Package 'MARSS'

February 4, 2020

Type Package

Title Multivariate Autoregressive State-Space Modeling

Version 3.10.12

Date 2020-01-28

Depends R (>= 3.1.0)

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

Suggests broom, Formula, ggplot2, Hmisc, lattice, lme4, maps, Matrix, stringr, survival, xtable

Author Eli Holmes, Eric Ward, Mark Scheuerell, and Kellie Wills, NOAA, Seattle, USA

Maintainer Elizabeth Holmes - NOAA Federal <eli.holmes@noaa.gov>

Description The MARSS package provides maximum-likelihood parameter estimation for constrained and unconstrained linear multivariate autoregressive state-space (MARSS) models fit to multivariate time-series data. Fitting is primarily via an Expectation-Maximization (EM) algorithm, although fitting via the BFGS algorithm (using the optim function) is also provided. MARSS models are a class of dynamic linear model (DLM) and vector autoregressive model (VAR) model. Functions are provided for parametric and innovations bootstrapping, Kalman filtering and smoothing, bootstrap model selection criteria (AICb), confidences intervals via the Hessian approximation and via bootstrapping and calculation of auxiliary residuals for detecting outliers and shocks. The user guide shows examples of using MARSS for parameter estimation for a variety of applications, model selection, dynamic factor analysis, outlier and shock detection, and addition of covariates. Type RShowDoc(``UserGuide'', package=``MARSS'') at the R command line to open the MARSS user guide. Online workshops (lectures and computer labs) at <https://nwfsc-timeseries.github.io/> See the NEWS file for update information.

License GPL-2

LazyData yes

BuildVignettes yes

ByteCompile TRUE

URL https://nwfsc-timeseries.github.io/MARSS

BugReports https://github.com/nwfsc-timeseries/MARSS/issues

NeedsCompilation no

Repository CRAN

Date/Publication 2020-02-04 05:30:02 UTC

R topics documented:

`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. Get started quickly, go to the Quick Start Guide (or type `RShowDoc("Quick_Start",package="MARSS")` at the command line). To open the MARSS User Guide with many vignettes and examples, go to User Guide (or type `RShowDoc("UserGuide",package="MARS`

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

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

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

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

All parameters can be time-varying; "_t" is left off the parameters to remove clutter. Note, by default $V0$ is a matrix of all zeros and thus x_1 or 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 $log(a)$ are not allowed as those would not be linear.

The MARSS model parameters, hidden state processes (x), and observations (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 Z parameters 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 a variety of output including 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 in tibble format,

Details

Main MARSS functions:

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

`fitted.marssMLE` Fitted Y and X estimates output as a tibble. Output can be conditioned on all the data (T), data up to t-1, or data up to t.

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

`tidy.marssMLE` Parameter estimates and fitted Y and X estimates output as a tibble. With confidence intervals if requested

`augment.marssMLE` Model residuals as a tibble.

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

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

`glance.marssMLE` Brief summary of fit.

`logLik.marssMLE` Log-likelihood.

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

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

`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.

eli(dot)holmes(at)noaa(dot)gov and eric(dot)ward(at)noaa(dot)gov,

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.

---

augment.marssMLE *Return the model and state fitted values, residuals, and residual sigma*

Description

augment.marssMLE returns a data.frame with fitted values, residuals, and upper and lower confidence intervals (if requested) for the fitted observations or states. augment is concerned with predictions of the states or observations at time t, i.e. the right part of a MARSS model equation with the error terms left off. The error terms are the residuals (the v_t or w_t). tidy.marssMLE is concerned with estimates of values (parameters, states or observations) conditioned on all the data.

Usage

```
augment.marssMLE(x, type = c("ytT", "xtT"),
                              interval = c("none", "confidence", "prediction"),
                              conf.level = 0.95,
                              form=attr(x[["model"]], "form")[1])
```

Arguments

x	a marssMLE object
type	What types of fitted values and residuals to return. ytT (observations) and xtT (states) are the values conditioned on all the data. Read the details below for xtT. tidy would be the more common function for returning xtT (smoothed state) estimates.
interval	Type of interval: none, confidence or prediction interval. If the confidence, approximate intervals from the standard errors of the fitted values is given.
conf.level	Confidence level.
form	If you want the augment function to use a different augment function than augment_form. This might be useful if you manually specified a DFA model and want to use augment.dfa for rotating.

Details

See residuals.marssMLE for a discussion of the residuals calculations for MARSS models. The reported CIs are the approximate CIs computed using the standard deviations: qnorm(alpha/2)*se.fit + fitted.

observations (type="ytT")

This returns a model predicted value of the response ($\mathbf{y}$) and the difference between the model prediction and the observed data is the residual. If there is no data point, then the residual is NA. The standard errors help visualize how well the model fits to the data. See fitted.marssMLE for a discussion of the calculation of the fitted values for the observations (the modeled values). The standardized residuals can be used for outlier detection. See residuals.marssMLE and the chapter on shock detection in the MARSS User Guide.

In the literature on state-space models, it is very common to use the one-step ahead predicted values of the data. The fitted values returned by type=ytT are NOT the one-step ahead values and the residuals are NOT the one-step ahead residuals (called Innovations in the state-space literature). If you want the one-step ahead fitted values, you can use fitted(x,conditioning="t-1"). The innovations are also returned by MARSSkf in Innov.

states (type="xtT")

If you want the expected value of the states and an estimate of their standard errors (for confidence intervals), then augment is not what you want to use. You want to use tidy.marssMLE to return the smoothed estimate of the state. augment(MLEobj,type="xtT") returns a model prediction of $\mathbf{x}(t)$ given $\mathbf{x}_{t-1}$. The residuals returned are for $\mathbf{w}_t$, the difference between the two. These types of residuals are used for outlier detection or shock detection in the state process. They are also used for model diagnostics. See residuals.marssMLE and read the references cited.

Value

If interval = "none", the data frame has the following columns:

.fitted	Fitted values of observations or states. 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.resid	Standardized residuals. Used for outlier detection. See residuals.marssMLE.

If interval = "confidence", the following are added to the data frame:

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

If interval = "prediction", the following are added to the data frame:

.sd.x or .sd.y	Standard deviation of new x or y iid values.
.lwr	Lower range at alpha = 1-conf.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 standard deviation is for new x or y, not estimated x nor y used in the model. Do not plot the observed data nor the states estimates on the confidence or prediction intervals. For that you need to use .sigma and construct intervals as noted above. autoplot() will show the residuals intervals on the observation plot.

