the negative log-likelihood function (which is the observed Fisher Information matrix).

Usage

```
MARSSFisherI(MLEobj, method = c("Harvey1989", "fdHess", "optim"))
```

Arguments

MLEobj	An object of class marssMLE. This object must have a $par element containing MLE parameter estimates from e.g. MARSSkem().

`method` The method to use for computing the observed Fisher Information matrix. Options are `"Harvey1989"` to use the Harvey (1989) recursion, which is an analytical solution, `"fdHess"` or `"optim"` which are two numerical methods. Although 'optim' can be passed to the function, 'fdHess' is used for all numerical estimates used in the MARSS package.

Details

Method 'fdHess' uses `fdHess()` from package `nlme` to numerically estimate the Hessian of the negative log-likelihood function at the MLEs. Method 'optim' uses `optim()` with `hessian=TRUE` and `list(maxit=0)` to ensure that the Hessian is computed at the values in the `par` element of the MLE object. The `par` element of the `marssMLE` object is the MLE.

Method 'Harvey1989' (the default) uses the recursion in Harvey (1989) to compute the observed Fisher Information of a MARSS model analytically. See Holmes (2016c) for a discussion of the Harvey (1989) algorithm and see Holmes (2017) on how to implement the algorithm for MARSS models with linear constraints (the type of MARSS models that the MARSS R package addresses).

There has been research on computing the observed Fisher Information matrix from the derivatives used by EM algorithms (discussed in Holmes (2016a, 2016b)), for example Louis (1982). Unfortunately, the EM algorithm used in the MARSS package is for time series data and the temporal correlation must be dealt with, e.g. Duan & Fulop (2011). Oakes (1999) has an approach that only involves derivatives of $\mathrm{E}[LL(\Theta)|\mathbf{y}, \Theta']$ but one of the derivatives will be the derivative of the $\mathrm{E}[\mathbf{X}|\mathbf{y}, \Theta']$ with respect to Θ'. It is not clear how to do that derivative. Moon-Ho, Shumway and Ombao (2006) suggest (page 157) that this derivative is hard to compute.

Value

Returns the observed Fisher Information matrix.

Author(s)

Eli Holmes, NOAA, Seattle, USA.

References

Harvey, A. C. (1989) Section 3.4.5 (Information matrix) in Forecasting, structural time series models and the Kalman filter. Cambridge University Press, Cambridge, UK.

See also J. E. Cavanaugh and R. H. Shumway (1996) On computing the expected Fisher information matrix for state-space model parameters. Statistics & Probability Letters 26: 347-355. This paper discusses the Harvey (1989) recursion (and proposes an alternative).

Holmes, E. E. 2016a. Notes on computing the Fisher Information matrix for MARSS models. Part I Background. Technical Report. https://doi.org/10.13140/RG.2.2.27306.11204/1 Notes

Holmes, E. E. 2016b. Notes on computing the Fisher Information matrix for MARSS models. Part II Louis 1982. Technical Report. https://doi.org/10.13140/RG.2.2.35694.72000 Notes

Holmes, E. E. 2016c. Notes on computing the Fisher Information matrix for MARSS models. Part III Overview of Harvey 1989. https://eeholmes.github.io/posts/2016-6-16-FI-recursion-3/

Holmes, E. E. 2017. Notes on computing the Fisher Information matrix for MARSS models. Part IV Implementing the Recursion in Harvey 1989. https://eeholmes.github.io/posts/2017-5-31-FI-recursion-4/

Duan, J. C. and A. Fulop. (2011) A stable estimator of the information matrix under EM for dependent data. Statistics and Computing 21: 83-91

Louis, T. A. 1982. Finding the observed information matrix when using the EM algorithm. Journal of the Royal Statistical Society. Series B (Methodological). 44: 226-233.

Oakes, D. 1999. Direct calculation of the information matrix via the EM algorithm. Journal of the Royal Statistical Society. Series B (Methodological). 61: 479-482.

Moon-Ho, R. H., R. H. Shumway, and Ombao 2006. The state-space approach to modeling dynamic processes. Chapter 7 in Models for Intensive Longitudinal Data. Oxford University Press.

See Also

MARSSharveyobsFI(), MARSShessian.numerical, MARSSparamCIs, marssMLE

Examples

```
dat <- t(harborSeal)
dat <- dat[2:4, ]
MLEobj <- MARSS(dat, model=list(Z=matrix(1,3,1), R="diagonal and equal"))
MARSSFisherI(MLEobj)
MARSSFisherI(MLEobj, method="fdHess")
```

MARSSfit *Generic for fitting MARSS models*

Description

This is an internal function used by MARSS(). It is not intended for use by users but needs to be exported so the marssTMB package can use it. Uses the method of a marssMLE class object. Will call a function such as MARSSkem(), MARSSoptim() in the MARSS package or MARSStmb() in the marssTMB package.

Usage

```
MARSSfit(x, ...)
```

Arguments

x	a marssMLE object.
...	additional arguments for the fitting function

MARSSharveyobsFI *Hessian Matrix via the Harvey (1989) Recursion*

Description

Calculates the observed Fisher Information analytically via the recursion by Harvey (1989) as adapted by Holmes (2017) for MARSS models with linear constraints. This is the same as the Hessian of the negative log-likelihood function at the MLEs. This is a utility function in the MARSS-package and is not exported. Use MARSShessian() to access.

Usage

```
MARSSharveyobsFI(MLEobj)
```

Arguments

MLEobj	An object of class marssMLE. This object must have a $par element containing MLE parameter estimates from e.g. MARSSkem.

Value

The observed Fisher Information matrix computed via equation 3.4.69 in Harvey (1989). The differentials in the equation are computed in the recursion in equations 3.4.73a to 3.4.74b. See Holmes (2016c) for a discussion of the Harvey (1989) algorithm and Holmes (2017) for the specific implementation of the algorithm for MARSS models with linear constraints.

Harvey (1989) discusses missing observations in section 3.4.7. However, the MARSSharveyobsFI() function implements the approach of Shumway and Stoffer (2006) in section 6.4 for the missing values. See Holmes (2012) for a full discussion of the missing values modifications.

Author(s)

Eli Holmes, NOAA, Seattle, USA.

References

R. H. Shumway and D. S. Stoffer (2006). Section 6.4 (Missing Data Modifications) in Time series analysis and its applications. Springer-Verlag, New York.

Harvey, A. C. (1989) Section 3.4.5 (Information matrix) in Forecasting, structural time series models and the Kalman filter. Cambridge University Press, Cambridge, UK.

See also J. E. Cavanaugh and R. H. Shumway (1996) On computing the expected Fisher information matrix for state-space model parameters. Statistics & Probability Letters 26: 347-355. This paper discusses the Harvey (1989) recursion (and proposes an alternative).

Holmes, E. E. (2012). Derivation of the EM algorithm for constrained and unconstrained multivariate autoregressive state-space (MARSS) models. Technical Report. arXiv:1302.3919 [stat.ME]

Holmes, E. E. 2016c. Notes on computing the Fisher Information matrix for MARSS models. Part III Overview of Harvey 1989. https://eeholmes.github.io/posts/2016-6-16-FI-recursion-3/

Holmes, E. E. 2017. Notes on computing the Fisher Information matrix for MARSS models. Part IV Implementing the Recursion in Harvey 1989. https://eeholmes.github.io/posts/2017-5-31-FI-recursion-4/

See Also

MARSShessian(), MARSSparamCIs()

Examples

```
dat <- t(harborSeal)
dat <- dat[c(2, 11), ]
fit <- MARSS(dat)
MARSS:::MARSSharveyobsFI(fit)
```

MARSShatyt *Compute Expected Value of Y, YY, and YX*

Description

Computes the expected value of random variables involving $\mathbf{Y}$. Users can use tsSmooth() or print(MLEobj, what="Ey") to access this output. See print.marssMLE.

Usage

```
MARSShatyt(MLEobj, only.kem = TRUE)
```

Arguments

MLEobj	A marssMLE object with the par element of estimated parameters, model element with the model description and data.
only.kem	If TRUE, return only ytT, OtT, yxtT, and yxttpT (values conditioned on the data from $1:T$) needed for the EM algorithm. If only.kem=FALSE, then also return values conditioned on data from 1 to $t-1$ (Ott1 and ytt1) and 1 to t (Ott and ytt), yxtt1T (var$[\mathbf{Y}_t, \mathbf{X}_{t-1}|\mathbf{y}_{1:T}]$), var.ytT (var$[\mathbf{Y}_t|\mathbf{y}_{1:T}]$), and var.EytT (var$_X[E_{Y|x}[\mathbf{Y}_t|\mathbf{y}_{1:T}, \mathbf{x}_t]]$).

Details

For state space models, MARSShatyt() computes the expectations involving $\mathbf{Y}$. If $\mathbf{Y}$ is completely observed, this entails simply replacing $\mathbf{Y}$ with the observed $\mathbf{y}$. When $\mathbf{Y}$ is only partially observed, the expectation involves the conditional expectation of a multivariate normal.

Value

A list with the following components (n is the number of state processes). Following the notation in Holmes (2012), $\mathbf{y}(1)$ is the observed data (for $t = 1 : T$) while $\mathbf{y}(2)$ is the unobserved data. $\mathbf{y}(1, 1 : t-1)$ is the observed data from time 1 to $t-1$.

`ytT`	E[Y(t) | Y(1,1:T)=y(1,1:T)] (n x T matrix).
`ytt1`	E[Y(t) | Y(1,1:t-1)=y(1,1:t-1)] (n x T matrix).
`ytt`	E[Y(t) | Y(1,1:t)=y(1,1:t)] (n x T matrix).
`OtT`	E[Y(t) t(Y(t)) | Y(1,1:T)=y(1,1:T)] (n x n x T array).
`var.ytT`	var[Y(t) | Y(1,1:T)=y(1,1:T)] (n x n x T array).
`var.EytT`	var_X[E_Y[Y(t) | Y(1,1:T)=y(1,1:T), X(t)=x(t)]] (n x n x T array).
`Ott1`	E[Y(t) t(Y(t)) | Y(1,1:t-1)=y(1,1:t-1)] (n x n x T array).
`var.ytt1`	var[Y(t) | Y(1,1:t-1)=y(1,1:t-1)] (n x n x T array).
`var.Eytt1`	var_X[E_Y[Y(t) | Y(1,1:t-1)=y(1,1:t-1), X(t)=x(t)]] (n x n x T array).
`Ott`	E[Y(t) t(Y(t)) | Y(1,1:t)=y(1,1:t)] (n x n x T array).
`yxtT`	E[Y(t) t(X(t)) | Y(1,1:T)=y(1,1:T)] (n x m x T array).
`yxtt1T`	E[Y(t) t(X(t-1)) | Y(1,1:T)=y(1,1:T)] (n x m x T array).
`yxttpT`	E[Y(t) t(X(t+1)) | Y(1,1:T)=y(1,1:T)] (n x m x T array).
`errors`	Any error messages due to ill-conditioned matrices.
`ok`	(TRUE/FALSE) Whether errors were generated.

Author(s)

Eli Holmes, NOAA, Seattle, USA.

References

Holmes, E. E. (2012) Derivation of the EM algorithm for constrained and unconstrained multivariate autoregressive state-space (MARSS) models. Technical report. arXiv:1302.3919 [stat.ME] Type `RShowDoc("EMDerivation",package="MARSS")` to open a copy. See the section on 'Computing the expectations in the update equations' and the subsections on expectations involving Y.

See Also

`MARSS()`, `marssMODEL`, `MARSSkem()`

Examples

```
dat <- t(harborSeal)
dat <- dat[2:3, ]
fit <- MARSS(dat)
EyList <- MARSShatyt(fit)
```

MARSShessian *Parameter Variance-Covariance Matrix from the Hessian Matrix*

Description

Calculates an approximate parameter variance-covariance matrix for the parameters using an inverse of the Hessian of the negative log-likelihood function at the MLEs (the observed Fisher Information matrix). It appends `$Hessian`, `$parMean`, `$parSigma` to the `marssMLE` object.

Usage

```
MARSShessian(MLEobj, method=c("Harvey1989", "fdHess", "optim"))
```

Arguments

`MLEobj`	An object of class `marssMLE`. This object must have a `$par` element containing MLE parameter estimates from e.g. `MARSSkem`.
`method`	The method to use for computing the Hessian. Options are `Harvey1989` to use the Harvey (1989) recursion, which is an analytical solution, `fdHess` or `optim` which are two numerical methods. Although `optim` can be passed to this function, in the internal functions which call this function, `fdHess` will be used if a numerical estimate is requested.

Details

See `MARSSFisherI` for a discussion of the observed Fisher Information matrix and references.

Method `fdHess` uses `fdHess` from package nlme to numerically estimate the Hessian matrix (the matrix of partial 2nd derivatives of the negative log-likelihood function at the MLE). Method `optim` uses `optim` with `hessian=TRUE` and `list(maxit=0)` to ensure that the Hessian is computed at the values in the `par` element of the MLE object. Method `Harvey1989` (the default) uses the recursion in Harvey (1989) to compute the observed Fisher Information of a MARSS model analytically.

Note that the parameter confidence intervals computed with the observed Fisher Information matrix are based on the asymptotic normality of maximum-likelihood estimates under a large-sample approximation.

Value

`MARSShessian()` attaches `Hessian`, `parMean` and `parSigma` to the `marssMLE` object that is passed into the function.

Author(s)

Eli Holmes, NOAA, Seattle, USA.

References

Harvey, A. C. (1989) Section 3.4.5 (Information matrix) in Forecasting, structural time series models and the Kalman filter. Cambridge University Press, Cambridge, UK.

See also J. E. Cavanaugh and R. H. Shumway (1996) On computing the expected Fisher information matrix for state-space model parameters. Statistics & Probability Letters 26: 347-355. This paper discusses the Harvey (1989) recursion (and proposes an alternative).

See Also

MARSSFisherI(), MARSSharveyobsFI(), MARSShessian.numerical(), MARSSparamCIs(), marssMLE

Examples

```
dat <- t(harborSeal)
dat <- dat[c(2, 11), ]
MLEobj <- MARSS(dat)
MLEobj.hessian <- MARSShessian(MLEobj)

# show the approx Hessian
MLEobj.hessian$Hessian

# generate a parameter sample using the Hessian
# this uses the rmvnorm function in the mvtnorm package
hess.params <- mvtnorm::rmvnorm(1,
  mean = MLEobj.hessian$parMean,
  sigma = MLEobj.hessian$parSigma
)
```

MARSShessian.numerical

Hessian Matrix via Numerical Approximation

Description

Calculates the Hessian of the log-likelihood function at the MLEs using either the fdHess function in the nlme package or the optim function. This is a utility function in the MARSS-package and is not exported. Use MARSShessian to access.

Usage

```
MARSShessian.numerical(MLEobj, fun=c("fdHess", "optim"))
```

Arguments

MLEobj	An object of class marssMLE. This object must have a $par element containing MLE parameter estimates from e.g. MARSSkem.
fun	The function to use for computing the Hessian. Options are 'fdHess' or 'optim'.

Details

Method `fdHess` uses `fdHess` from package nlme to numerically estimate the Hessian matrix (the matrix of partial 2nd derivatives) of the negative log-likelihood function with respect to the parameters. Method `optim` uses `optim` with `hessian=TRUE` and `list(maxit=0)` to ensure that the Hessian is computed at the values in the `par` element of the MLE object.

Value

The numerically estimated Hessian of the log-likelihood function at the maximum likelihood estimates.

Author(s)

Eli Holmes, NOAA, Seattle, USA.

See Also

`MARSSharveyobsFI()`, `MARSShessian()`, `MARSSparamCIs()`

Examples

```
dat <- t(harborSeal)
dat <- dat[c(2, 11), ]
MLEobj <- MARSS(dat)
MARSS:::MARSShessian.numerical(MLEobj)
```

`MARSSinfo` *MARSS Error Messages and Warnings*

Description

Prints out more information for MARSS error messages and warnings.

Usage

```
MARSSinfo(number)
```

Arguments

`number`	An error or warning message number.

Value

A print out of information.

Author(s)

Eli Holmes, NOAA, Seattle, USA.

Examples

```
# Show all the info options
MARSSinfo()
```

MARSSinits *Initial Values for MLE*

Description

Sets up generic starting values for parameters for maximum-likelihood estimation algorithms that use an iterative maximization routine needing starting values. Examples of such algorithms are the EM algorithm in `MARSSkem()` and Newton methods in `MARSSoptim()`. This is a utility function in the `MARSS-package`. It is not exported to the user. Users looking for information on specifying initial conditions should look at the help file for `MARSS()` and the User Guide section on initial conditions.

The function assumes that the user passed in the inits list using the parameter names in whatever form was specified in the `MARSS()` call. The default is form="marxss". The `MARSSinits()` function calls MARSSinits_foo, where foo is the form specified in the `MARSS()` call. MARSSinits_foo translates the inits list in form foo into form marss.

Usage

```
MARSSinits(MLEobj, inits=list(B=1, U=0, Q=0.05, Z=1, A=0,
    R=0.05, x0=-99, V0=5, G=0, H=0, L=0))
```

Arguments

`MLEobj`	An object of class `marssMLE`.
`inits`	A list of column vectors (matrices with one column) of the estimated values in each parameter matrix.

Details

Creates an `inits` parameter list for use by iterative maximization algorithms.

Default values for `inits` is supplied in `MARSSsettings.R`. The user can alter these and supply any of the following (m is the dim of X and n is the dim of Y in the MARSS model):

- elem=`A,U` A numeric vector or matrix which will be constructed into `inits$elem` by the command `array(inits$elem),dim=c(n or m,1))`. If elem is fixed in the model, any `inits$elem` values will be overridden and replaced with the fixed value. Default is `array(0,dim=c(n or m,1))`.
- elem=`Q,R,B` A numeric vector or matrix. If length equals the length `MODELobj$fixed$elem` then `inits$elem` will be constructed by `array(inits$elem),dim=dim(MODELobj$fixed$elem))`. If length is 1 or equals dim of `Q` or dim of `R` then `inits$elem` will be constructed into a diagonal matrix by the command `diag(inits$elem)`. If elem is fixed in the model, any `inits$elem` values will be overridden and replaced with the fixed value. Default is `diag(0.05, dim of Q or R)` for `Q` and `R`. Default is `diag(1,m)` for `B`.

- x0 If inits$x0=-99, then starting values for x0 are estimated by a linear regression through the count data assuming A is all zero. This will be a poor start if inits$A is not 0. If inits$x0 is a numeric vector or matrix, inits$x0 will be constructed by the command array(inits$x0),dim=c(m,1)). If x0 is fixed in the model, any inits$x0 values will be overridden and replaced with the fixed value. Default is inits$x0=-99.
- Z If Z is fixed in the model, inits$Z set to the fixed value. If Z is not fixed, then the user must supply inits$Z. There is no default.
- elem=V0 V0 is never estimated, so this is never used.

Value

A list with initial values for the estimated values for each parameter matrix in a MARSS model in marss form. So this will be a list with elements B, U, Q, Z, A, R, x0, V0, G, H, L.

Note

Within the base code, a form-specific internal MARSSinits function is called to allow the output to vary based on form: MARSSinits_dfa, MARSSinits_marss, MARSSinits_marxss.

Author(s)

Eli Holmes, NOAA, Seattle, USA.

See Also

marssMODEL, MARSSkem(), MARSSoptim()

MARSSinnovationsboot *Bootstrapped Data using Stoffer and Wall's Algorithm*

Description

Creates bootstrap data via sampling from the standardized innovations matrix. This is an internal function in the MARSS-package and is not exported. Users should access this with MARSSboot.

Usage

```
MARSSinnovationsboot(MLEobj, nboot = 1000, minIndx = 3)
```

Arguments

MLEobj	An object of class marssMLE. This object must have a $par element containing MLE parameter estimates from e.g. MARSSkem() or MARSS(). This algorithm may not be used if there are missing datapoints in the data.
nboot	Number of bootstraps to perform.
minIndx	Number of innovations to skip. Stoffer & Wall suggest not sampling from innovations 1-3.

Details

Stoffer and Wall (1991) present an algorithm for generating CIs via a non-parametric bootstrap for state-space models. The basic idea is that the Kalman filter can be used to generate estimates of the residuals of the model fit. These residuals are then standardized and resampled and used to generate bootstrapped data using the MARSS model and its maximum-likelihood parameter estimates. One of the limitations of the Stoffer and Wall algorithm is that it cannot be used when there are missing data, unless all data at time t are missing.

Value

A list containing the following components:

`boot.states`	Array (dim is m x tSteps x nboot) of simulated state processes.
`boot.data`	Array (dim is n x tSteps x nboot) of simulated data.
`marss`	`marssMODEL` object element of the `marssMLE` object (`marssMLE$marss`) in "marss" form.
`nboot`	Number of bootstraps performed.

m is the number state processes (x in the MARSS model) and n is the number of observation time series (y in the MARSS model).

Author(s)

Eli Holmes and Eric Ward, NOAA, Seattle, USA.

References

Stoffer, D. S., and K. D. Wall. 1991. Bootstrapping state-space models: Gaussian maximum likelihood estimation and the Kalman filter. Journal of the American Statistical Association 86:1024-1033.

See Also

`stdInnov()`, `MARSSparamCIs()`, `MARSSboot()`

Examples

```
dat <- t(kestrel)
dat <- dat[2:3, ]
fit <- MARSS(dat, model = list(U = "equal", Q = diag(.01, 2)))
boot.obj <- MARSSinnovationsboot(fit)
```

MARSSkem *EM Algorithm function for MARSS models*

Description

MARSSkem() performs maximum-likelihood estimation, using an EM algorithm for constrained and unconstrained MARSS models. Users would not call this function directly normally. The function MARSS() calls MARSSkem(). However users might want to use MARSSkem() directly if they need to avoid some of the error-checking overhead associated with the MARSS() function.

Usage

```
MARSSkem(MLEobj)
```

Arguments

MLEobj	An object of class marssMLE.

Details

Objects of class marssMLE may be built from scratch but are easier to construct using MARSS() with MARSS(..., fit=FALSE).