Note

Within the base code, a form-specific internal augment function is called to allow the output to vary based on form: augment_dfa, augment_marss, augment_marxss.

Examples

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

library(broom)
library(ggplot2)
theme_set(theme_bw())

# Make a plot of the observations and model fits
d <- augment(MLEobj, interval = "confidence")
ggplot(data = d) +
  geom_line(aes(t, .fitted)) +
  geom_point(aes(t, y)) +
  geom_ribbon(aes(x = t, ymin = .conf.low, ymax = .conf.up), linetype = 2, alpha = 0.1) +
  facet_grid(~.rownames) +
  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 states with CIs. Use tidy() for that.
d <- augment(MLEobj, type = "xtT")
```

```
ggplot(data = d) +
  geom_point(aes(t, xtT)) +
  geom_line(aes(x = t, .fitted)) +
  facet_grid(~.rownames) +
  xlab("Time Step") + ylab("Count") +
  ggtitle("xtT (points) and predition (line)")
```

`coef.marssMLE` *Coefficient function for MARSS MLE objects*

Description

`MARSS()` outputs `marssMLE` objects. `coef(MLEobj)`, where MLEobj is one's 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"`).

Usage

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

Arguments

`object`	A `marssMLE` object.
`...`	Other arguments for coef.
`type`	What to print. Default is "list". If you input type as a vector, coef returns a list of output. See examples. • "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`	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"`. This is mainly useful is for debugging numerical problems since the error reports will be for the "marss" form.

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.

eli(dot)holmes(at)noaa(dot)gov

See Also

augment.marssMLE, tidy.marssMLE, print.marssMLE

Examples

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

coef(MLEobj)
coef(MLEobj, type = "vector")
coef(MLEobj, type = "matrix")
# to retrieve just the Q matrix
coef(MLEobj, 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.

eli(dot)holmes(at)noaa(dot)gov

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 (`RShowDoc("UserGuide",package="MARSS")`) in the PVA chapter.

Author(s)

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

eli(dot)holmes(at)noaa(dot)gov

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 datasets: 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 *fitted function for MARSS MLE objects*

Description

MARSS() outputs marssMLE objects. fitted(MLEobj), where MLEobj is the output from a MARSS() call, will return the modeled value of $\mathbf{y}_t$ or $\mathbf{x}_t$. For $\mathbf{y}_t$, this is $\mathbf{Z}_t\tilde{\mathbf{x}}_t + \mathbf{a}_t$. For $\mathbf{x}_t$, this is $\mathbf{B}_t\tilde{\mathbf{x}}_{t-1} + \mathbf{u}_t$. If you want the estimate of $\mathbf{x}_t$, then use tidy.marssMLE.

Usage

```
  ## S3 method for class 'marssMLE'
fitted(object, ...,
  type = c("ytT", "xtT", "ytt", "ytt1", "xtt1"),
  interval = c("none", "confidence", "prediction"),
  conf.level = 0.95,
  output = c("tibble", "matrix"))
```

Arguments

object	A marssMLE object.
...	Other arguments. Used for backwards compatibility with old arguments.
type	Fitted values for the observations (y) or the states (x). If 'tT', then the estimate at time 't' is conditioned on all the data. If 'tt', then the estimate is conditioned on data up to time t. If 'tt1', estimate is conditioned on data up to time t-1. This is also known as one-step-ahead estimate or for y, the innovations.
interval	If interval="confidence", then the standard error and confidence interval of the fitted value is returned.
conf.level	Level for the intervals if interval != "none".
output	tibble or list of matrices

Details

observation fitted values

The model predicted (fitted) $\hat{\mathbf{y}}_t$ is $\mathbf{Z}_t\tilde{\mathbf{x}}_t + \mathbf{a}_t$, where the model is written in marss form. See MARSS.marss for a discussion of the conversion of MARSS models with covariates ($\mathbf{c}$ and $\mathbf{d}$) into marss form which is how models are written in the internal MARSS algorithms).

$\tilde{\mathbf{x}}_t$ is the expected value of the states at time t. If type="ytT", $\tilde{\mathbf{x}}_t$ is the expected value conditioned on all the data, i.e. xtT returned by MARSSkf(). If type="ytt1", then 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 se and interval is for the fitted $\mathbf{y}$. The standard error of the fitted values is $\mathbf{Z}_t\tilde{\mathbf{V}}_t\mathbf{Z}_t^\top$. If interval="prediction", the standard deviation of new iid $\mathbf{y}$ datasets is returned. The standard deviation of new $\mathbf{y}$ is $\mathbf{Z}_t\tilde{\mathbf{V}}_t\mathbf{Z}_t^\top + \tilde{\mathbf{V}}_t$ is either conditioned on 1:T, 1:t, or 1:t-1 depending on type. Do not plot the data used in the model on these intervals. For that you want the conditional model residuals (and se's). Use augment.marssMLE.

state fitted values

The model predicted $\mathbf{x}(t)$ given $\mathbf{x}_{t-1}$ is $\mathbf{B}(t)\tilde{\mathbf{x}}(t-1)+\mathbf{u}(t)$, where the model is written in "marss" form (MARSS.marss). This type of state fitted value is used for process outlier detection and shock detection. See residuals.marssMLE and read the references cited.

If you want estimates of the states, rather than the model predictions based on $\mathbf{x}_{t-1}$ then you'll want either the states estimate conditioned on all the data (or conditioned on the data up to time $t-1$ or up to time t). These are returned by MARSSkf() in xtT, xtt1 and xtt respectively. Which one you

want depends on your objective and application. You can also use the `tidy.marssMLE()` function to return a data.frame (tibble) with the estimated states with standard errors and intervals.

$\tilde{\mathbf{x}}_{t-1}$ used in the prediction is the expected value of the states at time $t-1$. 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 fitted state values conditioned on data up to t is 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.

The intervals returned by `fitted.marssMLE` for the fitted states, are not typically what one uses or needs–however might be useful for simulation work. If you are doing outlier detection or shock detection, you need the intervals on the smoothed state residuals which are returned by `residuals.marssMLE` or `augment.marssMLE`. If you want intervals on the states estimates, use `tidy.marssMLE`.

If interval="confidence", the standard error of the fitted values (meaning the 'expected value of $\mathbf{X}_t$' given $\mathbf{X}_{t-1}$) is returned. The standard error of the fitted value is $\mathbf{B}_t\tilde{\mathbf{V}}_{t-1}\mathbf{B}_t^\top$. If interval="prediction", the standard deviation of $\mathbf{X}_t$ given $\mathbf{X}_{t-1}$ is output. The latter is $\mathbf{B}_t\tilde{\mathbf{V}}_{t-1}\mathbf{B}_t^\top + \mathbf{Q}$ (notice it includes Q). $\tilde{\mathbf{V}}_{t-1}$ is either conditioned on 1:T or 1:t-1 depending on `type`. Do not plot (or compare) the estimate of $\mathbf{x}_t$ to the intervals for the fitted values. You need the conditional states residuals intervals in that case; use `augment.marssMLE`.

Value

If interval="none" (the default), a T column matrix of the fitted values with one row for each observation (or state) time series is returned.

If interval = "confidence", the following are returned in a list:

`.fitted`	Fitted values of observations (y) or states (x). See details.
`.se.fit`	Standard errors of fitted values
`.conf.low`	Lower confidence level at `alpha = 1-conf.level`. The interval is approximated using qnorm(alpha/2)*.se.fit + .fitted.
`.conf.up`	Upper confidence level. The interval is approximated using qnorm(1-alpha/2)*.se.fit + .fitted.

The confidence interval is for the fitted value, i.e. $\mathbf{Z}_t\tilde{\mathbf{x}}_t + \mathbf{a}_t$ or $\mathbf{B}_t\tilde{\mathbf{x}}_{t-1} + \mathbf{u}_t$.

If interval = "prediction", the following are returned in a list:

`.fitted`	Fitted values of observations (y) or states (x). See details.
`.sd.x or .sd.y`	Standard deviation of new $\mathbf{x}_t$ or $\mathbf{y}_t$ iid values.
`.lwr`	Lower range at `alpha = 1-conf.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 $\mathbf{y}_t$, not estimated $\mathbf{x}_t$ nor $\mathbf{y}_t$ used in the model. Do not plot the observed data nor the states estimates on these intervals. For that you need the residuals intervals provided by `augment.marssMLE`.

Author(s)

Eli Holmes, NOAA, Seattle, USA.

eli(dot)holmes(at)noaa(dot)gov

See Also

MARSSkf, augment.marssMLE, tidy.marssMLE

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)
plot(nile, type = "p", pch = 16, col = "blue")
lines(fitted(fit, type="ytT")[1, ], col = "red", lwd = 2)
lines(fitted(fit, type="ytt1")[1, ], col = "grey", lwd = 2)
```

---

glance.marssMLE *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

```
glance.marssMLE(x, ...)
```

Arguments

x	A marssMLE object
...	Not used.

Examples

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

library(broom)
glance(MLEobj)
```

---

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. "harborSeallnomiss" is an extrapolated data set where missing values in the original dataset 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)
```

---

is.marssMLE *Tests marssMLE object for completeness*

Description

Tests an marssMLE for completeness to determine if it has all the pieces and attributes necessary to be passed to MARSS functions for fitting, filtering, smoothing, or displaying. Internal function, use `MARSS:::` to access. This is a very slow function which should not be called repeatedly in a `for` loop for example.

Usage

```
is.marssMLE(MLEobj)
```

Arguments

MLEobj	An object of class marssMLE. See Details.

Details

The is.marssMLE() function checks components marss, start and control, which must be present for estimation by functions e.g. MARSSkem(). Components returned from estimation must include at least method, par, kf, numIter, convergence and logLik. Additional components (e.g. AIC) may be returned, as described in function help files.

model An object of class marssMODEL in whatever form the user specified in the call to MARSS(). Default is form "marxss".

marss An object of class marssMODEL in "marss" forms, needed for all the base MARSS functions.

start List with matrices specifying initial values for parameters to be used (if needed) by the maximization algorithm.

- B Initial value(s) for B matrix (m x m).
- U Initial value(s) for U matrix (m x 1).
- Q Initial value(s) for Q variance-covariance matrix (m x m).
- Z Initial value(s) for Z matrix (n x m).
- A Initial value(s) for A matrix (n x 1).
- R Initial value(s) for R variance-covariance matrix (n x n).
- x0 Initial value(s) for initial state vector (m x 1).
- V0 Initial variance(s) for initial state variance (m x m).

control A list specifying estimation options. The following options are needed by MARSSkem(). Other control options can be set if needed for other estimation methods, e.g. the control options listed for optim for use with MARSSoptim(). The default values for control options are set in alldefaults[[method]] which is specified in MARSSsettings.R.

- minit The minimum number of iterations to do in the maximization routine (if needed by method).
- maxit Maximum number of iterations to be used in the maximization routine (if needed by method).
- 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.
- abstol The logLik.(iter-1)-logLik.(iter) convergence tolerance for the maximization routine. Both the abstol and the slope of the log of the parameters versus the log iteration tests must be met for convergence.

`trace` A positive integer. If not 0, a record will be created during maximization iterations (what's recorded depends on method of maximization). -1 turns off most internal error checking.

`safe` Logical. If TRUE, then the Kalman filter is run after each update equation in the EM algorithm. This slows down the algorithm. The default is FALSE.

`allow.degen` If TRUE, replace Q or R diagonal elements by 0 when they become very small.

`min.degen.iter` Number of iterations before trying to set a diagonal element of Q or R to zero).

`degen.lim` How small the Q or R diagonal element should be before attempting to replace it with zero.

`silent` Suppresses printing of progress bar, error messages and convergence information.

`method` A string specifying the estimation method. MARSS allows "kem" for EM and "BFGS" for quasi-Newton. Once the model has been fitted, additional elements are added.

`par` A list with 8 matrices of estimated parameter values Z, A, R, B, U, Q, x0, V0.

`states` Expected values of the x (hidden states).

`states.se` Standard errors on the estimates states.

`ytT` Expected values of the y. This is just y for non-missing y.

`ytT.se` Standard errors on the ytT. This will be 0 for non-missing y.

`kf` A list containing Kalman filter/smoother output if `control$trace` is > 0.

`Ey` A list containing expectations involving y. Output if `control$trace` is > 0.