Options for MARSSkem() may be set using MLEobj$control. The commonly used elements of control are as follows (see marssMLE):

minit Minimum number of EM iterations. You can use this to force the algorithm to do a certain number of iterations. This is helpful if your solution is not converging.

maxit Maximum number of EM iterations.

min.iter.conv.test The minimum number of iterations before the log-log convergence test will be computed. If maxit is set less than this, then convergence will not be computed (and the algorithm will just run for maxit iterations).

kf.x0 Whether to set the prior at $t = 0$ ("x00") or at $t = 1$ ("x10"). The default is "x00".

conv.test.deltaT The number of iterations to use in the log-log convergence test. This defaults to 9.

abstol Tolerance for log-likelihood change for the delta logLik convergence test. If log-likelihood changes less than this amount relative to the previous iteration, the EM algorithm exits. This is normally (default) set to NULL and the log-log convergence test is used instead.

allow.degen Whether to try setting $\mathbf{Q}$ or $\mathbf{R}$ elements to zero if they appear to be going to zero.

trace A positive integer. If not 0, a record will be created of each variable over all EM iterations and detailed warning messages (if appropriate) will be printed.

safe If TRUE, MARSSkem will rerun MARSSkf after each individual parameter update rather than only after all parameters are updated. The latter is slower and unnecessary for many models, but in some cases, the safer and slower algorithm is needed because the ML parameter matrices have high condition numbers.

silent Suppresses printing of progress bars, error messages, warnings and convergence information.

Value

The marssMLE object which was passed in, with additional components:

`method` String "kem".

`kf` Kalman filter output.

`iter.record` If `MLEobj$control$trace = TRUE`, a list with `par` = a record of each estimated parameter over all EM iterations and `logLik` = a record of the log likelihood at each iteration.

`numIter` Number of iterations needed for convergence.

`convergence` Did estimation converge successfully?

convergence=0 Converged in both the abstol test and the log-log plot test.

convergence=1 Some of the parameter estimates did not converge (based on the log-log plot test AND abstol tests) before `MLEobj$control$maxit` was reached. This is not an error per se.

convergence=3 No convergence diagnostics were computed because all parameters were fixed thus no fitting required.

convergence=-1 No convergence diagnostics were computed because the MLE object was not fit (called with fit=FALSE). This isn't a convergence error just information. There is not par element so no functions can be run with the object.

convergence=2 No convergence diagnostics were computed because the MLE object had problems and was not fit. This isn't a convergence error just information.

convergence=10 Abstol convergence only. Some of the parameter estimates did not converge (based on the log-log plot test) before `MLEobj$control$maxit` was reached. However `MLEobj$control$abstol` was reached.

convergence=11 Log-log convergence only. Some of the parameter estimates did not converge (based on the abstol test) before `MLEobj$control$maxit` was reached. However the log-log convergence test was passed.

convergence=12 Abstol convergence only. Log-log convergence test was not computed because `MLEobj$control$maxit` was set to less than `control$min.iter.conv.test`.

convergence=13 Lack of convergence info. Parameter estimates did not converge based on the abstol test before `MLEobj$control$maxit` was reached. No log-log information since `control$min.iter.conv.test` is less than `MLEobj$control$maxit` so no log-log plot test could be done.

convergence=42 `MLEobj$control$abstol` was reached but the log-log plot test returned NAs. This is an odd error and you should set `control$trace=TRUE` and look at the outputted `$iter.record` to see what is wrong.

convergence=52 The EM algorithm was abandoned due to numerical errors. Usually this means one of the variances either went to zero or to all elements being equal. This is not an error per se. Most likely it means that your model is not very good for your data (too inflexible or too many parameters). Try setting `control$trace=1` to view a detailed error report.

convergence=53 The algorithm was abandoned due to numerical errors in the likelihood calculation from MARSSkf.

convergence=62 The algorithm was abandoned due to errors in the log-log convergence test. You should not get this error (it is included for debugging purposes to catch improper arguments passed into the log-log convergence test).

convergence=63 The algorithm was run for `control$maxit` iterations, `control$abstol` not reached, and the log-log convergence test returned errors. You should not get this error (it is included for debugging purposes to catch improper arguments passed into the log-log convergence test).

convergence=72 Other convergence errors. This is included for debugging purposes to catch misc. errors.

`logLik` Log-likelihood.

`states` State estimates from the Kalman smoother.

`states.se` Confidence intervals based on state standard errors, see caption of Fig 6.3 (p. 337) in Shumway & Stoffer (2006).

`errors` Any error messages.

Discussion

To ensure that the global maximum-likelihood values are found, it is recommended that you test the fit under different initial parameter values, particularly if the model is not a good fit to the data. This requires more computation time, but reduces the chance of the algorithm terminating at a local maximum and not reaching the true MLEs. For many models and for draft analyses, this is unnecessary, but answers should be checked using an initial conditions search before reporting final values. See the chapter on initial conditions in the User Guide for a discussion on how to do this.

`MARSSkem()` calls a Kalman filter/smoother `MARSSkf()` for hidden state estimation. The algorithm allows two options for the initial state conditions: fixed but unknown or a prior. In the first case, x0 (whether at t=0 or t=1) is treated as fixed but unknown (estimated); in this case, `fixed$V0=0` and x0 is estimated. This is the default behavior. In the second case, the initial conditions are specified with a prior and V0!=0. In the later case, x0 or V0 may be estimated. MARSS will allow you to try to estimate both, but many researchers have noted that this is not robust so you should fix one or the other.

If you get errors, you can type `MARSSinfo()` for help. Fitting problems often mean that the solution involves an ill-conditioned matrix. For example, your $\mathbf{Q}$ or $\mathbf{R}$ matrix is going to a value in which all elements have the same value, for example zero. If for example, you tried to fit a model with a fixed $\mathbf{R}$ matrix with high values on the diagonal and the variance in that $\mathbf{R}$ matrix (diagonal terms) was much higher than what is actually in the data, then you might drive $\mathbf{Q}$ to zero. Also if you try to fit a structurally inadequate model, then it is not unusual that $\mathbf{Q}$ will be driven to zero. For example, if you fit a model with 1 hidden state trajectory to data that clearly have 2 quite different hidden state trajectories, you might have this problem. Comparing the likelihood of this model to a model with more structural flexibility should reveal that the structurally inflexible model is inadequate (much lower likelihood).

Convergence testing is done via a combination of two tests. The first test (abstol test) is the test that the change in the absolute value of the log-likelihood from one iteration to another is less than some tolerance value (abstol). The second test (log-log test) is that the slope of a plot of the log of the parameter value or log-likelihood versus the log of the iteration number is less than some tolerance. Both of these must be met to generate the Success! parameters converged output. If

you want to circumvent one of these tests, then set the tolerance for the unwanted test to be high. That will guarantee that that test is met before the convergence test you want to use is met. The tolerance for the abstol test is set by `control$abstol` and the tolerance for the log-log test is set by `control$conv.test.slope.tol`. Anything over 1 is huge for both of these.

Author(s)

Eli Holmes and Eric Ward, NOAA, Seattle, USA.

References

R. H. Shumway and D. S. Stoffer (2006). Chapter 6 in Time series analysis and its applications. Springer-Verlag, New York.

Ghahramani, Z. and Hinton, G. E. (1996) Parameter estimation for linear dynamical systems. Technical Report CRG-TR-96-2, University of Toronto, Dept. of Computer Science.

Harvey, A. C. (1989) Chapter 5 in Forecasting, structural time series models and the Kalman filter. Cambridge University Press, Cambridge, UK.

The MARSS User Guide: Holmes, E. E., E. J. Ward, and M. D. Scheuerell (2012) Analysis of multivariate time-series using the MARSS package. NOAA Fisheries, Northwest Fisheries Science Center, 2725 Montlake Blvd E., Seattle, WA 98112 Go to User Guide to open the most recent version.

Holmes, E. E. (2012). Derivation of the EM algorithm for constrained and unconstrained multivariate autoregressive state-space (MARSS) models. Technical Report. arXiv:1302.3919 [stat.ME] EMDerivation has the most recent version.

See Also

`MARSSkf()`, `marssMLE`, `MARSSoptim()`, `MARSSinfo()`

Examples

```
dat <- t(harborSeal)
dat <- dat[2:4, ]
# you can use MARSS to construct a proper marssMLE object.
fit <- MARSS(dat, model = list(Q = "diagonal and equal", U = "equal"), fit = FALSE)
# Pass this marssMLE object to MARSSkem to do the fit.
kemfit <- MARSSkem(fit)
```

MARSSkf *Kalman Filtering and Smoothing*

Description

Provides Kalman filter and smoother output for MARSS models with (or without) time-varying parameters. `MARSSkf()` is a small helper function to select which Kalman filter/smoother function to use based on the value in `MLEobj$fun.kf`. The choices are `MARSSkfas()` which uses the filtering and smoothing algorithms in the KFAS package based on algorithms in Koopman and Durbin (2001-2003), and `MARSSkfss()` which uses the algorithms in Shumway and Stoffer. The default function is `MARSSkfas()` which is faster and generally more stable (fewer matrix inversions), but there are some cases where `MARSSkfss()` might be more stable and it returns a variety of diagnostics that `MARSSkfas()` does not.

Usage

```
MARSSkf(MLEobj, only.logLik = FALSE, return.lag.one = TRUE, return.kfas.model = FALSE,
           newdata = NULL, smoother = TRUE)
MARSSkfss(MLEobj, smoother=TRUE)
MARSSkfas(MLEobj, only.logLik=FALSE, return.lag.one=TRUE, return.kfas.model=FALSE)
```

Arguments

`MLEobj`	A `marssMLE` object with the `par` element of estimated parameters, `marss` element with the model description (in marss form) and data, and `control` element for the fitting algorithm specifications. `control$debugkf` specifies that detailed error reporting will be returned (only used by `MARSSkf()`). `model$diffuse=TRUE` specifies that a diffuse prior be used (only used by `MARSSkfas()`). See KFS documentation. When the diffuse prior is set, `V0` should be non-zero since the diffuse prior variance is `V0*kappa`, where kappa goes to infinity.
`smoother`	Used by `MARSSkfss()`. If set to FALSE, only the Kalman filter is run. The output `xtT`, `VtT`, `x0T`, `Vtt1T`, `V0T`, and `J0` will be NULL.
`only.logLik`	Used by `MARSSkfas()`. If set, only the log-likelihood is returned using the KFAS package function `logLik.SSModel`. This is much faster if only the log-likelihood is needed.
`return.lag.one`	Used by `MARSSkfas()`. If set to FALSE, the smoothed lag-one covariance values are not returned (output `Vtt1T` is set to NULL). This speeds up `MARSSkfas()` because to return the smoothed lag-one covariance a stacked MARSS model is used with twice the number of state vectors—thus the state matrices are larger and take more time to work with.
`return.kfas.model`	Used by `MARSSkfas()`. If set to TRUE, it returns the MARSS model in KFAS model form (class `SSModel`). This is useful if you want to use other KFAS functions or write your own functions to work with `optim()` to do optimization. This can speed things up since there is a bit of code overhead in `MARSSoptim()` associated with the `marssMODEL` model specification needed for the constrained EM algorithm (but not strictly needed for `optim()`; useful but not required.).
`newdata`	A new matrix of data to use in place of the data used to fit the model (in the `model$data` and `marss$data` elements of a `marssMLE` object). If the initial x was estimated (in `x0`) then this estimate will be used for `newdata` and this may not be appropriate.

Details

For state-space models, the Kalman filter and smoother provide optimal (minimum mean square error) estimates of the hidden states. The Kalman filter is a forward recursive algorithm which computes estimates of the states $\mathbf{x}_t$ conditioned on the data up to time t (`xtt`). The Kalman smoother is a backward recursive algorithm which starts at time T and works backwards to $t = 1$ to provide estimates of the states conditioned on all data (`xtT`). The data may contain missing values (NAs). All parameters may be time varying.

The initial state is either an estimated parameter or treated as a prior (with mean and variance). The initial state can be specified at $t = 0$ or $t = 1$. The EM algorithm in the MARSS package (`MARSSkem()`) provides both Shumway and Stoffer's derivation that uses $t = 0$ and Ghahramani et al algorithm which uses $t = 1$. The `MLEobj$model$tinitx` argument specifies whether the initial states (specified with `x0` and `V0` in the `model` list) is at $t = 0$ (`tinitx=0`) or $t = 1$ (`tinitx=1`). If `MLEobj$model$tinitx=0`, `x0` is defined as $\mathrm{E}[\mathbf{X}_0|\mathbf{y}_0]$ and `V0` is defined as $\mathrm{E}[\mathbf{X}_0\mathbf{X}_0|\mathbf{y}_0]$ which appear in the Kalman filter at $t = 1$ (first set of equations). If `MLEobj$model$tinitx=1`, `x0` is defined as $\mathrm{E}[\mathbf{X}_1|\mathbf{y}_0]$ and `V0` is defined as $\mathrm{E}[\mathbf{X}_1\mathbf{X}_1|\mathbf{y}_0]$ which appear in the Kalman filter at $t = 1$ (and the filter starts at t=1 at the 3rd and 4th equations in the Kalman filter recursion). Thus if `MLEobj$model$tinitx=1`, `x0=xtt1[,1]` and `V0=Vtt1[,,1]` in the Kalman filter output while if `MLEobj$model$tinitx=0`, the initial condition will not be in the filter output since time starts at 1 not 0 in the output.

`MARSSkfss()` is a native R implementation based on the Kalman filter and smoother equation as shown in Shumway and Stoffer (sec 6.2, 2006). The equations have been altered to allow the initial state distribution to be to be specified at $t = 0$ or $t = 1$ (data starts at $t = 1$) per per Ghahramani and Hinton (1996). In addition, the filter and smoother equations have been altered to allow partially deterministic models (some or all elements of the $\mathbf{Q}$ diagonals equal to 0), partially perfect observation models (some or all elements of the $\mathbf{R}$ diagonal equal to 0) and fixed (albeit unknown) initial states (some or all elements of the $\mathbf{V0}$ diagonal equal to 0) (per Holmes 2012). The code includes numerous checks to alert the user if matrices are becoming ill-conditioned and the algorithm unstable.

`MARSSkfas()` uses the (Fortran-based) Kalman filter and smoother function (`KFS()`) in the KFAS package (Helske 2012) based on the algorithms of Koopman and Durbin (2000, 2001, 2003). The Koopman and Durbin algorithm is faster and more stable since it avoids matrix inverses. Exact diffuse priors are also allowed in the KFAS Kalman filter function. The standard output from the KFAS functions do not include the lag-one covariance smoother needed for the EM algorithm. `MARSSkfas` computes the smoothed lag-one covariance using the Kalman filter applied to a stacked MARSS model as described on page 321 in Shumway and Stoffer (2000). Also the KFAS model specification only has the initial state at $t = 1$ (as $\mathbf{X}_1$ conditioned on $\mathbf{y}_0$, which is missing). When the initial state is specified at $t = 0$ (as $\mathbf{X}_0$ conditioned on $\mathbf{y}_0$), `MARSSkfas()` computes the required $\mathrm{E}[\mathbf{X}_1|\mathbf{y}_0$ and $\mathrm{var}[\mathbf{X}_1|\mathbf{y}_0$ using the Kalman filter equations per Ghahramani and Hinton (1996).

The likelihood returned for both functions is the exact likelihood when there are missing values rather than the approximate likelihood sometimes presented in texts for the missing values case. The functions return the same filter, smoother and log-likelihood values. The differences are that `MARSSkfas()` is faster (and more stable) but `MARSSkfss()` has many internal checks and error messages which can help debug numerical problems (but slow things down). Also `MARSSkfss()` returns some output specific to the traditional filter algorithm (`J` and `Kt`).

Value

A list with the following components. m is the number of state processes and n is the number of observation time series. "V" elements are called "P" in Shumway and Stoffer (2006, eqn 6.17 with s=T). The output is referenced against equations in Shumway and Stoffer (2006) denoted S&S; the Kalman filter and smoother implemented in MARSS is for a more general MARSS model than that shown in S&S but the output has the same meaning. In the expectations below, the parameters are left off; $\mathrm{E}[\mathbf{X}|\mathbf{y}_1^t]$ is really $\mathrm{E}[\mathbf{X}|\Theta, \mathbf{Y}_1^t = \mathbf{y}_1^t]$ where Θ is the parameter list. $\mathbf{y}_1^t$ denotes the data from $t = 1$ to $t = t$.

The notation for the conditional expectations is $\mathbf{x}_t^t = \mathrm{E}[\mathbf{X}|\mathbf{y}_1^t]$, $\mathbf{x}_t^{t-1} = \mathrm{E}[\mathbf{X}|\mathbf{y}_1^{t-1}]$ and $\mathbf{x}_t^T = \mathrm{E}[\mathbf{X}|\mathbf{y}_1^T]$. The conditional variances and covariances use similar notation. Note that in the Holmes (2012), the EM Derivation, $\mathbf{x}_t^T$ and $\mathbf{V}_t^T$ are given special symbols because they appear repeatedly: $\tilde{\mathbf{x}}_t$ and $\tilde{\mathbf{V}}_t$ but here the more general notation is used.

`xtT` $\mathbf{x}_t^T$ State first moment conditioned on $\mathbf{y}_1^T$: $\mathrm{E}[\mathbf{X}_t|\mathbf{y}_1^T]$ (m x T matrix). Kalman smoother output.

`VtT` $\mathbf{V}_t^T$ State variance matrix conditioned on $\mathbf{y}_1^T$: $\mathrm{E}[(\mathbf{X}_t - \mathbf{x}_t^T)(\mathbf{X}_t - \mathbf{x}_t^T)^\top|\mathbf{y}_1^T]$ (m x m x T array). Kalman smoother output. Denoted P_t^T in S&S (S&S eqn 6.18 with $s = T$, $t1 = t2 = t$). The state second moment $\mathrm{E}[\mathbf{X}_t\mathbf{X}_t^\top|\mathbf{y}_1^T]$ is equal to $\mathbf{V}_t^T + \mathbf{x}_t^T(\mathbf{x}_t^T)^\top$.

`Vtt1T` $\mathbf{V}_{t,t-1}^T$ State lag-one cross-covariance matrix $\mathrm{E}[(\mathbf{X}_t - \mathbf{x}_t^T)(\mathbf{X}_{t-1} - \mathbf{x}_{t-1}^T)^\top|\mathbf{y}_1^T]$ (m x m x T). Kalman smoother output. $P_{t,t-1}^T$ in S&S (S&S eqn 6.18 with $s = T$, $t1 = t$, $t2 = t - 1$). State lag-one second moments $\mathrm{E}[\mathbf{X}_t\mathbf{X}_{t-1}^\top|\mathbf{y}_1^T]$ is equal to $\mathbf{V}_{t,t-1}^T + \mathbf{x}_t^T(\mathbf{x}_{t-1}^T)^\top$.

`x0T` Initial smoothed state estimate $\mathrm{E}[\mathbf{X}_{t0}|\mathbf{y}_1^T$ (m x 1). If `model$tinitx=0`, $t0 = 0$; if `model$tinitx=1`, $t0 = 1$. Kalman smoother output.

`x01T` Smoothed state estimate $\mathrm{E}[\mathbf{X}_1|\mathbf{y}_1^T$ (m x 1).

`x00T` Smoothed state estimate $\mathrm{E}[\mathbf{X}_0|\mathbf{y}_1^T$ (m x 1). If `model$tinitx=1`, this will be NA.

`V0T` Initial smoothed state covariance matrix $\mathrm{E}[\mathbf{X}_{t0}\mathbf{X}_0^\top|\mathbf{y}_1^T$ (m x m). If `model$tinitx=0`, $t0 = 0$ and `V0T=V00T`; if `model$tinitx=1`, $t0 = 1$ and `V0T=V10T`. In the case of `tinitx=0`, this is P_0^T in S&S.

`V10T` Smoothed state covariance matrix $\mathrm{E}[\mathbf{X}_1\mathbf{X}_0^\top|\mathbf{y}_1^T$ (m x m).

`V00T` Smoothed state covariance matrix $\mathrm{E}[\mathbf{X}_0\mathbf{X}_0^\top|\mathbf{y}_1^T$ (m x m). If `model$tinitx=1`, this will be NA.

`J` (m x m x T) Kalman smoother output. Only for `MARSSkfss()`. (ref S&S eqn 6.49)

`J0` J at the initial time (t=0 or t=1) (m x m x T). Kalman smoother output. Only for `MARSSkfss()`.

`xtt` State first moment conditioned on $\mathbf{y}_1^t$: $\mathrm{E}[\mathbf{X}_t|\mathbf{y}_1^t$ (m x T). Kalman filter output. (S&S eqn 6.17 with $s = t$)

`xtt1` State first moment conditioned on $\mathbf{y}_1^{t-1}$: $\mathrm{E}[\mathbf{X}_t|\mathbf{y}_1^{t-1}$ (m x T). Kalman filter output. (S&S eqn 6.17 with $s = t - 1$)

`Vtt` State variance conditioned on $\mathbf{y}_1^t$: $\mathrm{E}[(\mathbf{X}_t - \mathbf{x}_t^t)(\mathbf{X}_t - \mathbf{x}_t^t)^\top|\mathbf{y}_1^t]$ (m x m x T array). Kalman filter output. P_t^t in S&S (S&S eqn 6.18 with s=t, t1=t2=t). The state second moment $\mathrm{E}[\mathbf{X}_t\mathbf{X}_t^\top|\mathbf{y}_1^t]$ is equal to $\mathbf{V}_t^t + \mathbf{x}_t^t(\mathbf{x}_t^t)^\top$.

`Vtt1`	State variance conditioned on $\mathbf{y}_1^{t-1}$: $\mathrm{E}[(\mathbf{X}_t - \mathbf{x}_t^{t-1})(\mathbf{X}_t - \mathbf{x}_t^{t-1})^\top \vert \mathbf{y}_1^{t-1}]$ (m x m x T array). Kalman filter output. The state second moment $\mathrm{E}[\mathbf{X}_t \mathbf{X}_t^\top \vert \mathbf{y}_1^{t-1}]$ is equal to $\mathbf{V}_t^{t-1} + \mathbf{x}_t^{t-1}(\mathbf{x}_t^{t-1})^\top$.
`Kt`	Kalman gain (m x m x T). Kalman filter output (ref S&S eqn 6.23). Only for `MARSSkfss()`.
`Innov`	Innovations $\mathbf{y}_t - \mathrm{E}[\mathbf{Y}_t \vert \mathbf{y}_1^{t-1}]$ (n x T). Kalman filter output. Only returned with `MARSSkfss()`. (ref page S&S 339).
`Sigma`	Innovations covariance matrix. Kalman filter output. Only returned with `MARSSkfss()`. (ref S&S eqn 6.61)
`logLik`	Log-likelihood logL(y(1:T) | Theta) (ref S&S eqn 6.62)
`kfas.model`	The model in KFAS model form (class `SSModel`). Only for `MARSSkfas`.
`errors`	Any error messages.