`numIter` Number of iterations which were required for convergence.

`convergence` Convergence status and errors. 0 means converged successfully. Anything else means an error or warning.

`logLik` Log-likelihood.

`AIC` AIC

`AICc` Corrected AIC.

`call` A list of all the arguments passed into the MARSS call. Not required for most functions, but is a record of what was used to call MARSS for checking and can be used to customize the printing of MARSS output.

Value

TRUE if no problems; otherwise a message describing the problems.

Author(s)

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

See Also

`marssMODEL`, `MARSSkem`

isleRoyal *Isle Royale Wolf and Moose Data*

Description

Example data set for estimation of species interaction strenths. 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 (F) 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)
```

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.

eli(dot)holmes(at)noaa(dot)gov

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:

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

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

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

All parameters (except x0 and V0) can be time-varying; t is left off the parameters to remove clutter. All parameters can be zero-ed out, including the variance matrices. Parameters can be estimated or fixed. The parameters are everything except $mathbfx$, $mathbfy$, $mathbfv$, $mathbfw$, $mathbfc$ and $mathbfc$. $mathbfy$ are data (missing values allowed). c and d are inputs.

The parameters 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 = "kem",
    form = "marxss",
    fit=TRUE,
    silent = FALSE,
    control = NULL,
    fun.kf = "MARSSkfas",
    ...)
```

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.
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 parameter model 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	TRUE/FALSE Suppresses printing of full error messages, warnings, progress bars and convergence information. 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.

- 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 of 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 Q or 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 MARSS. This is particularly useful for bootstrapping and simulation studies.
- silent TRUE/FALSE(default), Suppresses all printing including progress bars, error messages and convergence information. 0, Turns on all printing of progress bars, fitting information and error messages. 2, Prints a brief success/failure message.
- safe TRUE/FALSE(default), 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.

`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 implimentation 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).

`...` 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 c(t) or 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 Z parameters has a specific form and the Q is set at i.i.d 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 ML 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 ML values. EM steps up the likelihood and the convergence test is based on the rate of change of the LL 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 x equation). In some formulations (e.g. in `KFAS`), the process equation is $x(t) = B(t-1)x(t-1) + w(t-1)$ so $B(t-1)$ goes with $x(t)$ not $B(t)$. 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`, `augment.marssMLE` 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 Z, A, R, B, U, Q, x0, V0 in marss form. Use print.marssMLE, tidy.marssMLE or coef.marssMLE for outputing the model estimates in the MARSS() call (e.g. the default "marxss" form).
states	The expected value of x conditioned on all the data, i.e. smoothed states.
states.se	The standard errors of the expected value of x.
ytT	The expected value of y conditioned on the data. Note this is just y for those y that are not missing.
ytT.se	The standard errors of the expected value of y. Note this is 0 for any non-missing y.
numIter	Number of iterations required for convergence.
convergence	Convergence status. 0 means converged successfully. Anything else is a warning or error. 2 means the MLEobj has an error; the MLEobj 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.

eli(dot)holmes(at)noaa(dot)gov and eric(dot)ward(at)noaa(dot)gov

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`, `print.marssMODEL`, `MARSS.marxss`, `MARSS.dfa`, `augment.marssMLE`, `tidy.marssMLE`, `coef.marssMLE`

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.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 D and d are incorporated into the A matrix and C and c are incorporated into the U matrix.

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

$$x(t+1) = B(t)x(t) + U(t) + G(t)w(t), where w(t)\ MVN(0, Q(t))$$

$$y(t) = Z(t)x(t) + A(t) + H(t)v(t), where v(t)\ MVN(0, R(t))$$

$$x(1)\ MVN(x0, V0) or x(0)\ MVN(x0, V0)$$

Note, by default $V0$ is a matrix of all zeros and thus $x(1)$ or $x(0)$ is treated as an estimated parameter not a diffuse prior.

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,silen
= 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:

$$x(t+1) = B(t)x(t) + U(t) + C(t)c(t) + G(t)w(t), where w(t)\ MVN(0, Q(t))$$

$$y(t) = Z(t)x(t) + A(t) + D(t)d(t) + H(t)v(t), where v(t)\ MVN(0, R(t))$$

$$x(1)\ MVN(x0, V0) or x(0)\ MVN(x0, V0)$$

Note, by default $V0$ is a matrix of all zeros and thus $x(1)$ or $x(0)$ 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: "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 (`RShowDoc("Quick_Start",package="MARSS")`) or the User Guide (`RShowDoc("UserGuide",package="MARSS")`) for examples.

The argument `model` must be a list. The elements in the list specify the structure for the B, u, C, c, Q, Z, a, D, d, R, x0, and V0 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 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 B, but in this case all the eigenvalues of 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 U or x0. Notice that U is capitalized.

- 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 A. Care must be taken when specifying A so that the model is not under-constrained and unsolveable model. The default "scaling" only applies to Z matrices that are design matrices (only 1s and 0s and all rows sum to 1). When a column in Z has multiple 1s, the first row with a 1 is assigned A=0 and the rows with 1s for that column have an estimated A. This is used to treat A as an intercept where one A for each X (hidden state) is fixed at 0 and any other Ys associated with that X have an estimated A value. This ensures a solvable model (when Z is a design matrix). 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 Q is a m x m matrix. g is the num of columns in 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 R is a n x n matrix. h is the num of columns in 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 (D) or m rows (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, y. The numbers of rows in d must be the same as number of columns in D; similarly for c and C. c and d are lower case.
- 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 (G) or n rows and h columns (H). g is the dim of Q and h is the dim of R.
- tinitx Default=0. Whether the initial state is specified at t=0 (default) or t=1.

All parameters except x0 and V0 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 dataset 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,silen
= 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 pertubation 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 and Eric Ward, NOAA, Seattle, USA.

eli(dot)holmes(at)noaa(dot)gov, eric(dot)ward(at)noaa(dot)gov

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 Q or 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.

eli(dot)holmes(at)noaa(dot)gov, eric(dot)ward(at)noaa(dot)gov

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
```

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-likelikhood 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 funcation 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 E(LL(theta)|data,theta') but one of the derivatives will be the derivative of the E(X|data,theta') with respect to theta'. 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.

eli(dot)holmes(at)noaa(dot)gov

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")
```

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 implemention 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.

eli(dot)holmes(at)noaa(dot)gov

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), ]
MLEobj <- MARSS(dat)
MARSS:::MARSSharveyobsFI(MLEobj)
```

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

Description

Computes the expected value of random variables involving Y. Users can also use `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	Return only ytT, OtT, yxtT, and yxttpT (values conditioned on the data from 1:T). If only.kem If TRUE, only return (and compute) values 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 (E[Y(t),X(t-1)|1:T]), var.ytT (var[Y(t)|1:T]), and var.EytT (var_X[E_Y|x[Y(t)|1:T,x(t)]]).

Details

For state space models, MARSShatyt() computes the expectations involving Y. If Y is completely observed, this entails simply replacing Y with the observed y. When 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), y(1) is the observed data (for t=1:T) while y(2) is the unobserved data. 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|x[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).
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.

eli(dot)holmes(at)noaa(dot)gov

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, ]
MLEobj <- MARSS(dat)
EyList <- MARSShatyt(MLEobj)
```

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 the function, 'fdHess' is used for all numerical estimates used in the package.

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 ML 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.

eli(dot)holmes(at)noaa(dot)gov

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 MLEs.

Author(s)

Eli Holmes, NOAA, Seattle, USA.

eli(dot)holmes(at)noaa(dot)gov

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.

eli(dot)holmes(at)noaa(dot)gov

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$x0=-99`, then starting values for x0 are estimated by a linear regression through the count data assuming A=0. This will be a poor start if `inits$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.

eli(dot)holmes(at)noaa(dot)gov

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.

eli(dot)holmes(at)noaa(dot)gov, eric(dot)ward(at)noaa(dot)gov

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, ]
MLEobj <- MARSS(dat, model = list(U = "equal", Q = diag(.01, 2)))
boot.obj <- MARSSinnovationsboot(MLEobj)
```

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 scatch 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 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 soln 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 Q or 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 likelikelihood 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=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=3 No convergence diagnostics were computed because the MLE object 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=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 filter.

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

`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 compuation 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 Q or 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

fixed and high R matrix and the variance in that R matrix was much higher than what is actually in the data, then you might drive Q to zero. Also if you try to fit a structurally inadequate model, then it is not unusual that 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 structually 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.

eli(dot)holmes(at)noaa(dot)gov, eric(dot)ward(at)noaa(dot)gov

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 Totronto, 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 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]`RShowDoc("EMDerivation",` to open a copy.

See Also

`MARSSkf`, `marssMLE`, `MARSSoptim`, `MARSSinfo`

Examples

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

```
kemfit <- MARSSkem(MLEobj)
```

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()` will be more stable and `MARSSkfss()` returns some values that `MARSSkfas()` does not.

Usage

```
MARSSkf( MLEobj, only.logLik=FALSE, return.lag.one=TRUE, return.kfas.model=FALSE )
MARSSkfss( MLEobj )
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 specificitions. `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.
`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 (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.).

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 $x(t)$ 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 expected value of the initial state, x0, is an estimated parameter (or treated as a prior). This E(initial state) can be treated in two different ways. One can treat it as x00, meaning E(x at t=0 | y at t=0), and then compute x10, meaning E(x at t=1 | y at t=0), from x00. Or one can simply treat the initial state as x10, meaning E(x at t=1 | y at t=0). The approaches lead to the same parameter estimates, but the likelihood is written slightly differently in each case and you need your likelihood calculation to correspond to how the initial state is treated in your model (either x00 or x10). The EM algorithm in the MARSS package (`MARSSkem()`) provides both Shumway and Stoffer's derivation that uses tinitx=0 and Ghahramani et al algorithm which uses tinitx=1. The `MLEobj$model$tinitx` argument specifies whether the initial states (specified with x0 and V0) is at t=0 (`tinitx=0`) or t=1 (`tinitx=1`).

`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 slightly to 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 Q diagonals equal to 0), partially perfect observation models (some or all elements of the R diagonal equal to 0) and fixed (albeit unknown) initial states (some or all elements of the 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 x(1) conditioned on y(0), which is missing). When the initial state is specified at t=0 (as x(0) conditioned on y(0), which is missing), `MARSSkfas` computes the required E(x(1)|y(0)) and var(x(1)|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 `MARSSkf()` has many internal checks and error messages which can help debug numerical problems (but slow things down). Also `MARSSkf()` 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). "V" elements are called "P" in Shumway and Stoffer (S&S eqn 6.17 with s=T). The output is referenced against equations in Shumway and Stoffer (2006); 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, so E[x | y] is really E[x | theta, y] where theta is the parameter list.

`xtT`	State first moment conditioned on y(1:T): E[x(t) | y(1:T)] (m x T matrix). Kalman smoother output.
`VtT`	State variance conditioned on y(1:T): E[(x(t)-xtT(t))(x(t)-xtT(t))'| | y(1:T)] (m x m x T array). Kalman smoother output. P_t^T in S&S (S&S eqn 6.18 with s=T, t1=t2=t). State second moment E[x(t)x(t)'| y(1:T)] = VtT(t)+xtT(t)xtT(t)'
`Vtt1T`	State lag-one covariance E[(x(t)-xtT(t))(x(t-1)-xtT(t-1))' | 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 E[x(t)x(t-1)'| y(1:T)] = Vtt1T(t)+xtT(t)xtT(t-1)'.
`x0T`	Initial state estimate E[x(i) | y(1:T)] (m x 1). If control$kf.x0="x00", i=0; if ="x10", i=1. Kalman smoother output.
`V0T`	Estimate of initial state covariance matrix E[x(i)x(i)' | y(1:T)] (m x m). If model$tinitx=0, i=0; if =1, i=1. Kalman smoother output. P_0^T in S&S.
`J`	(m x m x T) Kalman smoother output. Only for `MARSSkfss`. (ref S&S eqn 6.49)
`J0`	J at init time (t=0 or t=1) (m x m x T). Kalman smoother output. Only for `MARSSkfss`.
`xtt`	State first moment conditioned on y(1:t): E[x(t) | y(1:t)] (m x T). Kalman filter output. (S&S eqn 6.17 with s=t)
`xtt1`	State first moment conditioned on y(1:t-1): E[x(t) | y(1:t-1)] (m x T). Kalman filter output. (S&S eqn 6.17 with s=t-1)
`Vtt`	State variance conditioned on y(1:t): E[(x(t)-xtt(t))(x(t)-xtt(t))'| | 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). State second moment E[x(t)x(t)'| y(1:t)] = Vtt(t)+xtt(t)xtt(t)'
`Vtt1`	State variance conditioned on y(1:t-1): E[(x(t)-xtt1(t))(x(t)-xtt1(t))'| | y(1:t-1)] (m x m x T array). Kalman filter output. P_t^t in S&S (S&S eqn 6.18 with s=t-1, t1=t2=t). State second moment E[x(t)x(t)'| y(1:t-1)] = Vtt1(t)+xtt1(t)xtt1(t)'
`Kt`	Kalman gain (m x m x T). Kalman filter output (ref S&S eqn 6.23). Only for `MARSSkfss`.
`Innov`	Innovations y(t) - E[y(t) | 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. eli(dot)holmes(at)noaa(dot)gov