Author(s)

Eli Holmes, NOAA, Seattle, USA.

References

A. C. Harvey (1989). Chapter 5, Forecasting, structural time series models and the Kalman filter. Cambridge University Press.

R. H. Shumway and D. S. Stoffer (2006). Time series analysis and its applications: with R examples. Second Edition. Springer-Verlag, New York.

Ghahramani, Z. and Hinton, G.E. (1996) Parameter estimation for linear dynamical systems. University of Toronto Technical Report CRG-TR-96-2.

Holmes, E. E. (2012). Derivation of the EM algorithm for constrained and unconstrained multivariate autoregressive state-space (MARSS) models. Technical Report. arXiv:1302.3919 [stat.ME] `RShowDoc("EMDerivation",package="MARSS")` to open a copy.

Jouni Helske (2012). KFAS: Kalman filter and smoother for exponential family state space models. https://CRAN.R-project.org/package=KFAS

Koopman, S.J. and Durbin J. (2000). Fast filtering and smoothing for non-stationary time series models, Journal of American Statistical Association, 92, 1630-38.

Koopman, S.J. and Durbin J. (2001). Time series analysis by state space methods. Oxford: Oxford University Press.

Koopman, S.J. and Durbin J. (2003). Filtering and smoothing of state vector for diffuse state space models, Journal of Time Series Analysis, Vol. 24, No. 1.

The MARSS User Guide: Holmes, E. E., E. J. Ward, and M. D. Scheuerell (2012) Analysis of multivariate time-series using the MARSS package. NOAA Fisheries, Northwest Fisheries Science Center, 2725 Montlake Blvd E., Seattle, WA 98112 Type `RShowDoc("UserGuide",package="MARSS")` to open a copy.

See Also

`MARSS()`, `marssMODEL`, `MARSSkem()`, `KFAS()`

Examples

```
dat <- t(harborSeal)
dat <- dat[2:nrow(dat), ]
# you can use MARSS to construct a marssMLE object
# MARSS calls MARSSinits to construct default initial values
# with fit = FALSE, the $par element of the marssMLE object will be NULL
fit <- MARSS(dat, fit = FALSE)
# MARSSkf needs a marssMLE object with the par element set
fit$par <- fit$start
# Compute the kf output at the params used for the inits
kfList <- MARSSkf(fit)
```

---

marssMLE-class *Class "marssMLE"*

Description

marssMLE objects specify fitted multivariate autoregressive state-space models (maximum-likelihood) in the package MARSS-package.

A marssMLE object in the MARSS-package that has all the elements needed for maximum-likelihood estimation of multivariate autoregressive state-space model: the data, model, estimation methods, and any control options needed for the method. If the model has been fit and parameters estimated, the object will also have the MLE parameters. Other functions add other elements to the marssMLE object, such as CIs, s.e.'s, AICs, and the observed Fisher Information matrix. There are `print`, `summary`, `coef`, `fitted`, `residuals`, `predict` and `simulate` methods for marssMLE objects and a bootstrap function. Rather than working directly with the elements of a marssMLE object, use `print()`, `tidy()`, `fitted()`, `tsSmooth()`, `predict()`, or `residuals()` to extract output.

Methods

print `signature(x = "marssMLE")`: ...

summary `signature(object = "marssMLE")`: ...

coef `signature(object = "marssMLE")`: ...

residuals `signature(object = "marssMLE")`: ...

predict `signature(object = "marssMLE")`: ...

fitted `signature(object = "marssMLE")`: ...

logLik `signature(object = "marssMLE")`: ...

simulate `signature(object = "marssMLE")`: ...

forecast `signature(object = "marssMLE")`: ...

accuracy `signature(object = "marssMLE")`: ...

toLatex `signature(object = "marssMLE")`: ...

Author(s)

Eli Holmes and Kellie Wills, NOAA, Seattle, USA

See Also

is.marssMLE(), print.marssMLE(), summary.marssMLE(), coef.marssMLE(), residuals.marssMLE(), fitted.marssMLE(), tsSmooth.marssMLE(), logLik.marssMLE(), simulate.marssMLE(), predict.marssMLE(), forecast.marssMLE(), accuracy.marssMLE(), toLatex.marssMLE()

---

marssMODEL-class *Class "marssMODEL"*

Description

marssMODEL objects describe a vectorized form for the multivariate autoregressive state-space models used in the package MARSS-package.

Details

The object has the following attributes:

form The form that the model object is in.

par.names The names of each parameter matrix in the model.

model.dims A list with the dimensions of all the matrices in non-vectorized form.

X.names Names for the X rows.

Y.names Names for the Y rows.

equation The model equation. Used to write the model in LaTeX.

These attributes are set in the MARSS_form.R file, in the MARSS.form() function and must be internally consistent with the elements of the model. These attributes are used for internal error checking.

Each parameter matrix in a MARSS equation can be written in vectorized form: vec(P) = f + Dp, where f is the fixed part, p are the estimated parameters, and D is the matrix that transforms the p into a vector to be added to f.

An object of class marssMODEL is a list with elements:

data Data supplied by user.

fixed A list with the f row vectors for each parameter matrix.

free A list with the D matrices for each parameter matrix.

tinitx At what t, 0 or 1, is the initial x defined at?

diffuse Whether a diffuse initial prior is used. TRUE or FALSE. Not used unless method="BFGS" was used.

For the marss form, the matrices are called: Z, A, R, B, U, Q, x0, V0. This is the form used by all internal algorithms, thus alternate forms must be transformed to marss form before fitting. For the marxss form (the default form in a MARSS() call), the matrices are called: Z, A, R, B, U, Q, D, C, d, c, x0, V0.

Each form, should have a file called MARSS_form.R, with the following functions. Let foo be some form.

MARSS.foo(MARSS.call) This is called in MARSS() and takes the input from the MARSS() call (a list called MARSS.call) and returns that list with two model objects added. First is a model object in marss form in the $marss element and a model object in the form foo.

marss_to_foo(marssMLE or marssMODEL) If called with marssMODEL (in form marss), marss_to_foo returns a model in form foo. If marss_to_foo is called with a marssMLE object (which must have a $marss element by definition), it returns a $model element in form foo and all if the marssMLE object has par, par.se, par.CI, par.bias, start elements, these are also converted to foo form. The function is mainly used by print.foo which needs the par (and related) elements of a marssMLE object to be in foo form for printing.

foo_to_marss(marssMODEL or marssMLE) This converts marssMODEL(form=foo) to marssMODEL(form=marss). This transformation is always possible since MARSS only works for models for which this is possible. If called with marssMODEL, it returns only a marssMODEL object. If called with a marssMLE object, it adds the $marss element with a marssMODEL in "marss" form and if the par (or related) elements exists, these are converted to "marss" form.

print_foo(marssMLE or marssMODEL) print.marssMLE prints the par (and par.se and start) element of a marssMLE object but does not make assumptions about its form. Normally this element is in form=marss. print.marssMLE checks for a print_foo function and runs that on the marssMLE object first. This allows one to call foo_to_marss() to covert the par (and related) element to foo form so they look familiar to the user (the marss form will look strange). If called with marssMLE, print_foo returns a marssMLE object with the par (and related) elements in foo form. If called with a marssMODEL, print_foo returns a marssMODEL in foo form.

coef_foo(marssMLE) See print_foo. coef.marssMLE also uses the par (and related) elements.

predict_foo(marssMLE) Called by predict.marssMLE to do any needed conversions. Typically a form will want the newdata element in a particular format and this will need to be converted to marss form. This returns an updated marssMLE object and newdata.

describe_foo(marssMODEL) Called by describe.marssMODEL to do allow custom description output.

is.marssMODEL_foo(marssMODEL) Check that the model object in foo form has all the parts it needs and that these have the proper size and form.

MARSSinits_foo(marssMLE, inits.list) Allows customization of the inits used by the form. Returns an inits list in marss form.

Methods

print `signature(x = "marssMODEL")`: ...

summary `signature(object = "marssMODEL")`: ...

toLatex `signature(object = "marssMODEL")`: ...

model.frame `signature(object = "marssMODEL")`: ...

Author(s)

Eli Holmes, NOAA, Seattle, USA.

MARSSoptim *Parameter estimation for MARSS models using optim*

Description

Parameter estimation for MARSS models using R's optim() function. This allows access to R's quasi-Newton algorithms available in that function. The MARSSoptim() function is called when MARSS() is called with method="BFGS". This is an internal function in the MARSS-package.

Usage

```
MARSSoptim(MLEobj)
```

Arguments

MLEobj	An object of class marssMLE.

Details

Objects of class marssMLE may be built from scratch but are easier to construct using MARSS() called with MARSS(..., fit=FALSE, method="BFGS").

Options for optim() are passed in using MLEobj$control. See optim() for a list of that function's control options. If lower and upper for optim() need to be passed in, they should be passed in as part of control as control$lower and control$upper. Additional control arguments affect printing and initial conditions.

MLEobj$control$kf.x0 The initial condition is at $t=0$ if kf.x0="x00". The initial condition is at $t=1$ if kf.x0="x10".

MLEobj$marss$diffuse If diffuse=TRUE, a diffuse initial condition is used. MLEobjparV0 is then the scaling function for the diffuse part of the prior. Thus the prior is V0*kappa where kappa–>Inf. Note that setting a diffuse prior does not change the correlation structure within the prior. If diffuse=FALSE, a non-diffuse prior is used and MLEobjparV0 is the non-diffuse prior variance on the initial states. The the prior is V0.

MLEobj$control$silent Suppresses printing of progress bars, error messages, warnings and convergence information.

Value

The marssMLE object which was passed in, with additional components:

method	String "BFGS".
kf	Kalman filter output.
iter.record	If MLEobj$control$trace = TRUE, then this is the $message value from optim.
numIter	Number of iterations needed for convergence.
convergence	Did estimation converge successfully?

convergence=0 Converged in less than `MLEobj$control$maxit` iterations and no evidence of degenerate solution.

convergence=3 No convergence diagnostics were computed because all parameters were fixed thus no fitting required.

convergence=-1 No convergence diagnostics were computed because the MLE object was not fit (called with fit=FALSE). This isn't a convergence error just information. There is not par element so no functions can be run with the object.

convergence=1 Maximum number of iterations `MLEobj$control$maxit` was reached before `MLEobj$control$abstol` condition was satisfied.

convergence=10 Some of the variance elements appear to be degenerate.

convergence=52 The algorithm was abandoned due to errors from the "L-BFGS-B" method.

convergence=53 The algorithm was abandoned due to numerical errors in the likelihood calculation from `MARSSkf`. If this happens with "BFGS", it can sometimes be helped with a better initial condition. Try using the EM algorithm first (`method="kem"`), and then using the parameter estimates from that to as initial conditions for `method="BFGS"`.

convergence=54 The algorithm successfully fit the model but the Kalman filter/smoother could not be run on the model. Consult `MARSSinfo('optimerror54')` for insight.

`logLik` Log-likelihood.

`states` State estimates from the Kalman smoother.

`states.se` Confidence intervals based on state standard errors, see caption of Fig 6.3 (p. 337) in Shumway & Stoffer (2006).

`errors` Any error messages.

Discussion

The function only returns parameter estimates. To compute CIs, use `MARSSparamCIs` but if you use parametric or non-parametric bootstrapping with this function, it will use the EM algorithm to compute the bootstrap parameter estimates! The quasi-Newton estimates are too fragile for the bootstrap routine since one often needs to search to find a set of initial conditions that work (i.e. don't lead to numerical errors).

Estimates from `MARSSoptim` (which come from `optim`) should be checked against estimates from the EM algorithm. If the quasi-Newton algorithm works, it will tend to find parameters with higher likelihood faster than the EM algorithm. However, the MARSS likelihood surface can be multimodal with sharp peaks at degenerate solutions where a $\mathbf{Q}$ or $\mathbf{R}$ diagonal element equals 0. The quasi-Newton algorithm sometimes gets stuck on these peaks even when they are not the maximum. Neither an initial conditions search nor starting near the known maximum (or from the parameters estimates after the EM algorithm) will necessarily solve this problem. Thus it is wise to check against EM estimates to ensure that the BFGS estimates are close to the MLE estimates (and vis-a-versa, it's wise to rerun with method="BFGS" after using method="kem"). Conversely, if there is a strong flat ridge in your likelihood, the EM algorithm can report early convergence while the BFGS may continue much further along the ridge and find very different parameter values. Of course a likelihood surface with strong flat ridges makes the MLEs less informative...

Note this is mainly a problem if the time series are short or very gappy. If the time series are long, then the likelihood surface should be nice with a single interior peak. In this case, the quasi-Newton algorithm works well but it can still be sensitive (and slow) if not started with a good initial condition. Thus starting it with the estimates from the EM algorithm is often desirable.

One should be aware that the prior set on the variance of the initial states at t=0 or t=1 can have catastrophic effects on one's estimates if the presumed prior covariance structure conflicts with the structure implied by the MARSS model. For example, if you use a diagonal variance-covariance matrix for the prior but the model implies a variance-covariance matrix with non-zero covariances, your MLE estimates can be strongly influenced by the prior variance-covariance matrix. Setting a diffuse prior does not help because the diffuse prior still has the correlation structure specified by V0. One way to detect priors effects is to compare the BFGS estimates to the EM estimates. Persistent differences typically signify a problem with the correlation structure in the prior conflicting with the implied correlation structure in the MARSS model.

Author(s)

Eli Holmes, NOAA, Seattle, USA.

See Also

MARSS(), MARSSkem(), marssMLE(), optim()

Examples

```
dat <- t(harborSealWA)
dat <- dat[2:4, ] # remove the year row

# fit a model with EM and then use that fit as the start for BFGS
# fit a model with 1 hidden state where obs errors are iid
# R="diagonal and equal" is the default so not specified
# Q is fixed
kemfit <- MARSS(dat, model = list(Z = matrix(1, 3, 1), Q = matrix(.01)))
bfgsfit <- MARSS(dat,
  model = list(Z = matrix(1, 3, 1), Q = matrix(.01)),
  inits = coef(kemfit, form = "marss"), method = "BFGS"
)
```

MARSSparamCIs *Standard Errors, Confidence Intervals and Bias for MARSS Parameters*

Description

Computes standard errors, confidence intervals and bias for the maximum-likelihood estimates of MARSS model parameters. If you want confidence intervals on the estimated hidden states, see print.marssMLE() and look for states.cis.

Usage

```
MARSSparamCIs(MLEobj, method = "hessian", alpha = 0.05, nboot =
    1000, silent = TRUE, hessian.fun = "Harvey1989")
```

Arguments

MLEobj	An object of class marssMLE. Must have a $par element containing the MLE parameter estimates.
method	Method for calculating the standard errors: "hessian", "parametric", and "innovations" implemented currently.
alpha	alpha level for the 1-alpha confidence intervals.
nboot	Number of bootstraps to use for "parametric" and "innovations" methods.
hessian.fun	The function to use for computing the Hessian. Options are "Harvey1989" (default analytical) or two numerical options: "fdHess" and "optim". See MARSShessian.
silent	If false, a progress bar is shown for "parametric" and "innovations" methods.

Details

Approximate confidence intervals (CIs) on the model parameters may be calculated from the observed Fisher Information matrix ("Hessian CIs", see MARSSFisherI()) or parametric or non-parametric (innovations) bootstrapping using nboot bootstraps. The Hessian CIs are based on the asymptotic normality of MLE parameters under a large-sample approximation. The Hessian computation for variance-covariance matrices is a symmetric approximation and the lower CIs for variances might be negative. Bootstrap estimates of parameter bias are reported if method "parametric" or "innovations" is specified.

Note, these are added to the par elements of a marssMLE object but are in "marss" form not "marxss" form. Thus the MLEobj$par.upCI and related elements that are added to the marssMLE object may not look familiar to the user. Instead the user should extract these elements using print(MLEobj) and passing in the argument what set to "par.se","par.bias","par.lowCIs", or "par.upCIs". See print(). Or use tidy().

Value

MARSSparamCIs returns the marssMLE object passed in, with additional components par.se, par.upCI, par.lowCI, par.CI.alpha, par.CI.method, par.CI.nboot and par.bias (if method is "parametric" or "innovations").

Author(s)

Eli Holmes, NOAA, Seattle, USA.

References

Holmes, E. E., E. J. Ward, and M. D. Scheuerell (2012) Analysis of multivariate time-series using the MARSS package. NOAA Fisheries, Northwest Fisheries Science Center, 2725 Montlake Blvd E., Seattle, WA 98112 Type RShowDoc("UserGuide",package="MARSS") to open a copy.

See Also

MARSSboot(), MARSSinnovationsboot(), MARSShessian()

Examples

```
dat <- t(harborSealWA)
dat <- dat[2:4, ]
kem <- MARSS(dat, model = list(
  Z = matrix(1, 3, 1),
  R = "diagonal and unequal"
))
kem.with.CIs.from.hessian <- MARSSparamCIs(kem)
kem.with.CIs.from.hessian
```

---

marssPredict-class *Class "marssPredict"*

Description

marssPredict objects are returned by predict.marssMLE and forecast.marssMLE.

A marssPredict object in the MARSS-package has the output with intervals, the original model and values needed for plotting. The object is mainly used for plot.marssPredict() and print.marssPredict().

Methods

print signature(x = "marssPredict"): ...

plot signature(object = "marssPredict"): ...

Author(s)

Eli Holmes, NOAA, Seattle, WA.

See Also

plot.marssPredict(), predict.marssMLE(), forecast.marssMLE()

MARSSresiduals *MARSS Residuals*

Description

The normal residuals function is residuals(). MARSSresiduals() returns residuals as a list of matrices while residuals() returns the same information in a data frame. This function calculates the residuals, residuals variance, and standardized residuals for the one-step-ahead (conditioned on data up to $t-1$), the smoothed (conditioned on all the data), and contemporaneous (conditioned on data up to t) residuals.

Usage

```
MARSSresiduals(object, ..., type = c("tT", "tt1", "tt"),
    normalize = FALSE, silent = FALSE,
    fun.kf = c("MARSSkfas", "MARSSkfss"))
```

Arguments

object	An object of class marssMLE.
...	Additional arguments to be passed to the residuals functions. For type="tT", Harvey=TRUE can be passed into to use the Harvey et al (1998) algorithm.
type	"tT" for smoothed residuals conditioned on all the data $t = 1$ to T, aka smoothation residuals. "tt1" for one-step-ahead residuals, aka innovations residuals. "tt" for contemporaneous residuals.
normalize	TRUE/FALSE See details.
silent	If TRUE, do not print inversion warnings.
fun.kf	Kalman filter function to use. Can be ignored.

Details

For smoothed residuals, see MARSSresiduals.tT().

For one-step-ahead residuals, see MARSSresiduals.tt1().

For contemporaneous residuals, see MARSSresiduals.tt().

Standardized residuals

Standardized residuals have been adjusted by the variance of the residuals at time t such that the variance of the residuals at time t equals 1. Given the normality assumption, this means that one typically sees +/- 2 confidence interval lines on standardized residuals plots.

std.residuals are Cholesky standardized residuals. These are the residuals multiplied by the inverse of the lower triangle of the Cholesky decomposition of the variance matrix of the residuals:

$$\hat{\Sigma}_t^{-1/2}\hat{\mathbf{v}}_t.$$

These residuals are uncorrelated with each other, although they are not necessarily temporally uncorrelated (innovations residuals are temporally uncorrelated).

The interpretation of the Cholesky standardized residuals is not straight-forward when the $\mathbf{Q}$ and $\mathbf{R}$ variance-covariance matrices are non-diagonal. The residuals which were generated by a non-diagonal variance-covariance matrices are transformed into orthogonal residuals in $\text{MVN}(0, \mathbf{I})$ space. For example, if v is 2x2 correlated errors with variance-covariance matrix $\mathbf{R}$. The transformed residuals (from this function) for the i-th row of v is a combination of the row 1 effect and the row 1 effect plus the row 2 effect. So in this case, row 2 of the transformed residuals would not be regarded as solely the row 2 residual but rather how different row 2 is from row 1, relative to expected. If the errors are highly correlated, then the Cholesky standardized residuals can look rather non-intuitive.

`mar.residuals` are the marginal standardized residuals. These are the residuals multiplied by the inverse of the diagonal matrix formed from the square-root of the diagonal of the variance matrix of the residuals:

$$\text{dg}(\hat{\Sigma}_t)^{-1/2}\hat{\mathbf{v}}_t,$$

where $dg(A)$ is the square matrix formed from the diagonal of A, aka `diag(diag(A))`. These residuals will be correlated if the variance matrix is non-diagonal.

The Block Cholesky standardized residuals are like the Cholesky standardized residuals except that the full variance-covariance matrix is not used, only the variance-covariance matrix for the model or state residuals (respectively) is used for standardization. For the model residuals, the Block Cholesky standardized residuals will be the same as the Cholesky standardized residuals because the upper triangle of the lower triangle of the Cholesky decomposition (which is what we standardize by) is all zero. For `type="tt1"` and `type="tt"`, the Block Cholesky standardized state residuals will be the same as the Cholesky standardized state residuals because in the former, the model and state residuals are uncorrelated and in the latter, the state residuals do not exist. For `type="tT"`, the model and state residuals are correlated and the Block Cholesky standardized residuals will be different than the Cholesky standardized residuals.

Normalized residuals

If `normalize=FALSE`, the unconditional variance of $\mathbf{V}_t$ and $\mathbf{W}_t$ are $\mathbf{R}$ and $\mathbf{Q}$ and the model is assumed to be written as

$$\mathbf{y}_t = \mathbf{Z}\mathbf{x}_t + \mathbf{a} + \mathbf{v}_t$$

$$\mathbf{x}_t = \mathbf{B}\mathbf{x}_{t-1} + \mathbf{u} + \mathbf{w}_t$$

If `normalize=TRUE`, the model is assumed to be written as

$$\mathbf{y}_t = \mathbf{Z}\mathbf{x}_t + \mathbf{a} + \mathbf{H}\mathbf{v}_t$$

$$\mathbf{x}_t = \mathbf{B}\mathbf{x}_{t-1} + \mathbf{u} + \mathbf{G}\mathbf{w}_t$$

with the variance of $\mathbf{V}_t$ and $\mathbf{W}_t$ equal to $\mathbf{I}$ (identity).

Missing or left-out data

See the discussion of residuals for missing and left-out data in `MARSSresiduals.tT()`.

Value

A list of the following components

`model.residuals`
: The model residuals (data minus model predicted values) as a n x T matrix.