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. `http://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 Assosiation, 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 MLEobj
# MARSS calls MARSSinits to construct default initial values
# with fit = FALSE, the $par element of MLEobj will be NULL
MLEobj <- MARSS(dat, fit = FALSE)
# MARSSkf needs a marss MLE object with the par element set
MLEobj$par <- MLEobj$start
# Compute the kf output at the params used for the inits
kfList <- MARSSkf(MLEobj)
```

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, 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.marssMLE, tidy.marssMLE, or augment.marssMLE to extract output.

Methods

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

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

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

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

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

Author(s)

Eli Holmes and Kellie Wills, NOAA, Seattle, USA

eli(dot)holmes(at)noaa(dot)gov

See Also

is.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.

eli(dot)holmes(at)noaa(dot)gov

`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$maxit iterations and no evidence of degenerate solution.
	convergence=1 Maximum number of iterations MLEobj$control$maxit was reached before MLEobj$control$abstol condition was satisfied.
	convergence=10 Some of the variance elements appear to be degenerate. T
	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".
logLik	Log-likelihood.
states	State estimates from the Kalman filter.

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

eli(dot)holmes(at)noaa(dot)gov

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. 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.marssMLE. Or use tidy.marssMLE.

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.

eli(dot)holmes(at)noaa(dot)gov

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
```

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 residuals(object,conditioning="T").

Usage

```
MARSSresiduals.tT(object, Harvey=FALSE, normalize=FALSE, silent=FALSE)
```

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
`silent`	If TRUE, don't print inversion warnings.

Details

This function returns the raw, Cholesky standardized and 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).

The residuals matrix has a value for each time step. The residuals in column t rows 1 to n are the model residual 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 $\mathbf{x}_t$ to $\mathbf{x}_{t+1}$. 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 $\mathbf{x}_t$:

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

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. So

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

where $\tilde{\mathbf{x}}_t^T$ 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,]
MLEobj = MARSS(dat)
Z = coef(MLEobj, type="matrix")$Z
A = coef(MLEobj, type="matrix")$A
res1 = dat - Z %*% MLEobj$states - A %*% matrix(1,1,ncol(dat))
res2 = residuals(MLEobj)$model.residuals
```

state.residuals

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

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

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

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

where $\tilde{\mathbf{x}}_t^T$ is the Kalman smoother estimate of the states at time t and $\tilde{\mathbf{x}}_{t-1}^T$ is the Kalman smoother estimate of the states at time $t-1$. The estimated state residuals are returned in `state.residuals` and rows $n+1$ to $n+m$ of `residuals`. 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)
MLEobj = MARSS(dat)
B = coef(MLEobj, type="matrix")$B
U = coef(MLEobj, type="matrix")$U
statest = MLEobj$states[,2:TT]
statestm1 = MLEobj$states[,1:(TT-1)]
res1 = statest - B %*% statestm1 - U %*% matrix(1,1,TT-1)
res2 = residuals(MLEobj)$state.residuals
```

Note that the state residual at the last time step (T) 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 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.

Standardized residuals

`residuals.marssMLE` 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 Cholesky decomposition of `var.residuals`. 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 standarized residual in the last time step will be all NA.

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,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 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.

Normalized residuals

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

$$y_t = Zx_t + a + v_t$$

$$x_t = Bx_{t-1} + u + w_t$$

Harvey et al (1998) writes the model as

$$y_t = Zx_t + a + Hv_t$$

$$x_t = Bx_{t-1} + u + Gw_t$$

with the variance of V_t and W_t equal to 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 I instead of $\mathbf{Q}$ and $\mathbf{R}$.

Missing or left-out data

E$[\widehat{\mathcal{E}}_t]$ and 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 obseved $\mathbf{y}^{(1)}$ (Holmes 2014). If there are no missing values, this is not very interesting as E$[\hat{\mathbf{V}}_t|\mathbf{y}^{(1)}] = \hat{\mathbf{v}}_t$ and var$[\hat{\mathbf{V}}_t|\mathbf{y}^{(1)}] = 0$. If we have data that are missing because we left them out, however, E$[\hat{\mathbf{V}}_t|\mathbf{y}^{(1)}]$ and 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 E$[\hat{\mathbf{V}}|\mathbf{y}^{(1)}]$ (notice small $\mathbf{y}$). It is

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

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 $\mathbf{Z}\tilde{\mathbf{x}}_t^T + \mathbf{a}$ 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 $\mathrm{var}[\hat{\mathbf{V}}_t|\mathbf{Y}^{(1)}]$ and variance of $\mathrm{var}[\hat{\mathbf{V}}_t|\mathbf{y}^{(1)}]$ 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).

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 $\tilde{\mathbf{x}}_{t+1}^T - \mathbf{Z}\tilde{\mathbf{x}}_t^T - \mathbf{u}$.

`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 is 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.

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

`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. This for convenience for residuals diagnostics.