`state.residuals`
: The state residuals. This is the state residual for the transition from $t = t$ to $t+1$ thus the last time step will be NA (since $T+1$ is past the data). State residuals do not exist for the `type="tt"` case (since this would required the expected value of $\mathbf{X}_t$ conditioned on data to $t + 1$).

`residuals`
: The residuals as a (n+m) x T matrix with `model.residuals` on top and `state.residuals` below.

`var.residuals`
: The variance of the model residuals and state residuals as a (n+m) x (n+m) x T matrix with the model residuals variance in rows/columns 1 to n and state residuals variances in rows/columns n+1 to n+m. The last time step will be all NA since the state residual is for $t = t$ to $t + 1$.

`std.residuals`
: The Cholesky standardized residuals as a (n+m) x T matrix. This is `residuals` multiplied by the inverse of the lower triangle of the Cholesky decomposition of `var.residuals`.

`mar.residuals`
: The marginal standardized residuals as a (n+m) x T matrix. This is `residuals` multiplied by the inverse of the diagonal matrix formed by the square-root of the diagonal of `var.residuals`.

`bchol.residuals`
: The Block Cholesky standardized residuals as a (n+m) x T matrix. This is `model.residuals` multiplied by the inverse of the lower triangle of the Cholesky decomposition of `var.residuals[1:n,1:n,]` and `state.residuals` multiplied by the inverse of the lower triangle of the Cholesky decomposition of `var.residuals[(n+1):(n+m),(n+1):(n+m),]`.

`E.obs.residuals`
: The expected value of the model residuals conditioned on the observed data. Returned as a n x T matrix. For observed data, this will be the observed model residuals. For unobserved data, this will be 0 if $\mathbf{R}$ is diagonal but non-zero if $\mathbf{R}$ is non-diagonal. See `MARSSresiduals.tT()`.

`var.obs.residuals`
: The variance of the model residuals conditioned on the observed data. Returned as a n x n x T matrix. For observed data, this will be 0. See `MARSSresiduals.tT()`.

`msg`
: Any warning messages. This will be printed unless Object$control$trace = -1 (suppress all error messages).

Author(s)

Eli Holmes, NOAA, Seattle, USA.

References

Holmes, E. E. 2014. Computation of standardized residuals for (MARSS) models. Technical Report. arXiv:1411.0045.

See also the discussion and references in `MARSSresiduals.tT()`, `MARSSresiduals.tt1()` and `MARSSresiduals.tt()`.

See Also

`residuals.marssMLE()`, `MARSSresiduals.tT()`, `MARSSresiduals.tt1()`, `plot.marssMLE()`

Examples

```
dat <- t(harborSeal)
dat <- dat[c(2,11),]
fit <- MARSS(dat)

#state smoothed residuals
state.resids1 <- MARSSresiduals(fit, type="tT")$state.residuals
#this is the same as
states <- fit$states
Q <- coef(fit, type="matrix")$Q
state.resids2 <- states[,2:30]-states[,1:29]-matrix(coef(fit,type="matrix")$U,2,29)
#compare the two
cbind(t(state.resids1[,-30]), t(state.resids2))

#normalize to variance of 1
state.resids1 <- MARSSresiduals(fit, type="tT", normalize=TRUE)$state.residuals
state.resids2 <- (solve(t(chol(Q))) %*% state.resids2)
cbind(t(state.resids1[,-30]), t(state.resids2))

#one-step-ahead standardized residuals
MARSSresiduals(fit, type="tt1")$std.residuals
```

---

marssResiduals-class *Class "marssResiduals"*

Description

marssResiduals are the objects returned by residuals.marssMLE in the package MARSS-package. It is a data frame in tibble format (but not tibble class).

standardization

- "Cholesky" means it is standardized by the Cholesky transformation of the full variance-covariance matrix of the model and state residuals.
- "marginal" means that the residual is standardized by its standard deviation, i.e. the square root of the value on the diagonal of the variance-covariance matrix of the model and state residuals.

type

- "tT" means the fitted values are computed conditioned on all the data. See fitted() with type="ytT" or type="xtT".
- "tt1" means the fitted values are computed conditioned on the data from $t = 1$ to $t - 1$. See fitted() with type="ytt1" or type="xtt1".

Author(s)

Eli Holmes, NOAA, Seattle, USA

See Also

residuals.marssMLE(), MARSSresiduals()

MARSSresiduals.tT *MARSS Smoothed Residuals*

Description

Calculates the standardized (or auxiliary) smoothed residuals sensu Harvey, Koopman and Penzer (1998). The expected values and variance for missing (or left-out) data are also returned (Holmes 2014). Not exported. Access this function with `MARSSresiduals(object, type="tT")`. At time t (in the returned matrices), the model residuals are for time t, while the state residuals are for the transition from t to $t+1$ following the convention in Harvey, Koopman and Penzer (1998).

Usage

```
MARSSresiduals.tT(object, Harvey = FALSE, normalize = FALSE,
    silent = FALSE, fun.kf = c("MARSSkfas", "MARSSkfss"))
```

Arguments

`object`	An object of class marssMLE.
`Harvey`	TRUE/FALSE. Use the Harvey et al. (1998) algorithm or use the Holmes (2014) algorithm. The values are the same except for missing values.
`normalize`	TRUE/FALSE See details.
`silent`	If TRUE, don't print inversion warnings.
`fun.kf`	Kalman filter function to use. Can be ignored.

Details

This function returns the raw, the Cholesky standardized and the marginal standardized smoothed model and state residuals. 'smoothed' means conditioned on all the observed data and a set of parameters. These are the residuals presented in Harvey, Koopman and Penzer (1998) pages 112-113, with the addition of the values for unobserved data (Holmes 2014). If Harvey=TRUE, the function uses the algorithm on page 112 of Harvey, Koopman and Penzer (1998) to compute the conditional residuals and variance of the residuals. If Harvey=FALSE, the function uses the equations in the technical report (Holmes 2014). Unlike the innovations residuals, the smoothed residuals are autocorrelated (section 4.1 in Harvey and Koopman 1992) and thus an ACF test on these residuals would not reveal model inadequacy.

The residuals matrix has a value for each time step. The residuals in column t rows 1 to n are the model residuals associated with the data at time t. The residuals in rows n+1 to n+m are the state residuals associated with the transition from $\mathbf{x}_t$ to $\mathbf{x}_{t+1}$, not the transition from $\mathbf{x}_{t-1}$ to $\mathbf{x}_t$. Because $\mathbf{x}_{t+1}$ does not exist at time T, the state residuals and associated variances at time T are NA.

Below the conditional residuals and their variance are discussed. The random variables are capitalized and the realizations from the random variables are lower case. The random variables are $\mathbf{X}$,

$\mathbf{Y}$, $\mathbf{V}$ and $\mathbf{W}$. There are two types of $\mathbf{Y}$. The observed $\mathbf{Y}$ that are used to estimate the states $\mathbf{x}$. These are termed $\mathbf{Y}^{(1)}$. The unobserved $\mathbf{Y}$ are termed $\mathbf{Y}^{(2)}$. These are not used to estimate the states $\mathbf{x}$ and we may or may not know the values of $\mathbf{y}^{(2)}$. Typically we treat $\mathbf{y}^{(2)}$ as unknown but it may be known but we did not include it in our model fitting. Note that the model parameters Θ are treated as fixed or known. The 'fitting' does not involve estimating Θ; it involves estimating $\mathbf{x}$. All MARSS parameters can be time varying but the t subscripts are left off parameters to reduce clutter.

Model residuals

$\mathbf{v}_t$ is the difference between the data and the predicted data at time t given $\mathbf{x}_t$:

$$\mathbf{v}_t = \mathbf{y}_t - \mathbf{Z}\mathbf{x}_t - \mathbf{a} - \mathbf{D}\mathbf{d}_t$$

$\mathbf{x}_t$ is unknown (hidden) and our data are one realization of $\mathbf{y}_t$. The observed model residuals $\hat{\mathbf{v}}_t$ are the difference between the observed data and the predicted data at time t using the fitted model. `MARSSresiduals.tT` fits the model using all the data, thus

$$\hat{\mathbf{v}}_t = \mathbf{y}_t - \mathbf{Z}\mathbf{x}_t^T - \mathbf{a} - \mathbf{D}\mathbf{d}_t$$

where $\mathbf{x}_t^T$ is the expected value of $\mathbf{X}_t$ conditioned on the data from 1 to T (all the data), i.e. the Kalman smoother estimate of the states at time t. $\mathbf{y}_t$ are your data and missing values will appear as NA in the observed model residuals. These are returned as `model.residuals` and rows 1 to n of `residuals`.

`res1` and `res2` in the code below will be the same.

```
dat = t(harborSeal)[2:3,]
fit = MARSS(dat)
Z = coef(fit, type="matrix")$Z
A = coef(fit, type="matrix")$A
res1 = dat - Z %*% fit$states - A %*% matrix(1,1,ncol(dat))
res2 = MARSSresiduals(fit, type="tT")$model.residuals
```

State residuals

$\mathbf{w}_{t+1}$ are the difference between the state at time $t+1$ and the expected value of the state at time $t+1$ given the state at time t:

$$\mathbf{w}_{t+1} = \mathbf{x}_{t+1} - \mathbf{B}\mathbf{x}_t - \mathbf{u} - \mathbf{C}\mathbf{c}_{t+1}$$

The estimated state residuals $\hat{\mathbf{w}}_{t+1}$ are the difference between estimate of $\mathbf{x}_{t+1}$ minus the estimate using $\mathbf{x}_t$.

$$\hat{\mathbf{w}}_{t+1} = \mathbf{x}_{t+1}^T - \mathbf{B}\mathbf{x}_t^T - \mathbf{u} - \mathbf{C}\mathbf{c}_{t+1}$$

where $\mathbf{x}_{t+1}^T$ is the Kalman smoother estimate of the states at time $t+1$ and $\mathbf{x}_t^T$ is the Kalman smoother estimate of the states at time t. The estimated state residuals $\mathbf{w}_{t+1}$ are returned in `state.residuals` and rows $n+1$ to $n+m$ of `residuals`. `state.residuals[,t]` is $\mathbf{w}_{t+1}$ (notice time subscript difference). There are no NAs in the estimated state residuals as an estimate of the state exists whether or not there are associated data.

`res1` and `res2` in the code below will be the same.

```
dat <- t(harborSeal)[2:3,]
TT <- ncol(dat)
fit <- MARSS(dat)
B <- coef(fit, type="matrix")$B
U <- coef(fit, type="matrix")$U
statestp1 <- MARSSkf(fit)$xtT[,2:TT]
statest <- MARSSkf(fit)$xtT[,1:(TT-1)]
res1 <- statestp1 - B %*% statest - U %*% matrix(1,1,TT-1)
res2 <- MARSSresiduals(fit, type="tT")$state.residuals[,1:(TT-1)]
```

Note that the state residual at the last time step (not shown) will be NA because it is the residual associated with $\mathbf{x}_T$ to $\mathbf{x}_{T+1}$ and $T+1$ is beyond the data. Similarly, the variance matrix at the last time step will have NAs for the same reason.

Variance of the residuals

In a state-space model, $\mathbf{X}$ and $\mathbf{Y}$ are stochastic, and the model and state residuals are random variables $\hat{\mathbf{V}}_t$ and $\hat{\mathbf{W}}_{t+1}$. To evaluate the residuals we observed (with $\mathbf{y}^{(1)}$), we use the joint distribution of $\hat{\mathbf{V}}_t, \hat{\mathbf{W}}_{t+1}$ across all the different possible data sets that our MARSS equations with parameters Θ might generate. Denote the matrix of $\hat{\mathbf{V}}_t, \hat{\mathbf{W}}_{t+1}$, as $\widehat{\mathcal{E}}_t$. That distribution has an expected value (mean) and variance:

$$\mathrm{E}[\widehat{\mathcal{E}}_t] = 0; \mathrm{var}[\widehat{\mathcal{E}}_t] = \hat{\Sigma}_t$$

Our observed residuals (returned in `residuals`) are one sample from this distribution. To standardize the observed residuals, we will use $\hat{\Sigma}_t$. $\hat{\Sigma}_t$ is returned in `var.residuals`. Rows/columns 1 to n are the conditional variances of the model residuals and rows/columns $n+1$ to $n+m$ are the conditional variances of the state residuals. The off-diagonal blocks are the covariances between the two types of residuals.

Standardized residuals

`MARSSresiduals` will return the Cholesky standardized residuals sensu Harvey et al. (1998) in `std.residuals` for outlier and shock detection. These are the model and state residuals multiplied by the inverse of the lower triangle of the Cholesky decomposition of `var.residuals` (note `chol()` in R returns the upper triangle thus a transpose is needed). The standardized model residuals are set to NA when there are missing data. The standardized state residuals however always exist since the expected value of the states exist without data. The calculation of the standardized residuals for both the observations and states requires the full residuals variance matrix. Since the state residuals variance is NA at the last time step, the standardized residual in the last time step will be all NA (for both model and state residuals).

The interpretation of the Cholesky standardized residuals is not straight-forward when the $\mathbf{Q}$ and $\mathbf{R}$ variance-covariance matrices are non-diagonal. The residuals which were generated by a non-diagonal variance-covariance matrices are transformed into orthogonal residuals in $\mathrm{MVN}(0, \mathbf{I})$ space. For example, if v is 2x2 correlated errors with variance-covariance matrix R. The transformed residuals (from this function) for the i-th row of $\mathbf{v}$ is a combination of the row 1 effect and the row 1 effect plus the row 2 effect. So in this case, row 2 of the transformed residuals would not be regarded as solely the row 2 residual but rather how different row 2 is from row 1, relative to expected. If the errors are highly correlated, then the transformed residuals can look rather non-intuitive.

The marginal standardized residuals are returned in `mar.residuals`. These are the model and state residuals multiplied by the inverse of the diagonal matrix formed by the square root of the diagonal

of `var.residuals`. These residuals will be correlated (across the residuals at time t) but are easier to interpret when $\mathbf{Q}$ and $\mathbf{R}$ are non-diagonal.

The Block Cholesky standardized residuals are like the Cholesky standardized residuals except that the full variance-covariance matrix is not used, only the variance-covariance matrix for the model or state residuals (respectively) is used for standardization. For the model residuals, the Block Cholesky standardized residuals will be the same as the Cholesky standardized residuals because the upper triangle of the lower triangle of the Cholesky decomposition (which is what we standardize by) is all zero. For the state residuals, the Block Cholesky standardization will be different because Block Cholesky standardization treats the model and state residuals as independent (which they are not in the smoothations case).

Normalized residuals

If `normalize=FALSE`, the unconditional variance of $\mathbf{V}_t$ and $\mathbf{W}_t$ are $\mathbf{R}$ and $\mathbf{Q}$ and the model is assumed to be written as

$$\mathbf{y}_t = \mathbf{Z}\mathbf{x}_t + \mathbf{a} + \mathbf{v}_t$$

$$\mathbf{x}_t = \mathbf{B}\mathbf{x}_{t-1} + \mathbf{u} + \mathbf{w}_t$$

If normalize=TRUE, the model is assumed to be written

$$\mathbf{y}_t = \mathbf{Z}\mathbf{x}_t + \mathbf{a} + \mathbf{H}\mathbf{v}_t$$

$$\mathbf{x}_t = \mathbf{B}\mathbf{x}_{t-1} + \mathbf{u} + \mathbf{G}\mathbf{w}_t$$

with the variance of $\mathbf{V}_t$ and $\mathbf{W}_t$ equal to $\mathbf{I}$ (identity).

`MARSSresiduals.tT` returns the residuals defined as in the first equations. To get the residuals defined as Harvey et al. (1998) define them (second equations), then use `normalize=TRUE`. In that case the unconditional variance of residuals will be $\mathbf{I}$ instead of $\mathbf{Q}$ and $\mathbf{R}$.

Missing or left-out data

$\mathrm{E}[\widehat{\mathcal{E}}_t]$ and $\mathrm{var}[\widehat{\mathcal{E}}_t]$ are for the distribution across all possible $\mathbf{X}$ and $\mathbf{Y}$. We can also compute the expected value and variance conditioned on a specific value of $\mathbf{Y}$, the one we observed $\mathbf{y}^{(1)}$ (Holmes 2014). If there are no missing values, this is not very interesting as $\mathrm{E}[\hat{\mathbf{V}}_t|\mathbf{y}^{(1)}] = \hat{\mathbf{v}}_t$ and $\mathrm{var}[\hat{\mathbf{V}}_t|\mathbf{y}^{(1)}] = 0$. If we have data that are missing because we left them out, however, $\mathrm{E}[\hat{\mathbf{V}}_t|\mathbf{y}^{(1)}]$ and $\mathrm{var}[\hat{\mathbf{V}}_t|\mathbf{y}^{(1)}]$ are the values we need to evaluate whether the left-out data are unusual relative to what you expect given the data you did collect.

`E.obs.residuals` is the conditional expected value $\mathrm{E}[\hat{\mathbf{V}}|\mathbf{y}^{(1)}]$ (notice small $\mathbf{y}$). It is

$$\mathrm{E}[\mathbf{Y}_t|\mathbf{y}^{(1)}] - \mathbf{Z}\mathbf{x}_t^T - \mathbf{a}$$

It is similar to $\hat{\mathbf{v}}_t$. The difference is the $\mathbf{y}$ term. $\mathrm{E}[\mathbf{Y}_t^{(1)}|\mathbf{y}^{(1)}]$ is $\mathbf{y}_t^{(1)}$ for the non-missing values. For the missing values, the value depends on $\mathbf{R}$. If $\mathbf{R}$ is diagonal, $\mathrm{E}[\mathbf{Y}_t^{(2)}|\mathbf{y}^{(1)}]$ is $\mathbf{Z}\mathbf{x}_t^T + \mathbf{a}$ and the expected residual value is 0. If $\mathbf{R}$ is non-diagonal however, it will be non-zero.

`var.obs.residuals` is the conditional variance $\mathrm{var}[\hat{\mathbf{V}}|\mathbf{y}^{(1)}]$ (eqn 24 in Holmes (2014)). For the non-missing values, this variance is 0 since $\hat{\mathbf{V}}|\mathbf{y}^{(1)}$ is a fixed value. For the missing values, $\hat{\mathbf{V}}|\mathbf{y}^{(1)}$ is not fixed because $\mathbf{Y}^{(2)}$ is a random variable. For these values, the variance of $\hat{\mathbf{V}}|\mathbf{y}^{(1)}$ is determined by the variance of $\mathbf{Y}^{(2)}$ conditioned on $\mathbf{Y}^{(1)} = \mathbf{y}^{(1)}$. This variance matrix is returned in `var.obs.residuals`. The variance of $\hat{\mathbf{W}}|\mathbf{y}^{(1)}$ is 0 and thus is not included.

The variance $\mathrm{var}[\hat{\mathbf{V}}_t|\mathbf{Y}^{(1)}]$ (uppercase $\mathbf{Y}$) returned in the 1 to n rows/columns of `var.residuals` may also be of interest depending on what you are investigating with regards to missing values. For

example, it may be of interest in a simulation study or cases where you have multiple replicated $\mathbf{Y}$ data sets. `var.residuals` would allow you to determine if the left-out residuals are unusual with regards to what you would expect for left-out data in that location of the $\mathbf{Y}$ matrix but not specifically relative to the data you did collect. If $\mathbf{R}$ is non-diagonal and the $\mathbf{y}^{(1)}$ and $\mathbf{y}^{(2)}$ are highly correlated, the variance of var$[\hat{\mathbf{V}}_t|\mathbf{Y}^{(1)}]$ and variance of var$[\hat{\mathbf{V}}_t|\mathbf{y}^{(1)}]$ for the left-out data would be quite different. In the latter, the variance is low because $\mathbf{y}^{(1)}$ has strong information about $\mathbf{y}^{(2)}$. In the former, we integrate over $\mathbf{Y}^{(1)}$ and the variance could be high (depending on the parameters).

Note, if `Harvey=TRUE` then the rows and columns of `var.residuals` corresponding to missing values will be NA. This is because the Harvey et al. algorithm does not compute the residual variance for missing values.

Value

A list with the following components

`model.residuals`
: The the observed smoothed model residuals: data minus the model predictions conditioned on all observed data. This is different than the Kalman filter innovations which use on the data up to time $t-1$ for the predictions. See details.

`state.residuals`
: The smoothed state residuals $\mathbf{x}_{t+1}^T - \mathbf{Z}\mathbf{x}_t^T - \mathbf{u}$. The last time step will be NA because the last step would be for T to T+1 (past the end of the data).

`residuals`
: The residuals conditioned on the observed data. Returned as a (n+m) x T matrix with `model.residuals` in rows 1 to n and `state.residuals` in rows n+1 to n+m. NAs will appear in rows 1 to n in the places where data are missing.

`var.residuals`
: The joint variance of the model and state residuals conditioned on observed data. Returned as a (n+m) x (n+m) x T matrix. For Harvey=FALSE, this is Holmes (2014) equation 57. For Harvey=TRUE, this is the residual variance in eqn. 24, page 113, in Harvey et al. (1998). They are identical except for missing values, for those Harvey=TRUE returns 0s. For the state residual variance, the last time step will be all NA because the last step would be for T to T+1 (past the end of the data).

`std.residuals`
: The Cholesky standardized residuals as a (n+m) x T matrix. This is `residuals` multiplied by the inverse of the lower triangle of the Cholesky decomposition of `var.residuals`. The model standardized residuals associated with the missing data are replaced with NA.

`mar.residuals`
: The marginal standardized residuals as a (n+m) x T matrix. This is `residuals` multiplied by the inverse of the diagonal matrix formed by the square-root of the diagonal of `var.residuals`. The model marginal residuals associated with the missing data are replaced with NA.

`bchol.residuals`
: The Block Cholesky standardized residuals as a (n+m) x T matrix. This is `model.residuals` multiplied by the inverse of the lower triangle of the Cholesky decomposition of `var.residuals[1:n,1:n,]` and `state.residuals` multiplied by the inverse of the lower triangle of the Cholesky decomposition of `var.residuals[(n+1):(n+m),(n+1):(n+m),]`.

`E.obs.residuals`
: The expected value of the model residuals conditioned on the observed data. Returned as a n x T matrix. For observed data, this will be the observed residuals (values in `model.residuals`). For unobserved data, this will be 0 if $\mathbf{R}$ is diagonal but non-zero if $\mathbf{R}$ is non-diagonal. See details.

`var.obs.residuals`
: The variance of the model residuals conditioned on the observed data. Returned as a n x n x T matrix. For observed data, this will be 0. See details.