`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. 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 value 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.

eli(dot)holmes(at)noaa(dot)gov

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). Eqn 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 eqn 23 and eqn 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 (chol) 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

`residuals.marssMLE`, `MARSSresiduals.tt1`, `fitted.marssMLE`, `plot.marssMLE`

Examples

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

#state residuals
state.resids1 <- residuals(MLEobj, conditioning="T")$state.residuals
#this is the same as
states <- MLEobj$states
```

```
Q <- coef(MLEobj,type="matrix")$Q
state.resids2 <- states[,2:30]-states[,1:29]-matrix(coef(MLEobj,type="matrix")$U,2,29)
#compare the two
cbind(t(state.resids1[,-30]), t(state.resids2))

#normalize to variance of 1
state.resids1 <- residuals(MLEobj, normalize=TRUE, conditioning="T")$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
residuals(MLEobj)$std.residuals
```

`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 `residuals(object,conditioning="t-1")`.

Usage

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

Arguments

`object`	An object of class `marssMLE`.
`method`	Algorithm to use. Currently only "SS".
`normalize`	TRUE/FALSE
`silent`	If TRUE, don't print inversion warnings.

Details

This function returns the conditional expected value (mean) and variance of the model 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 $\mathbf{x}_t$:

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

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

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

where $\tilde{\mathbf{x}}_t^{t-1}$ is the expected value of $\mathbf{X}_t$ conditioned on the data from 1 to $t-1$ 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 $t-1$ and the parameter set Θ. The conditional variance 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`.

Standardized residuals

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

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

These residuals are uncorrelated.

The interpretation of the Cholesky standardized residuals is not straight-forward when the Q and 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,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 muliplied by the inverse of the diagonal matrix formed from the square-root of the diagonal of the variance matrix of the residuals:

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

, 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 V_t and W_t are R and Q and the model is assumed to be written as

$$y_t = Z x_t + a + v_t$$

$$x_t = B x_{t-1} + u + w_t$$

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

$$y_t = Z x_t + a + H v_t$$

$$x_t = B x_{t-1} + u + G w_t$$

with the variance of V_t and W_t equal to I (identity).

`residuals.marssMLE` 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 I instead of R and Q. Note, that the 'normalized' residuals are not the same as the 'standardized' residuals. In former, the unconditional residuals have a variance of I while in the latter it is the conditional residuals that have a variance of I.

Value

A list with the following components

`residuals`	The model residuals conditioned on the data up to time $t-1$ and the set of model parameters. Called the innovations. Residuals associated with missing data will appear as NA.
`var.residuals`	The variance of the model residuals as a n x n x T matrix. The variance exists for all t values including missing data.
`std.residuals`	The Cholesky standardized model residuals as a n x T matrix. This is `residuals` multiplied by the inverse of the Cholesky decomposition of `var.residuals`.
`mar.residuals`	The marginal standardized model residuals as a n 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`.
`msg`	Any warning messages. This will be printed unless Object$control$trace = -1 (suppress all error messages).

Author(s)

Eli Holmes, NOAA, Seattle, USA.

eli(dot)holmes(at)noaa(dot)gov

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`, `fitted.marssMLE`, `plot.marssMLE`

Examples

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

residuals(MLEobj, conditioning="t-1")$std.residuals
```

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.

eli(dot)holmes(at)noaa(dot)gov, eric(dot)ward(at)noaa(dot)gov

See Also

marssMODEL, marssMLE, MARSSboot

Examples

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

# simulate data that are the
# same length as original data and no missing data
sim.obj <- MARSSsimulate(MLEobj, 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(MLEobj, 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 dataset consists for two datasets: 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 phosporous 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 all plots are plotted. Individual plots can be plotted by passing in `type.plot`. 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("observations", "states", "model.residuals",
                    "state.residuals", "model.residuals.qqplot",
                    "state.residuals.qqplot"),
          form=c("marxss", "marss", "dfa"),
          conf.int=TRUE, conf.level=0.95, decorate=TRUE,
          plot.par = list(), ...)
  ## S3 method for class 'marssMLE'
autoplot(x, plot.type=c("observations", "states", "model.residuals",
```

```
                "state.residuals", "model.residuals.qqplot",
                "state.residuals.qqplot", "expected.value.observations",
                "model.residuals.acf", "state.residuals.acf"),
        form=c("marxss", "marss", "dfa"),
        conf.int=TRUE, conf.level=0.95, decorate=TRUE, pi.int = FALSE,
        plot.par = list(), ...)
```

Arguments

x	A marssMLE object.
plot.type	Type of plot. If not passed in, all plots are drawn. Options for arguments include "observations" (fits to the raw data), "states" (estimates of the hidden or latent trends), "model.residuals" (residuals for the observation error), "state.residuals" (residuals associated with the process model), "model.residuals.qqplot" (qq plot for the observation residuals), "state.residuals.qqplot" (qq plot for the state residuals). "expected.value.observations" (estimates of the missing data points), "model.residuals.acf" and "state.residuals.acf" (ACF of the residuals).
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. 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).
...	Other arguments, not used.

Value

If an individual plot is selected using plot.type and autoplot is called, then the ggplot object is returned invisibly.

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 = "observations")

require(ggplot2)
autoplot(fit, plot.type = "observations")

## Not run:
# DFA example
dfa <- MARSS(t(harborSealWA[, -1]), model = list(m = 2), form = "dfa")
plot(dfa, plot.type = "states")

## End(Not run)
```

`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 dataset.

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 dataset 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, and corrected for changes in observer coverage and detectability. 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)
```

`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 estimted 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$model,"form")` .
- "paramvector" A vector of all the estimated values in each matrix. Standard function: `coef(x,type="vector")`. See `coef.marssMLE`.
- "par.se","par.bias","par.lowCIs","par.upCIs" A vector the estimated parameter standard errors, parameter bias, lower and upper confidence intervals. Standard function: `MARSSparamCIs(x)` See `MARSSparamCIs`.
- "xtT" or "states" The estimated states conditioned on all the data. `x$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$states.se`
- "states.cis" Approximate confidence intervals for the states. See `MARSSparamCIs`.
- "model.residuals" The smoothed model residuals. y(t)-E(y(t)|xtT(t)), aka actual data at time t minus the expected value of the data conditioned on the smoothed states estimate at time t. Standard function: `residuals(x)$model.residuals` See `residuals.marssMLE` for a discussion of residuals in the context of MARSS models.
- "state.residuals" The smoothed state residuals. E(xtT(t))-E(x(t)|xtT(t-1)), aka the expected value of x at t conditioned on all the data minus the expected value of x at t conditioned on (x(t-1) conditioned on all the data). Standard function: `residuals(x)$state.residuals` See `residuals.marssMLE`.
- parameter name Returns the parameter matrix for that parameter with fixed values at their fixed values and the estimated values at their estimated values. Standard function: `coef(x,type="matrix")$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.

eli(dot)holmes(at)noaa(dot)gov

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.

eli(dot)holmes(at)noaa(dot)gov

Examples

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