`msg`
: Any warning messages. This will be printed unless Object$control$trace = -1 (suppress all error messages).

Author(s)

Eli Holmes, NOAA, Seattle, USA.

References

Harvey, A., S. J. Koopman, and J. Penzer. 1998. Messy time series: a unified approach. Advances in Econometrics 13: 103-144 (see page 112-113). Equation 21 is the Kalman eqns. Eqn 23 and 24 is the backward recursion to compute the smoothations. This function uses the MARSSkf output for eqn 21 and then implements the backwards recursion in equation 23 and equation 24. Pages 120-134 discuss the use of standardized residuals for outlier and structural break detection.

de Jong, P. and J. Penzer. 1998. Diagnosing shocks in time series. Journal of the American Statistical Association 93: 796-806. This one shows the same equations; see eqn 6. This paper mentions the scaling based on the inverse of the sqrt (Cholesky decomposition) of the variance-covariance matrix for the residuals (model and state together). This is in the right column, half-way down on page 800.

Koopman, S. J., N. Shephard, and J. A. Doornik. 1999. Statistical algorithms for models in state space using SsfPack 2.2. Econometrics Journal 2: 113-166. (see pages 147-148).

Harvey, A. and S. J. Koopman. 1992. Diagnostic checking of unobserved-components time series models. Journal of Business & Economic Statistics 4: 377-389.

Holmes, E. E. 2014. Computation of standardized residuals for (MARSS) models. Technical Report. arXiv:1411.0045.

See Also

`MARSSresiduals()`, `MARSSresiduals.tt1()`, `fitted.marssMLE()`, `plot.marssMLE()`

Examples

```
dat <- t(harborSeal)
dat <- dat[c(2,11),]
fit <- MARSS(dat)

#state residuals
state.resids1 <- MARSSresiduals(fit, type="tT")$state.residuals
#this is the same as hatx_t-(hatx_{t-1}+u)
states <- fit$states
state.resids2 <- states[,2:30]-states[,1:29]-matrix(coef(fit,type="matrix")$U,2,29)
```

```
#compare the two
cbind(t(state.resids1[,-30]), t(state.resids2))

#normalize the state residuals to a variance of 1
Q <- coef(fit,type="matrix")$Q
state.resids1 <- MARSSresiduals(fit, type="tT", normalize=TRUE)$state.residuals
state.resids2 <- (solve(t(chol(Q))) %*% state.resids2)
cbind(t(state.resids1[,-30]), t(state.resids2))

#Cholesky standardized (by joint variance) model & state residuals
MARSSresiduals(fit, type="tT")$std.residuals

# Returns residuals in a data frame in long form
residuals(fit, type="tT")
```

MARSSresiduals.tt *MARSS Contemporaneous Residuals*

Description

Calculates the standardized (or auxiliary) contemporaneous residuals, aka the residuals and their variance conditioned on the data up to time t. Contemporaneous residuals are only for the observations. Not exported. Access this function with `MARSSresiduals(object, type="tt")`.

Usage

```
MARSSresiduals.tt(object, method = c("SS"), normalize = FALSE,
    silent = FALSE, fun.kf = c("MARSSkfas", "MARSSkfss"))
```

Arguments

`object`	An object of class `marssMLE`.
`method`	Algorithm to use. Currently only "SS".
`normalize`	TRUE/FALSE See details.
`silent`	If TRUE, don't print inversion warnings.
`fun.kf`	Can be ignored. This will change the Kalman filter/smoother function from the value in object$fun.kf if desired.

Details

This function returns the conditional expected value (mean) and variance of the model contemporaneous residuals. 'conditional' means in this context, conditioned on the observed data up to time t and a set of parameters.

Model residuals

$\mathbf{v}_t$ is the difference between the data and the predicted data at time t given $\mathbf{x}_t$:

$$\mathbf{v}_t = \mathbf{y}_t - \mathbf{Z}\mathbf{x}_t - \mathbf{a} - \mathbf{d}\mathbf{d}_t$$

The observed model residuals $\hat{\mathbf{v}}_t$ are the difference between the observed data and the predicted data at time t using the fitted model. `MARSSresiduals.tt` fits the model using the data up to time t. So

$$\hat{\mathbf{v}}_t = \mathbf{y}_t - \mathbf{Z}\mathbf{x}_t^t - \mathbf{a} - \mathbf{D}\mathbf{d}_t$$

where $\mathbf{x}_t^t$ is the expected value of $\mathbf{X}_t$ conditioned on the data from 1 to t from the Kalman filter. $\mathbf{y}_t$ are your data and missing values will appear as NA. These will be returned in `residuals`.

`var.residuals` returned by the function is the conditional variance of the residuals conditioned on the data up to t and the parameter set Θ. The conditional variance is

$$\hat{\Sigma}_t = \mathbf{R} + \mathbf{Z}\mathbf{V}_t^t\mathbf{Z}^\top$$

where $\mathbf{V}_t^t$ is the variance of $\mathbf{X}_t$ conditioned on the data up to time t. This is returned by `MARSSkfss` in `Vtt`.

Standardized residuals

`std.residuals` are Cholesky standardized residuals. These are the residuals multiplied by the inverse of the lower triangle of the Cholesky decomposition of the variance matrix of the residuals:

$$\hat{\Sigma}_t^{-1/2}\hat{\mathbf{v}}_t$$

. These residuals are uncorrelated unlike marginal residuals.

The interpretation of the Cholesky standardized residuals is not straight-forward when the $\mathbf{Q}$ and $\mathbf{R}$ variance-covariance matrices are non-diagonal. The residuals which were generated by a non-diagonal variance-covariance matrices are transformed into orthogonal residuals in MVN$(0, \mathbf{I})$ space. For example, if v is 2x2 correlated errors with variance-covariance matrix R. The transformed residuals (from this function) for the i-th row of v is a combination of the row 1 effect and the row 1 effect plus the row 2 effect. So in this case, row 2 of the transformed residuals would not be regarded as solely the row 2 residual but rather how different row 2 is from row 1, relative to expected. If the errors are highly correlated, then the Cholesky standardized residuals can look rather non-intuitive.

`mar.residuals` are the marginal standardized residuals. These are the residuals multiplied by the inverse of the diagonal matrix formed from the square-root of the diagonal of the variance matrix of the residuals:

$$\mathrm{dg}(\hat{\Sigma}_t)^{-1/2}\hat{\mathbf{v}}_t$$

, where 'dg(A)' is the square matrix formed from the diagonal of A, aka `diag(diag(A))`. These residuals will be correlated if the variance matrix is non-diagonal.

Normalized residuals

If `normalize=FALSE`, the unconditional variance of $\mathbf{V}_t$ and $\mathbf{W}_t$ are $\mathbf{R}$ and $\mathbf{Q}$ and the model is assumed to be written as

$$\mathbf{y}_t = \mathbf{Z}\mathbf{x}_t + \mathbf{a} + \mathbf{v}_t$$

$$\mathbf{x}_t = \mathbf{B}\mathbf{x}_{t-1} + \mathbf{u} + \mathbf{w}_t$$

If normalize=TRUE, the model is assumed to be written

$$\mathbf{y}_t = \mathbf{Z}\mathbf{x}_t + \mathbf{a} + \mathbf{H}\mathbf{v}_t$$

$$\mathbf{x}_t = \mathbf{B}\mathbf{x}_{t-1} + \mathbf{u} + \mathbf{G}\mathbf{w}_t$$

with the variance of $\mathbf{V}_t$ and $\mathbf{W}_t$ equal to $\mathbf{I}$ (identity).

`MARSSresiduals()` returns the residuals defined as in the first equations. To get normalized residuals (second equation) as used in Harvey et al. (1998), then use `normalize=TRUE`. In that case the unconditional variance of residuals will be $\mathbf{I}$ instead of $\mathbf{R}$ and $\mathbf{Q}$. Note, that the normalized residuals are not the same as the standardized residuals. In former, the unconditional residuals have a variance of $\mathbf{I}$ while in the latter it is the conditional residuals that have a variance of $\mathbf{I}$.

Value

A list with the following components

`model.residuals`
: The observed contemporaneous model residuals: data minus the model predictions conditioned on the data 1 to t. A n x T matrix. NAs will appear where the data are missing.

`state.residuals`
: All NA. There are no contemporaneous residuals for the states.

`residuals`
: The residuals. `model.residuals` are in rows 1:n and `state.residuals` are in rows n+1:n+m.

`var.residuals`
: The joint variance of the residuals conditioned on observed data from 1 to t-. This only has values in the 1:n,1:n upper block for the model residuals.

`std.residuals`
: The Cholesky standardized residuals as a n+m x T matrix. This is `residuals` multiplied by the inverse of the lower triangle of the Cholesky decomposition of `var.residuals`. The model standardized residuals associated with the missing data are replaced with NA. Note because the contemporaneous state residuals do not exist, rows n+1:n+m are all NA.

`mar.residuals`
: The marginal standardized residuals as a n+m x T matrix. This is `residuals` multiplied by the inverse of the diagonal matrix formed by the square-root of the diagonal of `var.residuals`. The model marginal residuals associated with the missing data are replaced with NA.

`bchol.residuals`
: Because state residuals do not exist, this will be equivalent to the Cholesky standardized residuals, `std.residuals`.

`E.obs.residuals`
: The expected value of the model residuals conditioned on the observed data 1 to t. Returned as a n x T matrix.

`var.obs.residuals`
: The variance of the model residuals conditioned on the observed data. Returned as a n x n x T matrix. For observed data, this will be 0. See `MARSSresiduals.tT()` for a discussion of these residuals and where they might be used.

`msg`
: Any warning messages. This will be printed unless Object$control$trace = -1 (suppress all error messages).

Author(s)

Eli Holmes, NOAA, Seattle, USA.

References

Holmes, E. E. 2014. Computation of standardized residuals for (MARSS) models. Technical Report. arXiv:1411.0045.

See Also

MARSSresiduals.tT(), MARSSresiduals.tt1(), fitted.marssMLE(), plot.marssMLE()

Examples

```
dat <- t(harborSeal)
dat <- dat[c(2,11),]
fit <- MARSS(dat)

# Returns a matrix
MARSSresiduals(fit, type="tt")$std.residuals
# Returns a data frame in long form
residuals(fit, type="tt")
```

MARSSresiduals.tt1 *MARSS One-Step-Ahead Residuals*

Description

Calculates the standardized (or auxiliary) one-step-ahead residuals, aka the innovations residuals and their variance. Not exported. Access this function with `MARSSresiduals(object, type="tt1")`. To get the residuals as a data frame in long-form, use `residuals(object, type="tt1")`.

Usage

```
MARSSresiduals.tt1(object, method = c("SS"), normalize = FALSE,
    silent = FALSE, fun.kf = c("MARSSkfas", "MARSSkfss"))
```

Arguments

object	An object of class marssMLE.
method	Algorithm to use. Currently only "SS".
normalize	TRUE/FALSE See details.
silent	If TRUE, don't print inversion warnings.
fun.kf	Can be ignored. This will change the Kalman filter/smoother function from the value in object$fun.kf if desired.

Details

This function returns the conditional expected value (mean) and variance of the one-step-ahead residuals. 'conditional' means in this context, conditioned on the observed data up to time $t-1$ and a set of parameters.

Model residuals

$\mathbf{v}_t$ is the difference between the data and the predicted data at time t given $\mathbf{x}_t$:

$$\mathbf{v}_t = \mathbf{y}_t - \mathbf{Z}\mathbf{x}_t - \mathbf{a} - \mathbf{D}\mathbf{d}_t$$

The observed model residuals $\hat{\mathbf{v}}_t$ are the difference between the observed data and the predicted data at time t using the fitted model. `MARSSresiduals.tt1` fits the model using the data up to time $t-1$. So

$$\hat{\mathbf{v}}_t = \mathbf{y}_t - \mathbf{Z}\mathbf{x}_t^{t-1} - \mathbf{a} - \mathbf{D}\mathbf{d}_t$$

where $\mathbf{x}_t^{t-1}$ is the expected value of $\mathbf{X}_t$ conditioned on the data from \$t=1\$ to $t-1$ from the Kalman filter. $\mathbf{y}_t$ are your data and missing values will appear as NA.

State residuals

$\mathbf{w}_{t+1}$ are the difference between the state at time $t+1$ and the expected value of the state at time $t+1$ given the state at time t:

$$\mathbf{w}_{t+1} = \mathbf{x}_{t+1} - \mathbf{B}\mathbf{x}_t - \mathbf{u} - \mathbf{C}\mathbf{c}_{t+1}$$

The estimated state residuals $\hat{\mathbf{w}}_{t+1}$ are the difference between estimate of $\mathbf{x}_{t+1}$ minus the estimate using $\mathbf{x}_t$.

$$\hat{\mathbf{w}}_{t+1} = \mathbf{x}_{t+1}^{t+1} - \mathbf{B}\mathbf{x}_t^t - \mathbf{u} - \mathbf{C}\mathbf{c}_{t+1}$$

where $\mathbf{x}_{t+1}^{t+1}$ is the Kalman filter estimate of the states at time $t+1$ conditioned on the data up to time $t+1$ and $\mathbf{x}_t^t$ is the Kalman filter estimate of the states at time t conditioned on the data up to time t. The estimated state residuals $\mathbf{w}_{t+1}$ are returned in `state.residuals` and rows $n+1$ to $n+m$ of `residuals`. `state.residuals[,t]` is $\mathbf{w}_{t+1}$ (notice time subscript difference). There are no NAs in the estimated state residuals (except for the last time step) as an estimate of the state exists whether or not there are associated data.

`res1` and `res2` in the code below will be the same.

```
dat <- t(harborSeal)[2:3,]
TT <- ncol(dat)
fit <- MARSS(dat)
B <- coef(fit, type="matrix")$B
U <- coef(fit, type="matrix")$U
xt <- MARSSkfss(fit)$xtt[,1:(TT-1)] # t 1 to TT-1
xtp1 <- MARSSkfss(fit)$xtt[,2:TT] # t 2 to TT
res1 <- xtp1 - B %*% xt - U %*% matrix(1,1,TT-1)
res2 <- MARSSresiduals(fit, type="tt1")$state.residuals
```

Joint residual variance

In a state-space model, $\mathbf{X}$ and $\mathbf{Y}$ are stochastic, and the model and state residuals are random variables $\hat{\mathbf{V}}_t$ and $\hat{\mathbf{W}}_{t+1}$. The joint distribution of $\hat{\mathbf{V}}_t, \hat{\mathbf{W}}_{t+1}$ is the distribution across all the different

possible data sets that our MARSS equations with parameters Θ might generate. Denote the matrix of $\hat{\mathbf{V}}_t, \hat{\mathbf{W}}_{t+1}$, as $\widehat{\mathcal{E}}_t$. That distribution has an expected value (mean) and variance:

$$\mathrm{E}[\widehat{\mathcal{E}}_t] = 0; \mathrm{var}[\widehat{\mathcal{E}}_t] = \hat{\Sigma}_t$$

Our observed residuals `residuals` are one sample from this distribution. To standardize the observed residuals, we will use $\hat{\Sigma}_t$. $\hat{\Sigma}_t$ is returned in `var.residuals`. Rows/columns 1 to n are the conditional variances of the model residuals and rows/columns $n+1$ to $n+m$ are the conditional variances of the state residuals. The off-diagonal blocks are the covariances between the two types of residuals. For one-step-ahead residuals (unlike smoothation residuals MARSSresiduals.tT), the covariance is zero.

`var.residuals` returned by this function is the conditional variance of the residuals conditioned on the data up to $t-1$ and the parameter set Θ. The conditional variance for the model residuals is

$$\hat{\Sigma}_t = \mathbf{R} + \mathbf{Z}_t \mathbf{V}_t^{t-1} \mathbf{Z}_t^\top$$

where $\mathbf{V}_t^{t-1}$ is the variance of $\mathbf{X}_t$ conditioned on the data up to time $t-1$. This is returned by `MARSSkf` in `Vtt1`. The innovations variance is also returned in `Sigma` from `MARSSkf` and are used in the innovations form of the likelihood calculation.

Standardized residuals

`std.residuals` are Cholesky standardized residuals. These are the residuals multiplied by the inverse of the lower triangle of the Cholesky decomposition of the variance matrix of the residuals:

$$\hat{\Sigma}_t^{-1/2} \hat{\mathbf{v}}_t$$

These residuals are uncorrelated unlike marginal residuals.

The interpretation of the Cholesky standardized residuals is not straight-forward when the $\mathbf{Q}$ and $\mathbf{R}$ variance-covariance matrices are non-diagonal. The residuals which were generated by a non-diagonal variance-covariance matrices are transformed into orthogonal residuals in $\mathrm{MVN}(0, \mathbf{I})$ space. For example, if v is 2x2 correlated errors with variance-covariance matrix R. The transformed residuals (from this function) for the i-th row of v is a combination of the row 1 effect and the row 1 effect plus the row 2 effect. So in this case, row 2 of the transformed residuals would not be regarded as solely the row 2 residual but rather how different row 2 is from row 1, relative to expected. If the errors are highly correlated, then the Cholesky standardized residuals can look rather non-intuitive.

`mar.residuals` are the marginal standardized residuals. These are the residuals multiplied by the inverse of the diagonal matrix formed from the square-root of the diagonal of the variance matrix of the residuals:

$$\mathrm{dg}(\hat{\Sigma}_t)^{-1/2} \hat{\mathbf{v}}_t$$

, where 'dg(A)' is the square matrix formed from the diagonal of A, aka `diag(diag(A))`. These residuals will be correlated if the variance matrix is non-diagonal.

The Block Cholesky standardized residuals are like the Cholesky standardized residuals except that the full variance-covariance matrix is not used, only the variance-covariance matrix for the model or state residuals (respectively) is used for standardization. For the one-step-ahead case, the model and state residuals are independent (unlike in the smoothations case) thus the Cholesky and Block Cholesky standardized residuals will be identical (unlike in the smoothations case).

Normalized residuals

If `normalize=FALSE`, the unconditional variance of $\mathbf{V}_t$ and $\mathbf{W}_t$ are $\mathbf{R}$ and $\mathbf{Q}$ and the model is assumed to be written as

$$\mathbf{y}_t = \mathbf{Z}\mathbf{x}_t + \mathbf{a} + \mathbf{v}_t$$

$$\mathbf{x}_t = \mathbf{B}\mathbf{x}_{t-1} + \mathbf{u} + \mathbf{w}_t$$

If normalize=TRUE, the model is assumed to be written

$$\mathbf{y}_t = \mathbf{Z}\mathbf{x}_t + \mathbf{a} + \mathbf{H}\mathbf{v}_t$$

$$\mathbf{x}_t = \mathbf{B}\mathbf{x}_{t-1} + \mathbf{u} + \mathbf{G}\mathbf{w}_t$$

with the variance of $\mathbf{V}_t$ and $\mathbf{W}_t$ equal to $\mathbf{I}$ (identity).

`MARSSresiduals` returns the residuals defined as in the first equations. To get the residuals defined as Harvey et al. (1998) define them (second equations), then use `normalize=TRUE`. In that case the unconditional variance of residuals will be $\mathbf{I}$ instead of $\mathbf{Q}$ and $\mathbf{R}$. Note, that the normalized residuals are not the same as the standardized residuals. In former, the unconditional residuals have a variance of $\mathbf{I}$ while in the latter it is the conditional residuals that have a variance of $\mathbf{I}$.

Value

A list with the following components

`model.residuals`
: The the observed one-step-ahead model residuals: data minus the model predictions conditioned on the data $t = 1$ to $t - 1$. These are termed innovations. A n x T matrix. NAs will appear where the data are missing.

`state.residuals`
: The one-step-ahead state residuals $\mathbf{x}_{t+1}^{t+1} - \mathbf{B}\mathbf{x}_t^t - \mathbf{u}$. Note, state residual at time t is the transition from time $t = t$ to $t + 1$.

`residuals`
: The residuals conditioned on the observed data up to time $t - 1$. Returned as a (n+m) x T matrix with `model.residuals` in rows 1 to n and `state.residuals` in rows n+1 to n+m. NAs will appear in rows 1 to n in the places where data are missing.

`var.residuals`
: The joint variance of the one-step-ahead residuals. Returned as a n+m x n+m x T matrix.

`std.residuals`
: The Cholesky standardized residuals as a n+m x T matrix. This is `residuals` multiplied by the inverse of the lower triangle of the Cholesky decomposition of `var.residuals`. The model standardized residuals associated with the missing data are replaced with NA.

`mar.residuals`
: The marginal standardized residuals as a n+m x T matrix. This is `residuals` multiplied by the inverse of the diagonal matrix formed by the square-root of the diagonal of `var.residuals`. The model marginal residuals associated with the missing data are replaced with NA.

`bchol.residuals`
: The Block Cholesky standardized residuals as a (n+m) x T matrix. This is `model.residuals` multiplied by the inverse of the lower triangle of the Cholesky decomposition of `var.residuals[1:n,1:n,]` and `state.residuals` multiplied by the inverse of the lower triangle of the Cholesky decomposition of `var.residuals[(n+1):(n+m),(n+1):(n+m),]`.

`E.obs.residuals`
: The expected value of the model residuals conditioned on the observed data $t = 1$ to $t - 1$. Returned as a n x T matrix. Because all the data at time t are unobserved for the purpose of estimation (since conditioning is from $t = 1$ to $t - 1$), this will be all 0s (unlike the case where we condition on the data from $t = 1$ to T or to t). This and `var.obs.residuals` are included for completeness since they are returned for `MARSSresiduals.tT()`, but they are not relevant for one-step-ahead residuals. See the discussion there.

`var.obs.residuals`
: For one-step-ahead residuals, this will be the same as the 1:n, 1:n upper diagonal block in `var.residuals` since none of the t data affect the residuals at time t (the model residuals are conditioned only on the data up to $t - 1$). This is different for smoothation residuals which are conditioned on the data from $t = 1$ to T. This and `E.obs.residuals` are included for completeness since they are returned for `MARSSresiduals.tT()`, but they are not relevant for one-step-ahead residuals. See the discussion there. Note, also included as a code check. They are computed differently, but `var.obs.residuals` and `var.residuals` should always be the same.

`msg`
: Any warning messages. This will be printed unless `object$control$trace = -1` (suppress all error messages).

Author(s)

Eli Holmes, NOAA, Seattle, USA.

References

R. H. Shumway and D. S. Stoffer (2006). Section on the calculation of the likelihood of state-space models in Time series analysis and its applications. Springer-Verlag, New York.

Holmes, E. E. 2014. Computation of standardized residuals for (MARSS) models. Technical Report. arXiv:1411.0045.

See Also

`MARSSresiduals.tT()`, `MARSSresiduals.tt()`, `fitted.marssMLE()`, `plot.marssMLE()`

Examples

```
dat <- t(harborSeal)
dat <- dat[c(2,11),]
fit <- MARSS(dat)

MARSSresiduals(fit, type="tt1")$std.residuals
residuals(fit, type="tt1")
```

`MARSSsimulate` *Simulate Data from a MARSS Model*

Description

Generates simulated data from a MARSS model with specified parameter estimates. This is a base function in the `MARSS-package`.

Usage

```
MARSSsimulate(object, tSteps = NULL, nsim = 1, silent = TRUE,
    miss.loc = NULL)
```

Arguments

`object`	A fitted `marssMLE` object, as output by `MARSS()`.
`tSteps`	Number of time steps in each simulation. If left off, it is taken to be consistent with `MLEobj`.
`nsim`	Number of simulated data sets to generate.
`silent`	Suppresses progress bar.
`miss.loc`	Optional matrix specifying where to put missing values. See Details.

Details

Optional argument `miss.loc` is an array of dimensions n x tSteps x nsim, specifying where to put missing values in the simulated data. If missing, this would be constructed using `MLEobj$marss$data`. If the locations of the missing values are the same for all simulations, `miss.loc` can be a matrix of `dim=c(n, tSteps)` (the original data for example). The default, if `miss.loc` is left off, is that there are no missing values even if `MLEobj$marss$data` has missing values.

Value

`sim.states`	Array (dim m x tSteps x nsim) of state processes simulated from parameter estimates. m is the number of states (rows in X).
`sim.data`	Array (dim n x tSteps x nsim) of data simulated from parameter estimates. n is the number of rows of data (Y).
`MLEobj`	The `marssMLE` object from which the data were simulated.
`miss.loc`	Matrix identifying where missing values were placed. It should be exactly the same dimensions as the data matrix. The location of NAs in the miss.loc matrix indicate where the missing values are.
`tSteps`	Number of time steps in each simulation.
`nsim`	Number of simulated data sets generated.

Author(s)

Eli Holmes and Eric Ward, NOAA, Seattle, USA.

See Also

marssMODEL, marssMLE, MARSSboot()

Examples

```
d <- harborSeal[, c(2, 11)]
dat <- t(d)
fit <- MARSS(dat)

# simulate data that are the
# same length as original data and no missing data
sim.obj <- MARSSsimulate(fit, tSteps = dim(d)[1], nsim = 5)

# simulate data that are the
# same length as original data and have missing data in the same location
sim.obj <- MARSSsimulate(fit, tSteps = dim(d)[1], nsim = 5, miss.loc = dat)
```

---

plankton *Plankton Data Sets*

Description

Example plankton data sets for use in MARSS vignettes for the MARSS-package.

The lakeWAplankton data set consists for two data sets: lakeWAplanktonRaw and a dataset derived from the raw dataset, lakeWAplanktonTrans. lakeWAplanktonRaw is a 32-year time series (1962-1994) of monthly plankton counts from Lake Washington, Washington, USA. Columns 1 and 2 are year and month. Column 3 is temperature (C), column 4 is total phosphorous, and column 5 is pH. The next columns are the plankton counts in units of cells per mL for the phytoplankton and organisms per L for the zooplankton. Since MARSS functions require time to be across columns, these data matrices must be transposed before passing into MARSS functions.

lakeWAplanktonTrans is a transformed version of lakeWAplanktonRaw. Zeros have been replaced with NAs (missing). The logged (natural log) raw plankton counts have been standardized to a mean of zero and variance of 1 (so logged and then z-scored). Temperature, TP & pH were also z-scored but not logged (so z-score of the untransformed values for these covariates). The single missing temperature value was replaced with -1 and the single missing TP value was replaced with -0.3.

The Ives data are from Ives et al. (2003) for West Long Lake (the low planktivory case). The Ives data are unlogged. ivesDataLP and ivesDataByWeek are the same data with LP having the missing weeks in winter removed while in ByWeek, the missing values are left in. The phosporous column is the experimental input rate + the natural input rate for phosphorous, and Ives et al. used 0.1 for the natural input rate when no extra phosporous was added. The phosporous input rates for weeks with no sampling (and no experimental phosphorous input) have been filled with 0.1 in the "by week" data.

Usage

```
data(ivesDataLP)
data(ivesDataByWeek)
data(lakeWAplankton)
```

Format

The data are provided as a matrix with time running down the rows.

Source

ivesDataLP **and** ivesDataByWeek Ives, A. R. Dennis, B. Cottingham, K. L. Carpenter, S. R. (2003) Estimating community stability and ecological interactions from time-series data. Ecological Monographs, 73, 301-330.

lakeWAplanktonTrans Hampton, S. E. Scheuerell, M. D. Schindler, D. E. (2006) Coalescence in the Lake Washington story: Interaction strengths in a planktonic food web. Limnology and Oceanography, 51, 2042-2051.

lakeWAplanktonRaw Adapted from the Lake Washington database of Dr. W. T. Edmondson, as funded by the Andrew Mellon Foundation; data courtesy of Dr. Daniel Schindler, University of Washington, Seattle, WA.

Examples

```
str(ivesDataLP)
str(ivesDataByWeek)
```

---

plot.marssMLE *Plot MARSS MLE objects*

Description

Plots fitted observations and estimated states with confidence intervals using base R graphics (plot) and ggplot2 (autoplot). Diagnostic plots also shown. By default a subset of standard diagnostic plots are plotted. Individual plots can be plotted by passing in plot.type. If an individual plot is made using autoplot(), the ggplot object is returned which can be further manipulated.

Usage

```
## S3 method for class 'marssMLE'
plot(x, plot.type = c(
    "fitted.ytT", "fitted.ytt", "fitted.ytt1",
    "ytT", "ytt", "ytt1",
    "fitted.xtT", "fitted.xtt1",
    "xtT", "xtt", "xtt1",
    "model.resids.ytt1", "qqplot.model.resids.ytt1", "acf.model.resids.ytt1",
   "std.model.resids.ytt1", "qqplot.std.model.resids.ytt1", "acf.std.model.resids.ytt1",
    "model.resids.ytT", "qqplot.model.resids.ytT", "acf.model.resids.ytT",
   "std.model.resids.ytT", "qqplot.std.model.resids.ytT", "acf.std.model.resids.ytT",
    "model.resids.ytt", "qqplot.model.resids.ytt", "acf.model.resids.ytt",
   "std.model.resids.ytt", "qqplot.std.model.resids.ytt", "acf.std.model.resids.ytt",
    "state.resids.xtT", "qqplot.state.resids.xtT", "acf.state.resids.xtT",
   "std.state.resids.xtT", "qqplot.std.state.resids.xtT", "acf.std.state.resids.xtT",
```

```
    "residuals", "all"),
    form=c("marxss", "marss", "dfa"),
    standardization = c("Cholesky", "marginal", "Block.Cholesky"),
    conf.int=TRUE, conf.level=0.95, decorate=TRUE, pi.int = FALSE,
    plot.par = list(), ..., silent = FALSE)
## S3 method for class 'marssMLE'
autoplot(x, plot.type = c(
    "fitted.ytT", "fitted.ytt", "fitted.ytt1",
    "ytT", "ytt", "ytt1",
    "fitted.xtT", "fitted.xtt1",
    "xtT", "xtt", "xtt1",
    "model.resids.ytt1", "qqplot.model.resids.ytt1", "acf.model.resids.ytt1",
   "std.model.resids.ytt1", "qqplot.std.model.resids.ytt1", "acf.std.model.resids.ytt1",
    "model.resids.ytT", "qqplot.model.resids.ytT", "acf.model.resids.ytT",
   "std.model.resids.ytT", "qqplot.std.model.resids.ytT", "acf.std.model.resids.ytT",
    "model.resids.ytt", "qqplot.model.resids.ytt", "acf.model.resids.ytt",
   "std.model.resids.ytt", "qqplot.std.model.resids.ytt", "acf.std.model.resids.ytt",
    "state.resids.xtT", "qqplot.state.resids.xtT", "acf.state.resids.xtT",
   "std.state.resids.xtT", "qqplot.std.state.resids.xtT", "acf.std.state.resids.xtT",
    "residuals", "all"),
    form=c("marxss", "marss", "dfa"),
    standardization = c("Cholesky", "marginal", "Block.Cholesky"),
    conf.int=TRUE, conf.level=0.95, decorate=TRUE, pi.int = FALSE,
    fig.notes = TRUE, plot.par = list(), ..., silent = FALSE)
```

Arguments

x	A marssMLE object.
plot.type	Type of plot. If not passed in, a subset of the standard plots are drawn. See details for plot types.
standardization	The type of standardization to be used plots, if the user wants to specify a specific standardization. Otherwise Cholesky standardization is used.
form	Optional. Form of the model. This is normally taken from the form attribute of the MLE object (x), but the user can specify a different form.
conf.int	TRUE/FALSE. Whether to include a confidence interval.
pi.int	TRUE/FALSE. Whether to include a prediction interval on the observations plot
conf.level	Confidence level for CIs.
decorate	TRUE/FALSE. Add smoothing lines to residuals plots or qqline to qqplots and add data points plus residuals confidence intervals to states and observations plots.
plot.par	A list of plot parameters to adjust the look of the plots. See details.
fig.notes	Add notes to the bottom of the plots (only for autoplot().
silent	No console interaction or output.
...	Other arguments, not used.

Details

The plot types are as follows:

`"fitted.y"` This plots the fitted $\mathbf{y}$, which is the expected value of $\mathbf{Y}$ conditioned on the data from $t = 1$ to $t - 1$, t or T. It is $\mathbf{Z}\mathbf{x}_t^T + \mathbf{a}$. The data are plotted for reference but note that the lines and intervals are for new data not the observed data.

`"fitted.x"` This plots the fitted x, which is the expected value of $\mathbf{X}$ conditioned on the data from $t = 1$ to $t - 1$ or T. It is $B\mathrm{E}[\mathbf{X}_{t-1}|\mathbf{y}] + u$. The $\mathrm{E}[\mathbf{X}_t|\mathbf{y}]$ are plotted for reference but note that the lines and intervals are for new $\mathbf{x}$. This is not the estimated states; these are used for residuals calculations. If you want the state estimates use `xtT` (or `xtt`).

`"xtT"` The estimated states from the Kalman smoother (conditioned on all the data).

`"xtt1"` The estimated states conditioned on the data up to $t - 1$. Kalman filter output.

`"model.resids.ytT"`, `"model.resids.ytt1"`, `"model.resids.ytt"` Model residuals (data minus fitted y). `ytT` indicates smoothation residuals, `ytt1` indicates innovation residuals (the standard state-space residuals), and `ytt` are the residuals conditioned on data up to t.

`"state.resids.xtT"` State smoothation residuals (E(x(t) | xtT(t-1)) minus xtT(t)). The intervals are the CIs for the smoothation residuals not one-step-ahead residuals.

`"std"` `std` in front of any of the above plot names indicates that the plots are for the standardized residuals.

`"qqplot"` Visual normality test for the residuals, model or state.

`"acf"` ACF of the residuals. The only residuals that should be temporally independent are the innovation residuals: `acf.model.residuals.ytt1` and `acf.std.model.residuals.ytt1`. This ACF is a standard residuals diagnostic for state-space models. The other ACF plots will show temporal dependence and are not used for diagnostics.

`"ytT"` The expected value of $\mathbf{Y}$ conditioned on all the data. Use this for estimates of the missing data points. Note for non-missing $\mathbf{y}$ values, the expected value of $\mathbf{Y}$ is $\mathbf{y}$.

`"ytt"`, `ytt1` The expected value of $\mathbf{Y}$ conditioned on the data from 1 to t or $t - 1$.

The plot parameters can be passed in as a list to change the look of the plots. For `plot.marssMLE()`, the default is `plot.par = list(point.pch = 19, point.col = "blue", point.fill = "blue", point.size = 1, line.col = "black", line.size = 1, line.linetype = "solid", ci.col = "grey70", ci.border = NA, ci.lwd = 1, ci.lty = 1)`. For `autoplot.marssMLE`, the default is `plot.par = list(point.pch = 19, point.col = "blue", point.fill = "blue", point.size = 1, line.col = "black", line.size = 1, line.linetype = "solid", ci.fill = "grey70", ci.col = "grey70", ci.linetype = "solid", ci.linesize = 0, ci.alpha = 0.6)`.

Value

`autoplot()` will invisibly return the list of ggplot2 plot objects. Use `plts <- autoplot()` to obtain that list.

Author(s)

Eric Ward and Eli Holmes

Examples

```
data(harborSealWA)
model.list <- list( Z = as.factor(c(1, 1, 1, 1, 2)), R = "diagonal and equal")
fit <- MARSS(t(harborSealWA[, -1]), model = model.list)
plot(fit, plot.type = "fitted.ytT")

require(ggplot2)
autoplot(fit, plot.type = "fitted.ytT")

## Not run:
# DFA example
dfa <- MARSS(t(harborSealWA[, -1]), model = list(m = 2), form = "dfa")
plot(dfa, plot.type = "xtT")

## End(Not run)
```

---

plot.marssPredict *Plot MARSS Forecast and Predict objects*

Description

Plots forecasts with prediction (default) or confidence intervals using base R graphics (`plot`) and ggplot2 (`autoplot`). The plot function is built to mimic `plot.forecast` in the forecast package in terms of arguments and look.

Usage

```
## S3 method for class 'marssPredict'
plot(x, include, decorate = TRUE, main = NULL, showgap = TRUE,
    shaded = TRUE, shadebars = (x$h < 5 & x$h != 0), shadecols = NULL, col = 1,
    fcol = 4, pi.col = 1, pi.lty = 2, ylim = NULL,
    xlab = "", ylab = "", type = "l", flty = 1, flwd = 2, ...)
## S3 method for class 'marssPredict'
autoplot(x, include, decorate = TRUE, plot.par = list(),  ...)
```

Arguments

x	marssPredict produced by `forecast.marssMLE()` or `predict.marssMLE()`.
include	number of time step from the training data to include before the forecast. Default is all values.
main	Text to add to plot titles.
showgap	If showgap=FALSE, the gap between the training data and the forecasts is removed.
shaded	Whether prediction intervals should be shaded (TRUE) or lines (FALSE).

shadebars	Whether prediction intervals should be plotted as shaded bars (if TRUE) or a shaded polygon (if FALSE). Ignored if shaded=FALSE. Bars are plotted by default if there are fewer than five forecast horizons.
shadecols	Colors for shaded prediction intervals.
col	Color for the data line.
fcol	Color for the forecast line.
pi.col	If shaded=FALSE and PI=TRUE, the prediction intervals are plotted in this color.
pi.lty	If shaded=FALSE and PI=TRUE, the prediction intervals are plotted using this line type.
ylim	Limits on y-axis.
xlab	X-axis label.
ylab	Y-axis label.
type	Type of plot desired. As for plot.default.
flty	Line type for the forecast line.
flwd	Line width for the forecast line.
...	Other arguments, not used.
decorate	TRUE/FALSE. Add data points and CIs or PIs to the plots.
plot.par	A list of plot parameters to adjust the look of the plot. The default is `list(point.pch = 19, point.col = "blue", point.fill = "blue", point.size = 1, line.col = "black", line.size = 1, line.type = "solid", ci.fill = NULL, ci.col = NULL, ci.linetype = "blank", ci.linesize = 0, ci.alpha = 0.6, f.col = "#0000AA", f.linetype = "solid", f.linesize=0.5, theme = theme_bw())`.

Value

None. Plots are plotted

Author(s)

Eli Holmes and based off of `plot.forecast` in the forecast package written by Rob J Hyndman & Mitchell O'Hara-Wild.

See Also

`predict.marssMLE()`

Examples

```
data(harborSealWA)
dat <- t(harborSealWA[, -1])
fit <- MARSS(dat[1:2,])
fr <- predict(fit, n.ahead=10)
plot(fr, include=10)

# forecast.marssMLE does the same thing as predict with h
```

```
fr <- forecast(fit, n.ahead=10)
plot(fr)

# without h, predict will show the prediction intervals
fr <- predict(fit)
plot(fr)

# you can fit to a new set of data using the same model and same x0
fr <- predict(fit, newdata=list(y=dat[3:4,]), x0="use.model")
plot(fr)

# but you probably want to re-estimate x0
fr <- predict(fit, newdata=list(y=dat[3:4,]), x0="reestimate")
plot(fr)

# forecast; note h not n.ahead is used for forecast()
fr <- forecast(fit, h=10)
```

`plot.marssResiduals` *Plot MARSS marssResiduals objects*

Description

Plots residuals using the output from a `residuals()` call. By default all available residuals plots are plotted. Individual plots can be plotted by passing in `plot.type`. If an individual plot is made using `autoplot()`, the ggplot object is returned which can be further manipulated. Plots are only shown for those residual types in the `marssResiduals` object.

Usage

```
## S3 method for class 'marssResiduals'
plot(x, plot.type = c("all", "residuals", "qqplot", "acf"),
                  conf.int = TRUE, conf.level = 0.95, decorate = TRUE,
                  plot.par = list(), silent = FALSE, ...)
## S3 method for class 'marssResiduals'
autoplot(x,
           plot.type = c("all", "residuals", "qqplot", "acf"),
           conf.int = TRUE, conf.level = 0.95, decorate = TRUE,
           plot.par = list(),
           silent = FALSE)
```

Arguments

`x`	A `marssResiduals` object.
`plot.type`	Type of plot. If not passed in, all plots are drawn. See details for plot types.
`conf.int`	TRUE/FALSE. Whether to include a confidence interval.
`conf.level`	Confidence level for CIs.

decorate	TRUE/FALSE. Add smoothing lines to residuals plots or qqline to qqplots and add data points plus residuals confidence intervals to states and observations plots.
plot.par	A list of plot parameters to adjust the look of the plots. The default is list(point.pch = 19, point.col = "blue", point.fill = "blue", point.size = 1, line.col = "black", line.size = 1, line.linetype = "solid", ci.fill = "grey70", ci.col = "grey70", ci.linetype = "solid", ci.linesize = 0, ci.alpha = 0.6).
silent	No console interaction or output.
...	Not used.

Details

If `resids <- residuals(x)` is used (default) where `x` is a `marssMLE` object from a `MARSS()` call, then `resids` has the innovations residuals, or one-step-ahead residuals. These are what are commonly used for residuals diagnostics in state-space modeling. However, other types of residuals are possible for state-space models; see `MARSSresiduals()` for details. The plot function for `marssResiduals` objects will handle all types of residuals that might be in the `marssResiduals` object. However if you simply use the default behavior, `resids <- residuals(x)` and `plot(resids)`, you will get the standard model residuals diagnostics plots for state-space models, i.e. only model residuals plots and only plots for innovations model residuals (no smoothations model residuals).

The plot types are as follows:

`"all"` All the residuals in the residuals object plus QQ plots and ACF plots.

`"residuals"` Only residuals versus time.

`"qqplot"` Only QQ plots. Visual normality test for the residuals.

`"acf"` ACF of the residuals. If `x$type is "ytt1"`, these are the one-step-ahead (aka innovations) residuals and they should be temporally independent.

Value

If an individual plot is selected using `plot.type` and `autoplot()` is called, then the ggplot object is returned invisibly.

Author(s)

Eli Holmes

Examples

```
data(harborSealWA)
model.list <- list( Z = as.factor(c(1, 1, 1, 1, 2)), R = "diagonal and equal")
fit <- MARSS(t(harborSealWA[, -1]), model = model.list)
resids <- residuals(fit)

require(ggplot2)
# plots of residuals versus time, QQ-norm plot, and ACF
autoplot(resids)
```

```
# only the ACF plots
# autoplot(resids, plot.type = "acf")
```

---

population-count-data *Population Data Sets*

Description

Example data sets for use in the MARSS-package User Guide. Some are logged and some unlogged population counts. See the details below on each data set.

The data sets are matrices with year in the first column and counts in other columns. Since MARSS functions require time to be across columns, these data matrices must be transposed before passing into MARSS functions.

Usage

```
data(graywhales)
data(grouse)
data(prairiechicken)
data(wilddogs)
data(kestrel)
data(okanaganRedds)
data(rockfish)
data(redstart)
```

Format

The data are supplied as a matrix with years in the first column and counts in the second (and higher) columns.

Source

graywhales Gerber L. R., Master D. P. D. and Kareiva P. M. (1999) Gray whales and the value of monitoring data in implementing the U.S. Endangered Species Act. Conservation Biology, 13, 1215-1219.

grouse Hays D. W., Tirhi M. J. and Stinson D. W. (1998) Washington state status report for the sharptailed grouse. Washington Department Fish and Wildlife, Olympia, WA. 57 pp.

prairiechicken Peterson M. J. and Silvy N. J. (1996) Reproductive stages limiting productivity of the endangered Attwater's prairie chicken. Conservation Biology, 10, 1264-1276.

wilddogs Ginsberg, J. R., Mace, G. M. and Albon, S. (1995). Local extinction in a small and declining population: Wild Dogs in the Serengeti. Proc. R. Soc. Lond. B, 262, 221-228.

okanaganRedds A data set of Chinook salmon redd (egg nest) surveys. This data comes from the Okanagan River in Washington state, a major tributary of the Columbia River (headwaters in British Columbia). Unlogged.

rockfish Logged catch per unit effort data for Puget Sound total total rockfish (mix of species) from a series of different types of surveys.

kestrel Three time series of American kestrel logged abundance from adjacent Canadian provinces along a longitudinal gradient (British Columbia, Alberta, Saskatchewan). Data have been collected annually, corrected for changes in observer coverage and detectability, and logged.

redstart 1966 to 1995 counts for American Redstart from the North American Breeding Bird Survey (BBS record number 0214332808636; Peterjohn 1994) used in Dennis et al. (2006). Peterjohn, B.G. 1994. The North American Breeding Bird Survey. Birding 26, 386–398. and Dennis et al. 2006. Estimating density dependence, process noise, and observation error. Ecological Monographs 76:323-341.

Examples

```
str(graywhales)
str(grouse)
str(prairiechicken)
str(wilddogs)
str(kestrel)
str(okanaganRedds)
str(rockfish)
```

`predict` *predict and forecast MARSS MLE objects*

Description

See the following help files:

- `predict.marssMLE()` Predict and forecast.
- `forecast.marssMLE()` Forecast. Use `predict.marssMLE()` to call with argument `h`.
- `plot.marssPredict()` Plot a prediction or forecast.
- `autoplot.marssPredict()` Plot a prediction or forecast using ggplot2 package.
- `print.marssPredict()` Print prediction or forecast. If `h!=0`, i.e. forecast, only the forecast is printed but the `marssPredict` object (in `pred`) has all the historical time steps also.

predict.marssMLE *predict and forecast MARSS MLE objects*

Description

This function will return the modeled value of $\mathbf{y}_t$ or $\mathbf{x}_t$ conditioned on the data (either the data used to fit the model or data in newdata). For $\mathbf{y}_t$, this is $\mathbf{Z}_t\mathbf{x}_t^T + \mathbf{a}_t + \mathbf{D}_t\mathbf{d}_t$. For $\mathbf{x}_t$, this is $\mathbf{B}_t\mathbf{x}_{t-1}^T + \mathbf{u}_t + \mathbf{C}_t\mathbf{c}_t$. $\mathbf{x}_t^T$ is the smoothed state estimate at time t conditioned on all the data (either data used to fit the model or the optional data passed into newdata).

If you want the estimate of $\mathbf{x}_t$ conditioned on all the data (i.e. output from the Kalman filter or smoother), then use tsSmooth(). Note that the prediction of $\mathbf{x}_t$ conditioned on the data up to time t is not provided since that would require the estimate of $\mathbf{x}_t$ conditioned on data 1 to $t+1$, which is not output from the Kalman filter or smoother.

If h is passed in, predict(object) will return a forecast h steps past the end of the model data. predict(object) returns a marssPredict object which can be passed to plot() or ggplot2::autoplot()for automatic plotting of predictions and forecasts with intervals.

Usage

```
## S3 method for class 'marssMLE'
predict(object, n.ahead = 0,
     level = c(0.80, 0.95),
     type = c("ytt1", "ytT", "xtT", "ytt", "xtt1"),
     newdata = list(t=NULL, y=NULL, c=NULL, d=NULL),
     interval = c("none", "confidence", "prediction"),
     fun.kf = c("MARSSkfas", "MARSSkfss"),
     x0 = "reestimate", ...)
```

Arguments

object	A marssMLE object.
n.ahead	Number of steps ahead to forecast. If n.ahead !=0, then newdata is for the forecast, i.e. for the n.ahead time steps after the end of the model data. See details.
level	Level for the intervals if interval != "none".
type	ytT, ytt or ytt1: predictions for the observations based on the states estimate at time t conditioned on all the data, data up to t or data up to $t-1$. xtT or xtt1: predictions for the states at time t based on the states estimate at time $t-1$ conditioned on all the data or data up to $t-1$. The data are the data used to fit the model unless y is passed in in newdata.
newdata	An optional list with new y (data), c or d (covariates) to use for the predictions or forecasts. y, c or d must have the same structure (matrix dimensions) as used in the MARSS() call but the number of time steps can be different. t is used if there is ambuiquity as to which time steps the newdata refer to. See examples and details.

interval	If interval="confidence", then the standard error and confidence intervals of the predictions are returned. If interval="prediction", prediction intervals are returned. See fitted for a discussion of the intervals.
fun.kf	Only if you want to change the default Kalman filter. Can be ignored.
x0	If "reestimate" (the default), then the initial value for the states is re-estimated. If "use.model", then the initial values in the fitted model (object) are used. If you change the data, then this initial condition may not be appropriate. You can also pass in a new x0 to use. It must be a matrix that is the same dimensions as x0 in the model. x0 is ignored if h!=0 since in that case a forecast is being done. See example.
...	Other arguments. Not used.

Details

Forecasts n.ahead != 0

The type="xtT" forecast is the states forecast conditioned on all the data. If n.ahead !=0, then 'data' that is being conditioned on is the original data (model data) plus any data in newdata$y for the h forecast time steps. Note, typically forecasts would not have data, since they are forecasts, but predict.marssMLE() allows you to specify data for the forecast time steps if you need to. If the model includes covariates ($\mathbf{c}$ and/or $\mathbf{d}$ matrices passed into the model list in the MARSS() call), then c and/or d must be passed into newdata.

The type="ytT" forecast is the expected value of NEW data ($\mathbf{Y}$) conditioned on the data used for fitting. The data used for fitting is the same as for type="xtT" (above). The $\mathbf{y}$ forecast is Z xtT[,T+i] + A + D d[,T+i].

If the model has time-varying parameters, the value of the parameters at the last time step are used for the forecast.

Model predictions n.ahead == 0

If newdata is not passed in, then the model data ($\mathbf{y}$) and $\mathbf{c}$ and $\mathbf{d}$ (if part of model) are used for the predictions. fitted(object, type="ytT") is the internal function for model predictions in that case.

If newdata is passed in, then the predictions are computed using newdata but with the MARSS model estimated from the original data, essentially the Kalman filter/smoother is run using the estimated MARSS model but with data (and $\mathbf{c}$ and $\mathbf{d}$ if in the model) in newdata. y, c and d in the newdata list must all have the same number of columns (time-steps) and the length of t in newdata must be the same as the number of columns and must be sequential.

For type="ytT", the predictions are conceptually the same as predictions returned by predict.lm for a linear regression. The confidence interval is the interval for the expected value of NEW data. The prediction interval is the interval for NEW data. Prediction intervals will always be wider (or equal if R=0) to confidence intervals. The difference is that the uncertainty in predict.lm comes from parameter uncertainty and the data error while in predict.marssMLE, the uncertainty is from $\mathbf{x}$ uncertainty and data error. Parameter uncertainty does not enter the interval calculations; parameters are treated as known at their point estimates. This is not specific to the MARSS package. This is how prediction and confidence intervals are presented for MARSS models in the literature, i.e. no parameter uncertainty.

t in newdata: If the model has time-varying parameters, t in newdata removes any ambiguity as to which parameter values (time steps) will be used for prediction. In this case, t specifies which time values of the parameters you want to use. If you leave off t, then it is assumed that t starts at the first time step in the data used to fit the original model. If the model is time-constant, t is used to set the time step values (used for plotting, etc.).

The model has c and/or d: c and/or d must be included in newdata. If y (new data) is not in newdata, it is assumed to be absent (all NA). That is, the default behavior if y is absent but c and/or d is present is y="none". If you want to use the original data used to fit the model, then pass in y="model" in newdata. Pass in t in newdata if it is ambiguous which time steps of the model data to use.

The model has time-varying parameters: You have to pass in t in newdata to specify what parameter values to use. If any $t > T$ (T equals the last time step in the model data), then it is assumed that you want to use the parameter values at the last time step of the original time series for values beyond the last time step. See examples.

y, c and d in newdata have more time steps than the original data: If the model has time-varying parameters, you will need to pass in t. If the model is time-constant, then t is assumed to start at the first time step in the original data but you can pass in t to change that. It will not change the prediction, but will change the t column in the output.

x0 estimation If you are passing in y in newdata, then it is likely that you will need to re-estimate the x initial condition. The default behavior of predict.marssMLE. Use x0 = "use.model" to use the initial values in the estimated model (object).

Value

A list with the following components:

method	The method used for fitting, e.g. MARSS kem.
model	The marssMLE object passed into predict().
newdata	The newdata list if passed in.
level	The confidence or prediction intervals level.
pred	A data frame the predictions or forecasts along with the intervals.
type	The type passed in.
t	The time steps in the pred data frame.
n.ahead and h	The number of forecast time steps.
x0	The x0 used for the predictions.
tinitx	The tinitx used.

The pred data frame has the following columns:

.rownames	Names of the data or states.
t	Time step.
y	The data if type is "ytT", "ytt" or "ytt1".
xtT	The estimate of $\mathbf{x}_t$ conditioned on all the data if type="xtT". From tsSmooth().
xtt	The estimate of $\mathbf{x}_t$ conditioned on the data 1 to t if type="xtt1". From tsSmooth().

estimate	Model predicted values of observations ($\mathbf{y}$) or the states ($\mathbf{x}$). See details.

If intervals are returned, the following are added to the data frame:

se	Standard errors of the predictions.
Lo ...	Lower confidence level at alpha = 1-level. The interval is approximated using qnorm(alpha/2)*se + prediction.
Hi ...	Upper confidence level. The interval is approximated using qnorm(1-alpha/2)*se + prediction.

Author(s)

Eli Holmes, NOAA, Seattle, USA.

See Also

plot.marssPredict(), fitted.marssMLE()

Examples

```
dat <- t(harborSealWA)
dat <- dat[2:4,] #remove the year row
fit <- MARSS(dat, model=list(R="diagonal and equal"))

# 2 steps ahead forecast
fr <- predict(fit, type="ytT", n.ahead=2)
plot(fr)

# use model data with the estimated initial values (at t=0) for
# initial values at t=9
# This would be a somewhat strange thing to do and the value at t=10 will look wrong.
fr <- predict(fit,  newdata=list(t=10:20, y=dat[,10:20]), x0 = "use.model")
plot(fr)

# pass in new data and give it new t; initial conditions will be estimated
fr <- predict(fit,  newdata=list(t=23:33, y=matrix(10,3,11)))
plot(fr, ylim=c(8,12))


# Covariate example
fulldat <- lakeWAplanktonTrans
years <- fulldat[,"Year"]>=1965 & fulldat[,"Year"]<1975
dat <- t(fulldat[years,c("Greens", "Bluegreens")])
dat <- zscore(dat)
covariates <- rbind(
  Temp = fulldat[years, "Temp"],
  TP = fulldat[years, "TP"])
covariates <- zscore(covariates)
A <- U <- "zero"
B <- Z <- "identity"
R <- diag(0.16,2)
Q <- "equalvarcov"
```

```
C <- "unconstrained"
model.list <- list(B=B,U=U,Q=Q,Z=Z,A=A,R=R,C=C,c=covariates)
fit <- MARSS(dat, model=model.list)

# Use a new c (covariate) but no data.
fr <- predict(fit,  newdata=list(c=matrix(5,2,10)), x0="use.model")
plot(fr)

# Use first 10 time steps of model data
plot(predict(fit, newdata=list(y=dat[,1:10], c=matrix(5,2,10))))

# Use all model data but new covariates
# Why does it look so awful? Because this is a one-step ahead
# prediction and there is no info on what the c will be at t
plot(predict(fit,  newdata=list(y=dat, c=matrix(5,2,120))))

# Use all model data but new covariates with ytT type
# this looks better because is uses all the c data to estimate (so knows what c is at t and beyond)
plot(predict(fit,  newdata=list(y=dat, c=matrix(5,2,120)), type="ytT"))

# Use no data; cannot estimate initial conditions without data
# so x0 must be "use.model"
fr <- predict(fit,  newdata=list(c=matrix(5,2,22)), x0="use.model")
plot(fr)

# forecast with covariates
# n.ahead and the number column in your covariates in newdata must match
plot(predict(fit,  newdata=list(c=matrix(5,2,10)), n.ahead=10))

# forecast with covariates and only show last 10 steps of original data
plot(predict(fit,  newdata=list(c=matrix(5,2,10)), n.ahead=10), include=10)
```

`print.marssMLE` *Printing functions for MARSS MLE objects*

Description

`MARSS()` outputs `marssMLE` objects. `print(MLEobj)`, where `MLEobj` is a `marssMLE` object, will print out information on the fit. However, `print` can be used to print a variety of information (residuals, smoothed states, imputed missing values, etc) from a `marssMLE` object using the `what` argument in the print call.

Usage

```
## S3 method for class 'marssMLE'
print(x, digits = max(3, getOption("digits")-4), ...,
    what = "fit", form = NULL, silent = FALSE)
```

Arguments

x	A marssMLE object.
digits	Number of digits for printing.
...	Other arguments for print.
what	What to print. Default is "fit". If you input what as a vector, print returns a list. See examples.

"model" The model parameters with names for the estimated parameters. The output is customized by the form of the model that was fit. This info is in `attr(x$model, "form")` .

"par" A list of only the estimated values in each matrix. Each model matrix has it's own list element. Standard function: `coef(x)`

"start" or "inits" The values that the optimization algorithm was started at. Note, `x$start` shows this in form="marss" while `print` shows it in whatever form is in `attr(x$model, "form")` .

"paramvector" A vector of all the estimated values in each matrix. Standard function: `coef(x, type="vector")`. See `coef()`.

"par.se","par.bias","par.lowCIs","par.upCIs" A vector the estimated parameter standard errors, parameter bias, lower and upper confidence intervals. Standard function: `MARSSparamCIs(x)` See `MARSSparamCIs()`.

"xtT" or "states" The estimated states conditioned on all the data. `x$states`

"data" The data. This is in `x$model$data`

"logLik" The log-likelihood. Standard function: `x$logLik`. See `MARSSkf()` for a discussion of the computation of the log-likelihood for MARSS models.

"ytT" The expected value of the data conditioned on all the data. Returns the data if present and the expected value if missing. This is in `x$ytT` (ytT is analogous to xtT).

"states.se" The state standard errors. `x$states.se`

"states.cis" Approximate confidence intervals for the states. See `MARSSparamCIs()`.

"model.residuals" The one-step ahead model residuals or innovations. $\mathbf{y}_t - \mathrm{E}[\mathbf{Y}_t|\mathbf{y}_1^{t-1}]$, aka actual data at time t minus the expected value of the data conditioned on the data from $t = 1$ to $t - 1$. Standard function: `residuals(x, type="tt1")` See `MARSSresiduals()` for a discussion of residuals in the context of MARSS models.

"state.residuals" The smoothed state residuals. $\mathbf{x}_t^T - \mathrm{E}[\mathbf{X}_t|\mathbf{x}_{t-1}^T]$, aka the expected value of the states at time t conditioned on all the data minus the expected value of the states at time t conditioned on $\mathbf{x}_{t-1}^T]$. Standard function: `residuals(x, type="tT")` See `MARSSresiduals()`.

parameter name Returns the parameter matrix for that parameter with fixed values at their fixed values and the estimated values at their estimated values. Standard function: `coef(x, type="matrix")$elem`

"kfs" The Kalman filter and smoother output. See `MARSSkf()` for a description of the output. The full kf output is not normally attached to the output from a `MARSS()` call. This will run the filter/smoother if needed and return the list invisibly. So assign the output as `foo=print(x,what="kfs")`

"Ey" The expectations involving y conditioned on all the data. See `MARSShatyt()` for a discussion of these expectations. This output is not normally attached to the output from a `MARSS()` call–except `ytT` which is the predicted value of any missing y. The list is returned invisibly so assign the output as `foo=print(x,what="Ey")`.

`form` By default, print uses the model form specified in the call to `MARSS()`. This information is in `attr(marssMLE$model, "form")` , however you can specify a different form. `form="marss"` should always work since this is the model form in which the model objects are stored (in `marssMLE$marss`).

`silent` If TRUE, do not print just return the object. If print call is assigned, nothing will be printed. See examples. If `what="fit"`, there is always output printed.

Value

A print out of information. If you assign the print call to a value, then you can reference the output. See the examples.

Author(s)

Eli Holmes, NOAA, Seattle, USA.

Examples

```
dat <- t(harborSeal)
dat <- dat[c(2,11),]
MLEobj <- MARSS(dat)

print(MLEobj)

print(MLEobj, what="model")

print(MLEobj,what="par")

#silent doesn't mean silent unless the print output is assigned
print(MLEobj, what="paramvector", silent=TRUE)
tmp <- print(MLEobj, what="paramvector", silent=TRUE)
#silent means some info on what you are printing is shown whether
#or not the print output is assigned
print(MLEobj, what="paramvector", silent=FALSE)
tmp <- print(MLEobj, what="paramvector", silent=FALSE)

cis <- print(MLEobj, what="states.cis")
cis$up95CI

vars <- print(MLEobj, what=c("R","Q"))
```

print.marssMODEL *Printing marssMODEL Objects*

Description

print(MODELobj), where MODELobj is a marssMODEL object, will print out information on the model in short form (e.g. 'diagonal and equal').

summary(marssMODEL), where marssMODEL is a marssMODEL object, will print out detailed information on each parameter matrix showing where the estimated values (and their names) occur.

Usage

```
## S3 method for class 'marssMODEL'
print(x, ...)
## S3 method for class 'marssMODEL'
summary(object, ..., silent = FALSE)
```

Arguments

x	A marssMODEL object.
object	A marssMODEL object.
...	Other arguments .
silent	TRUE/FALSE Whether to print output.

Value

print(marssMODEL) prints out of the structure of each parameter matrix in 'English' (e.g. 'diagonal and unequal') and returns invisibly the list. If you assign the print call to a value, then you can reference the output.

summary(marssMODEL) prints out of the structure of each parameter matrix in as list matrices showing where each estimated value occurs in each matrix and returns invisibly the list. The output can be verbose, especially if parameter matrices are time-varying. Pass in silent=TRUE and assign output (a list with each parameter matrix) to a variable. Then specific parameters can be looked at.

Author(s)

Eli Holmes, NOAA, Seattle, USA.

Examples

```
dat <- t(harborSeal)
dat <- dat[c(2, 11), ]
fit <- MARSS(dat)

print(fit$model)
# this is identical to
print(fit, what = "model")
```

`print.marssPredict` *Printing function for MARSS Predict objects*

Description

`MARSS()` outputs `marssMLE` objects. `predict(object)`, where object is `marssMLE` object, will return the predictions of $\mathbf{y}_t$ or the smoothed value of $\mathbf{x}_t$ for `h` steps past the end of the model data. `predict(object)` returns a `marssPredict` object which can be passed to `print.marssPredict()` for automatic printing.

Usage

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

Arguments

`x`	A `marssPredict` object.
`...`	Other arguments for print. Not used.

Value

A print out of the predictions as a data frame.

Author(s)

Eli Holmes, NOAA, Seattle, USA.

Examples

```
dat <- t(harborSealWA)
dat <- dat[2:4,] #remove the year row
fit <- MARSS(dat, model=list(R="diagonal and equal"))

# 2 steps ahead forecast
predict(fit, type="ytT", n.ahead=2)

# smoothed x estimates with intervals
predict(fit, type="xtT")
```

`residuals.marssMLE` *Model and state fitted values, residuals, and residual sigma*

Description

`residuals.marssMLE` returns a data frame with fitted values, residuals, residual standard deviation (sigma), and standardized residuals. A residual is the difference between the "value" of the model ($\mathbf{y}$) or state ($\mathbf{x}$) and the fitted value. At time t (in the returned data frame), the model residuals are for time t. For the the state residuals, the residual is for the transition from t to $t+1$ following the convention in Harvey, Koopman and Penzer (1998). For the the state innovation residuals, this means that `state.residual[,t]` is for the transition from t to $t+1$ and is conditioned on data 1 to t while `model.residual[,t]` is is conditioned on data 1 to $t-1$. State innovation residuals are not normally used while state smoothation residuals are used in trend outlier analysis. If warnings are reported, use `attr(residuals(fit), "msg")` to retrieve the messages.

Because the state residuals is for the transition from t to $t+1$, this means that the state residual `.resids[t]` is `value[t-1]` minus `.fitted[t-1]` in the outputted data frame.

Usage

```
## S3 method for class 'marssMLE'
residuals(object, ...,
          type=c("tt1", "tT", "tt"),
          standardization=c("Cholesky", "marginal", "Block.Cholesky"),
          form=attr(object[["model"]], "form")[1],
          clean=TRUE)
```

Arguments

`object`	a `marssMLE` object
`type`	`tt1` means innovations residuals. The fitted values are computed conditioned on the data up to $t-1$. See `fitted()` with `type="ytt1"` or `type="xtt1"`. `tT` means smoothation residuals. the fitted values are computed conditioned on all the data. See `fitted()` with `type="ytT"` or `type="xtT"`. `tt` means contemporaneous residuals. The fitted values are computed conditioned on the data up to t. In MARSS functions, estimates at time t conditioned on data 1 to T are denoted `tT`, conditioned on the data from $t=1$ to $t-1$ are denoted `tt1` and conditioned on data 1 to t are `tt`.
`standardization`	"Cholesky" means it is standardized by the lower triangle of the Cholesky transformation of the full variance-covariance matrix of the model and state residuals. "marginal" means that the residual is standardized by its standard deviation, i.e. the square root of the value on the diagonal of the variance-covariance matrix of the model and state residuals. "Block.Cholesky" means the model or state residuals are standardized by the lower triangle of the Cholesky transformation of only their variance-covariance matrix (not the joint model and state variance-covariance matrix).

`form`	For developers. Can be ignored. If you want the function to use a different function than `residuals_form`. This might be useful if you manually specified a DFA model and want to use `residuals_dfa` for rotating.
`clean`	Can be ignored. For `type="tt1"`, state residuals are not used for residuals analysis and for `type="tt"`, they don't exist (all NA). They are used only for smoothation residuals, `type="tT"`. For `type="tt1"` and `type="tt"`, the data frame is cleaned by removing `name=="state"` when `clean=TRUE`.
`...`	Not used.

Details

See `MARSSresiduals` for a discussion of the residuals calculations for MARSS models.

value and **.fitted**

See the discussion below on the meaning of these for $\mathbf{y}$ associated residuals (model residuals) or $\mathbf{x}$ associated residuals (state residuals).

model residuals

The model residuals are in the data frame with `name=="model"`.

The model residuals are the familiar type of residuals, they are the difference between the data at time t and the predicted value at time t, labeled `.fitted` in the data frame. For the model residuals, the "value"" is the data (or NA if data are missing). If `type="tT"`, the predicted value is the expected value of $\mathbf{Y}$ conditioned on all the data, i.e. is computed using the smoothed estimate of $\mathbf{x}$ at time t (`xtT`). If `type="tt1"`, the predicted value is the expected value of $\mathbf{Y}$ conditioned on the data up to time $t-1$, i.e. is computed using the estimate of $\mathbf{x}$ at time t conditioned on the data up to time $t-1$ (`xtt1`). These are known as the one-step-ahead predictions and the residuals are known as the innovations.

The standard errors help visualize how well the model fits to the data. See `fitted` for a discussion of the calculation of the model predictions for the observations. The standardized smoothation residuals can be used for outlier detection. See the references in `MARSSresiduals` and the chapter on shock detection in the MARSS User Guide.

state residuals

The state residuals are in the data frame with `name=="state"`.

If you want the expected value of the states and an estimate of their standard errors (for confidence intervals), then `residuals()` is not what you want to use. You want to use `tsSmooth(..., type="xtT")` to return the smoothed estimate of the state or you can find the states in the `states` element of the `marssMLE` object returned by a `MARSS()` call. For the one-step-ahead state estimates, use `tsSmooth(..., type="xtt1")`.

The state residuals are only for state-space models. At time t, the state residuals are the difference between the state estimate at time $t+1$ and the predicted value of the state at time $t+1$ given the estimate of the state at time t. For smoothation state residuals, this is

$$\hat{\mathbf{w}}_{t+1} = \mathbf{x}_{t+1}^{T} - \mathbf{B}\mathbf{x}_{t}^{T} - \mathbf{u} - \mathbf{C}\mathbf{c}_{t+1}$$

For "tt1" state residuals, this is

$$\hat{\mathbf{w}}_{t+1} = \mathbf{x}_{t+1}^{t+1} - \mathbf{B}\mathbf{x}_{t}^{t} - \mathbf{u} - \mathbf{C}\mathbf{c}_{t+1}$$

. Note the t indexing is offset. The state residual at time t is the estimate at time t+1 minus the fitted value at t+1.

Smoothation state residuals are used for outlier detection or shock detection in the state process. See MARSSresiduals and read the references cited. Note that the state residual at time T (the last time step) is NA since this would be the transition from T to $T+1$ (past the end of the data).

Note, because the state residuals are for the transition from t to $t+1$, this means that in the outputted data frame, the state residual `.resids[t]` is `value[t-1]` minus `.fitted[t-1]`.

Value

A data frame with the following columns:

`type`	tT, tt1 or tt
`.rownames`	The names of the observation rows or the state rows.
`name`	model or state
`t`	time step
`value`	The data value if `name` equals "model" or the x estimate if `name` equals "state" at time t. See details.
`.fitted`	Model predicted values of observations or states at time t. See details.
`.resids`	Model or states residuals. See details.
`.sigma`	The standard error of the model or state residuals. Intervals for the residuals can be constructed from `.sigma` using `qnorm(alpha/2)*.sigma + .fitted`.
`.std.resids`	Standardized residuals. See MARSSresiduals for a discussion of residual standardization.

References

Holmes, E. E. 2014. Computation of standardized residuals for (MARSS) models. Technical Report. arXiv:1411.0045.

See also the discussion and references in MARSSresiduals.tT, MARSSresiduals.tt1 and MARSSresiduals.tt.

Examples

```
dat <- t(harborSeal)
dat <- dat[c(2, 11, 12), ]
fit <- MARSS(dat, model = list(Z = factor(c("WA", "OR", "OR"))))

library(ggplot2)
theme_set(theme_bw())

## Not run:
# Show a series of standard residuals diagnostic plots for state-space models
autoplot(fit, plot.type="residuals")

## End(Not run)

d <- residuals(fit, type="tt1")
```

```
## Not run:
# Make a series of diagnostic plots from a residuals object
autoplot(d)

## End(Not run)

# Manually make a plot of the model residuals (innovations) with intervals
d$.conf.low <- d$.fitted+qnorm(0.05/2)*d$.sigma
d$.conf.up <- d$.fitted-qnorm(0.05/2)*d$.sigma
ggplot(data = d) +
  geom_line(aes(t, .fitted)) +
  geom_point(aes(t, value), na.rm=TRUE) +
  geom_ribbon(aes(x = t, ymin = .conf.low, ymax = .conf.up), linetype = 2, alpha = 0.1) +
  ggtitle("Model residuals (innovations)") +
  xlab("Time Step") + ylab("Count") +
  facet_grid(~.rownames)

# NOTE state residuals are for t to t+1 while the value and fitted columns
# are for t. So (value-fitted)[t] matches .resids[t+1] NOT .resids[t]
# This is only for state residuals. For model residuals, the time-indexing matches.
d <- residuals(fit, type="tT")
dsub <- subset(d, name=="state")
# note t in col 1 matches t+1 in col 2
head(cbind(.resids=dsub$.resids, valminusfitted=dsub$value-dsub$.fitted))

# Make a plot of the smoothation residuals
ggplot(data = d) +
  geom_point(aes(t, value-.fitted), na.rm=TRUE) +
  facet_grid(~.rownames+name) +
  ggtitle("Smoothation residuals (state and model)") +
  xlab("Time Step") + ylab("Count")

# Make a plot of xtT versus prediction of xt from xtT[t-1]
# This is NOT the estimate of the smoothed states with CIs. Use tsSmooth() for that.
ggplot(data = subset(d, name=="state")) +
  geom_point(aes(t, value), na.rm=TRUE) +
  geom_line(aes(x = t, .fitted), color="blue") +
  facet_grid(~.rownames) +
  xlab("Time Step") + ylab("Count") +
  ggtitle("xtT (points) and prediction (line)")

# Make a plot of y versus prediction of yt from xtT[t]
# Why doesn't the OR line go through the points?
# Because there is only one OR state line and it needs to go through
# both sets of OR data.
ggplot(data = subset(d, name=="model")) +
  geom_point(aes(t, value), na.rm=TRUE) +
  geom_line(aes(x = t, .fitted), color="blue") +
  facet_grid(~.rownames) +
  xlab("Time Step") + ylab("Count") +
  ggtitle("data (points) and prediction (line)")
```

SalmonSurvCUI *Salmon Survival Indices*

Description

Example data set for use in MARSS vignettes for the DLM chapter in the MARSS-package User Guide. This is a 42-year time-series of the logit of juvenile salmon survival along with an index of April coastal upwelling. See the source for details.

Usage

```
data(SalmonSurvCUI)
```

Format

The data are provided as a matrix with time running down the rows. Column 1 is year, column 2 is the logit of the proportion of juveniles that survive to adulthood, column 3 is an index of the April coastal upwelling index.

Source

Scheuerell, Mark D., and John G. Williams. "Forecasting climate-induced changes in the survival of Snake River spring/summer Chinook salmon (Oncorhynchus tshawytscha)." Fisheries Oceanography 14.6 (2005): 448-457.

Examples

```
str(SalmonSurvCUI)
```

summary.marssMLE *Summary methods for marssMLE objects*

Description

A brief summary of the fit: number of state and observation time series and the estimates. See also glance() and tidy() for other summary like output.

Usage

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

Arguments

object	A marssMLE object.
digits	Number of digits for printing.
...	Not used.

Value

Returns ‘object‘ invisibly.

Author(s)

Eli Holmes, NOAA, Seattle, USA.

See Also

glance(), tidy()

Examples

```
dat <- t(harborSeal)
dat <- dat[c(2,11),]
fit <- MARSS(dat)

summary(fit)
glance(fit)
tidy(fit)
```

---

tidy.marssMLE *Return estimated parameters with summary information*

Description

tidy.marssMLE is the method for the tidy generic. It returns the parameter estimates and their confidence intervals.

Usage

```
## S3 method for class 'marssMLE'
tidy(x, conf.int = TRUE, conf.level = 0.95, ...)
```

Arguments

x	a marssMLE object
conf.int	Whether to compute confidence and prediction intervals on the estimates.
conf.level	Confidence level. alpha=1-conf.level
...	Optional arguments. If conf.int=TRUE, then arguments to specify how CIs are computed can be passed in. See details and MARSSparamCIs.

Details

`tidy.marssMLE()` assembles information available via the `print()` and `coef()` functions into a data frame that summarizes the estimates. If `conf.int=TRUE`, `MARSSparamCIs()` will be run to add confidence intervals to the model object if these are not already added. The default CIs are calculated using a analytically computed Hessian matrix. This can be changed by passing in optional arguments for `MARSSparamCIs()`.

Value

A data frame with estimates, sample standard errors, and confidence intervals.

Examples

```
dat <- t(harborSeal)
dat <- dat[c(2, 11, 12), ]
fit <- MARSS(dat)

# A data frame of the estimated parameters
tidy(fit)
```

---

toLatex.marssMODEL *Create a LaTeX Version of the Model*

Description

Creates LaTex and a PDF (if LaTeX compiler available) using the tools in the Hmisc package. The files are saved in the working directory.

Usage

```
## S3 method for class 'marssMODEL'
toLatex(object, ..., file = NULL, digits = 2, greek = TRUE, orientation = "landscape",
math.sty = "amsmath", output = c("pdf", "tex", "rawtex"), replace = TRUE, simplify = TRUE)
## S3 method for class 'marssMLE'
toLatex(object, ..., file = NULL, digits = 2, greek = TRUE, orientation = "landscape",
math.sty = "amsmath", output = c("pdf", "tex", "rawtex"), replace = TRUE, simplify = TRUE)
```

Arguments

`object`	A `marssMODEL` or `marssMLE` object.
`...`	Other arguments. Not used.
`file`	Name of file to save to. Optional.
`digits`	Number of digits to display for numerical values (if real).
`greek`	Use greek symbols.
`orientation`	Orientation to use. landscape or portrait.

math.sty	LaTeX math styling to use.
output	pdf, tex or rawtex. If blank, both are output.
replace	Replace existing file if present.
simplify	If TRUE, then if $\mathbf{B}$ or $\mathbf{Z}$ are identity, they do not appear. Any zero-ed out elements also do not appear.

Value

A LaTeX and or PDF file of the model.

Author(s)

Eli Holmes, NOAA, Seattle, USA.

Examples

```
# Example with linear constraints
dat <- t(harborSeal)
dat <- dat[c(2:4), ]
Z1 <- matrix(list("1*z1+-1*z2",0,"z2","2*z1","z1",0),3,2)
A1 <- matrix(list("a1",0,0),3,1)
MLEobj <- MARSS(dat, model=list(Z=Z1, A=A1, Q=diag(0.01,2)))
 ## Not run:
 toLatex(MLEobj)
 toLatex(MLEobj$model)

## End(Not run)
```

---

tsSmooth.marssMLE *Smoothed and filtered x and y time series*

Description

tsSmooth.marssMLE returns the estimated state and observations conditioned on the data. This function will return either the smoothed values (conditioned on all the data) or the filtered values (conditioned on data 1 to t or $t-1$). This is output from the Kalman filter and smoother MARSSkf() for the $\mathbf{x}$ and from the corresponding function MARSShatyt() for the $\mathbf{y}$.

These are the expected value of the full right side of the MARSS equations with the error terms (expected value of $\mathbf{X}_t$ and $\mathbf{Y}_t$). Conditioning on data $t = 1$ to $t-1$ (one-step ahead), t (contemporaneous), or T (smoothed) is provided. This is in contrast to fitted() which returns the expected value of the right side without the error term, aka model predictions.

In the state-space literature, the $\mathbf{y}$ "estimates" would normally refer to the expected value of the right-side of the $\mathbf{y}$ equation without the error term (i.e. the expected value of $\mathbf{Z}\mathbf{X}_t + \mathbf{a} + \mathbf{D}\mathbf{d}_t$). That is provided in fitted(). tsSmooth.marssMLE() provides the expected value with the error terms conditioned on the data from 1 to $t-1$, t, or T. These estimates are used to estimate missing

values in the data. If $\mathbf{y}$ is multivariate, some y are missing at time t and some not, and $\mathbf{R}$ is non-diagonal, then the expected value of $\mathbf{Y}_t$ from the right-side of the $\mathbf{y}$ without the error terms would be incorrect because it would not take into account the information in the observed data at time t on the missing data at time t (except as it influences $\mathrm{E}[\mathbf{x}_t]$).

Note, if there are no missing values, the expected value of $\mathbf{Y}_t$ (with error terms) conditioned on the data from 1 to t or T is simply $\mathbf{y}_t$. The expectation is only useful when there are missing values for which an estimate is needed. The expectation of the $\mathbf{Y}$ with the error terms is used in the EM algorithm for the general missing values case and the base function is `MARSShatyt()`.

Usage

```
## S3 method for class 'marssMLE'
tsSmooth(object,
    type = c("xtT", "xtt", "xtt1", "ytT", "ytt", "ytt1"),
    interval = c("none", "confidence", "prediction"),
    level = 0.95, fun.kf = c("MARSSkfas", "MARSSkfss"), ...)
```

Arguments

`object`	A `marssMLE` object.
`type`	Type of estimates to return. Smoothed states (`xtT`), one-step-ahead states (`xtt1`), contemporaneous states (`xtt`), the model `ytT` (`Z xtT + A + D d(t)`), the model `ytt` (`Z xtt + A + D d(t)`), the model `ytt1` (`Z xtt1 + A + D d(t)`), the expected value of $\mathbf{Y}_t$ conditioned on data 1 to $t-1$ (`ytt1`), the expected value of $\mathbf{Y}_t$ conditioned on data 1 to t (`ytt`), or the expected value of $\mathbf{Y}_t$ conditioned on data 1 to T (`ytT`). See details.
`interval`	If `interval="confidence"`, then the standard error and confidence intervals are returned. There are no prediction intervals for estimated states and observations except for `ytT` (which is a unusual case.) If you are looking for prediction intervals, then you want `fitted()` or `predict()`.
`level`	Confidence level. alpha=1-level
`fun.kf`	By default, `tsSmooth()` will use the Kalman filter/smoother function in `object$fun.kf` (either `MARSSkfas()` or `MARSSkfss()`). You can pass in `fun.kf` to force a particular Kalman filter/smoother function to be used.
`...`	Optional arguments. If form="dfa", `rotate=TRUE` can be passed in to rotate the trends (only trends not the $\mathbf{Z}$ matrix).

Details

Below, X and Y refers to the random variable and x and y refer to a specific realization from this random variable.

state estimates (x)

For `type="xtT"`, `tsSmooth.marssMLE` returns the confidence intervals of the state at time t conditioned on the data from 1 to T using the estimated model parameters as true values. These are the standard intervals that are shown for the estimated states in state-space models. For example see, Shumway and Stoffer (2000), edition 4, Figure 6.4. As such, this is probably what you

are looking for if you want to put intervals on the estimated states (the $\mathbf{x}$). However, these intervals do not include parameter uncertainty. If you want state residuals (for residuals analysis), use `MARSSresiduals()` or `residuals()`.

Quantiles The state $\mathbf{X}_t$ in a MARSS model has a conditional multivariate normal distribution, that can be computed from the model parameters and data. In Holmes (2012, Equation 11) notation, its expected value conditioned on all the observed data and the model parameters Θ is denoted $\tilde{\mathbf{x}}_t$ or equivalently $\mathbf{x}_t^T$ (where the T superscript is not a power but the upper extent of the time conditioning). In `MARSSkf`, this is `xtT[,t]`. The variance of $\mathbf{X}_t$ conditioned on all the observed data and Θ is $\tilde{\mathbf{V}}_t$ (`VtT[,,t]`). Note that `VtT[,,t] != B VtT[,,t-1] t(B) + Q`, which you might think by looking at the MARSS equations. That is because the variance of $\mathbf{W}_t$ conditioned on the data (past, current and FUTURE) is not equal to $\mathbf{Q}$ ($\mathbf{Q}$ is the unconditional variance).

$\mathbf{x}_t^T$ (`xtT[,t]`) is an estimate of $\mathbf{x}_t$ and the standard error of that estimate is given by $\mathbf{V}_t^T$ (`VtT[,,t]`). Let `se.xt` denote the sqrt of the diagonal of `VtT`. The equation for the $\alpha/2$ confidence interval is `(qnorm(alpha/2)*se.xt + xtT)`. $\mathbf{x}_t$ is multivariate and this interval is for one of the x's in isolation. You could compute the m-dimensional confidence region for the multivariate $\mathbf{x}_t$, also, but `tsSmooth.marssMLE` returns the univariate confidence intervals.

The variance `VtT` gives information on the uncertainty of the true location of $\mathbf{x}_t$ conditioned on the observed data. As more data are collected (or added to the analysis), this variance will shrink since the data, especially data at time t, increases the information about the locations of $\mathbf{x}_t$. This does not affect the estimation of the model parameters, those are fixed (we are assuming), but rather our information about the states at time t.

If you have a DFA model (form='dfa'), you can pass in `rotate=TRUE` to return the rotated trends. If you want the rotated loadings, you will need to compute those yourself:

```
dfa <- MARSS(t(harborSealWA[,-1]), model=list(m=2), form="dfa")
Z.est <- coef(dfa, type="matrix")$Z
H.inv <- varimax(coef(dfa, type="matrix")$Z)$rotmat
Z.rot <- Z.est %*% H.inv
```

For `type="xtt"` and `type=="xtt1"`, the calculations and interpretations of the intervals are the same but the conditioning is for data $t = 1$ to t or $t = 1$ to $t - 1$.

observation estimates (y)

For `type="ytT"`, this returns the expected value and standard error of $\mathbf{Y}_t$ (left-hand side of the $\mathbf{y}$ equation) conditioned on $\mathbf{Y}_t = y_t$. If you have no missing data, this just returns your data set. But you have missing data, this what you want in order to estimate the values of missing data in your data set. The expected value of $\mathbf{Y}_t|\mathbf{Y} = \mathbf{y}(1:T)$ is in `ytT` in `MARSShatyt()` output and the variance is `OtT-tcrossprod(ytT)` from the `MARSShatyt()` output.

The intervals reported by `tsSmooth.marssMLE` for the missing values take into account all the information in the data, specifically the correlation with other data at time t if $\mathbf{R}$ is not diagonal. This is what you want to use for interpolating missing data. You do not want to use `predict.marssMLE()` as those predictions are for entirely new data sets and thus will ignore relevant information if $\mathbf{y}_t$ is multivariate, not all $\mathbf{y}_t$ are missing, and the $\mathbf{R}$ matrix is not diagonal.

The standard error and confidence interval for the expected value of the missing data along with the standard deviation and prediction interval for the missing data are reported. The former uses the variance of $\mathrm{E}[\mathbf{Y}_t]$ conditioned on the data while the latter uses variance of $\mathbf{Y}_t$ conditioned on the data. `MARSShatyt()` returns these variances and expected values. See Holmes (2012) for a

discussion of the derivation of expectation and variance of $\mathbf{Y}_t$ conditioned on the observed data (in the section 'Computing the expectations in the update equations').

For `type="ytt"`, only the estimates are provided. `MARSShatyt()` does not return the necessary variances matrices for the standard errors for this cases.

Value

A data frame with the following columns is returned. Values computed from the model are prefaced with ".".

If `interval="none"`, the following are returned:

`.rownames`	Names of the data or states.
`t`	Time step.
`y`	The data if `type` is `"ytT"`, `"ytt"` or `"ytt1"`.
`.estimate`	The estimated values. See details.

If `interval = "confidence"`, the following are also returned:

`.se`	Standard errors of the estimates.
`.conf.low`	Lower confidence level at `alpha = 1-level`. The interval is approximated using qnorm(alpha/2)*se + estimate
`.conf.up`	Upper confidence level. The interval is approximated using qnorm(1-alpha/2)*se + estimate

If `interval = "prediction"`, the following are also returned:

`.sd`	Standard deviation of new $\mathbf{y}_t$ values.
`.lwr`	Lower range at `alpha = 1-level`. The interval is approximated using qnorm(alpha/2)*sd + estimate
`.upr`	Upper range at `level`. The interval is approximated using qnorm(1-alpha/2)*sd + estimate

References

R. H. Shumway and D. S. Stoffer (2000). Time series analysis and its applications. Edition 4. Springer-Verlag, New York.

Holmes, E. E. (2012). Derivation of the EM algorithm for constrained and unconstrained multivariate autoregressive state-space (MARSS) models. Technical Report. arXiv:1302.3919 [stat.ME]

Examples

```
dat <- t(harborSeal)
dat <- dat[c(2, 11, 12), ]
fit <- MARSS(dat)

# Make a plot of the estimated states
library(ggplot2)
d <- tsSmooth(fit, type = "xtT", interval="confidence")
```

```
ggplot(data = d) +
  geom_line(aes(t, .estimate)) +
  geom_ribbon(aes(x = t, ymin = .conf.low, ymax = .conf.up), linetype = 2, alpha = 0.3) +
  facet_grid(~.rownames) +
  xlab("Time Step") + ylab("State estimate")

# Make a plot of the estimates for the missing values
library(ggplot2)
d <- tsSmooth(fit, type = "ytT", interval="confidence")
d2 <- tsSmooth(fit, type = "ytT", interval="prediction")
d$.lwr <- d2$.lwr
d$.upr <- d2$.upr
ggplot(data = d) +
  geom_point(aes(t, .estimate)) +
  geom_line(aes(t, .estimate)) +
  geom_point(aes(t, y), color = "blue", na.rm=TRUE) +
  geom_ribbon(aes(x = t, ymin = .conf.low, ymax = .conf.up), alpha = 0.3) +
  geom_line(aes(t, .lwr), linetype = 2) +
  geom_line(aes(t, .upr), linetype = 2) +
  facet_grid(~.rownames) +
  xlab("Time Step") + ylab("Count") +
  ggtitle("Blue=data, Black=estimate, grey=CI, dash=prediction interval")

# Contrast this with the model prediction of y(t), i.e., put a line through the points
# Intervals are for new data not the blue dots
# (which were used to fit the model so are not new)
library(ggplot2)
d <- fitted(fit, type = "ytT", interval="confidence", level=0.95)
d2 <- fitted(fit, type = "ytT", interval="prediction", level=0.95)
d$.lwr <- d2$.lwr
d$.upr <- d2$.upr
ggplot(data = d) +
  geom_line(aes(t, .fitted), linewidth = 1) +
  geom_point(aes(t, y), color = "blue", na.rm=TRUE) +
  geom_ribbon(aes(x = t, ymin = .conf.low, ymax = .conf.up), alpha = 0.3) +
  geom_line(aes(t, .lwr), linetype = 2) +
  geom_line(aes(t, .upr), linetype = 2) +
  facet_grid(~.rownames) +
  xlab("Time Step") + ylab("Count") +
  ggtitle("Blue=data, Black=estimate, grey=CI, dash=prediction interval")
```

`zscore` *z-score a vector or matrix*

Description

Removes the mean and standardizes the variance to 1.

Usage

```
zscore(x, mean.only = FALSE)
```

Arguments

x	n x T matrix of numbers
mean.only	If TRUE, only remove the mean.

Details

n = number of observation (y) time series. T = number of time steps in the time series.

The z-scored values (z) of a matrix of y values are $z_i = \Sigma^{-1}(y_i - \bar{y})$ where Σ is a diagonal matrix with the standard deviations of each time series (row) along the diagonal, and $\bar{y}$ is a vector of the means.

Value

n x T matrix of z-scored values.

Author(s)

Eli Holmes, NOAA, Seattle, USA.

Examples

```
zscore(1:10)
x <- zscore(matrix(c(NA, rnorm(28), NA), 3, 10))
# mean is 0 and variance is 1
apply(x, 1, mean, na.rm = TRUE)
apply(x, 1, var, na.rm = TRUE)
```

Index