```
print(MLEobj$model)
# this is identical to
print(MLEobj, what = "model")
```

`residuals.marssMLE` *MARSS Residuals*

Description

Calculates the residuals, residuals variance, and standardized residuals for both the one-step-ahead (conditioned on data up to t-1) and smoothed (conditioned on all the data) residuals.

Usage

```
## S3 method for class 'marssMLE'
residuals(object,..., conditioning=c("T", "t-1"), normalize=FALSE, silent=FALSE)
```

Arguments

`object`	An object of class `marssMLE`.
`...`	Additional arguments to be passed to the residuals functions. For conditioning="T", `Harvey=TRUE` can be passed into to use the Harvey et al (1998) algorithm.
`conditioning`	"T" for smoothed residuals and "t-1" for one-step-ahead residuals.
`normalize`	TRUE/FALSE
`silent`	If TRUE, do not print inversion warnings.

Details

For smoothed residuals, see `MARSSresiduals.tT`.

For one-step-ahead residuals, see `MARSSresiduals.tt1`.

Standardized residuals

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

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

These residuals are uncorrelated.

The interpretation of the Cholesky standardized residuals is not straight-forward when the Q and 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,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 muliplied by the inverse of the diagonal matrix formed from the square-root of the diagonal of the variance matrix of the residuals:

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

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 V_t and W_t are R and Q and the model is assumed to be written as

$$y_t = Zx_t + a + v_t$$

$$x_t = Bx_{t-1} + u + w_t$$

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

$$y_t = Zx_t + a + Hv_t$$

$$x_t = Bx_{t-1} + u + Gw_t$$

with the variance of V_t and W_t equal to I (identity).

Missing or left-out data

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

Value

Smoothed residuals

If `conditioning="T"`, a list with the following components

`model.residuals`
: The smoothed model residuals (data minus model predicted values) as a n x T matrix. The residuals are conditioned on all the data and the set of model parameters. Called the smoothations. This is different than the Kalman filter innovations which are conditioned on the data up to $t-1$.

`state.residuals`
: The smoothed state residuals.

`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.

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

`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).

One-step-ahead residuals

If `conditioning="t-1"`, a list with the following components

`residuals`
: The model residuals (data minus model predicted values), as a n x T matrix. The residuals are conditioned on the data up to time $t-1$ and the set of model parameters. Called the innovations. Residuals associated with missing data will appear as NA.

`var.residuals`
: The variance of the model residuals as a n x n x T matrix.

`std.residuals`
: The Cholesky standardized model residuals as a n x T matrix. This is `residuals` multiplied by the inverse of the Cholesky decomposition of `var.residuals`.

`mar.residuals`
: The marginal standardized model residuals as a n 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`.

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

Author(s)

Eli Holmes, NOAA, Seattle, USA.

eli(dot)holmes(at)noaa(dot)gov

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` and `MARSSresiduals.tt1`.

See Also

`MARSSresiduals.tT`, `MARSSresiduals.tt1`, `plot.marssMLE`

Examples

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

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

#normalize to variance of 1
state.resids1 <- residuals(MLEobj, 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
residuals(MLEobj, conditioning="t-1")$std.residuals
```

SalmonSurvCUI *Salmon Survial 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)
```

`tidy.marssMLE` *Return estimated parameters, expected value of X(t) and Y(t) with summary information*

Description

`tidy.marssMLE` returns summary information about the model parameters and estimated state and observation processes. In all cases, all the data are used, thus conditioning is on the data from 1 to T.

For parameters, `tidy.marssMLE` returns their estimates and their confidence intervals. For states (X) and observations (Y), it returns the expected values (mean value) and intervals. If you want to analyze your model residuals or are doing a crossvalidation (leave-one-out or k-fold), you need the residuals intervals. These are the intervals for model residuals (data - model fitted value). Use `augment.marssMLE` for the model residuals and their intervals.

The tidy function is designed to work with the `broom` package and you will need to load that package if you want to call `tidy(fit)` instead of `tidy.marssMLE(fit)`.

Usage

```
tidy.marssMLE(x, type = c("parameters", "xtT", "fitted.ytT", "ytT"),
              conf.int = TRUE,
              conf.level = 0.95,
              form=attr(x[["model"]], "form")[1], ...)
```

Arguments

`x`	a `marssMLE` object
`type`	What you want estimates and intervals for. Parameters, smoothed states (xtT), the fitted y (Z xtT + A), or observations conditioned on all the data (ytT). If you want intervals for new data sets, use fitted.ytT. If you are getting estimates for missing data, use ytT. See details.
`conf.int`	Whether to compute confidence and prediction intervals on the estimates.
`conf.level`	Confidence level. alpha=1-conf.level
`form`	If you want the tidy function to use a different form than that specified in attr(x$model, "form"). Useful if you have a DFA model that you manually set up, which does not have the form attribute set. Normally just ignore and let the function use the "form" set in the attributes.
`...`	Optional arguments. If `conf.int=TRUE`, then arguments to specify how CIs are computed can be passed in. See details and `MARSSparamCIs`. If form="dfa", `rotate=TRUE` can be passed in to rotate the trends (only trends not 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.

type="parameters"

If `type="parameters"`, this returns a data.frame with the estimated parameters of a MARSS model with, optionally, standard errors and confidence intervals. This assembles information available via the `print.marssMLE` and `coef.marssMLE` 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`.

type="xtT"

`tidy.marssMLE` returns the confidence and prediction intervals of the state at time t conditioned on all the data and using the estimated model parameters as true values. The prediction intervals (and .sd.x) 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 x). However, these intervals do not include parameter uncertainty. If you want state residiuals (for residuals analysis), use `residuals.marssMLE` or `augment.marssMLE`.

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, Eqn. 11) notation, its expected value conditioned on all the observed data (1:T) and the model parameters Θ is $\tilde{\mathbf{x}}_t$. In `MARSSkf`, this is `xtT[,t]`. The variance of $\mathbf{X}_t$ conditioned on 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 equation for x. That is because the variance of W(t) conditioned on the data (past, current and FUTURE) is != Q (Q is the unconditional variance).

$\tilde{\mathbf{x}}_t$ (xtT) is an estimate of $\mathbf{x}_t$ (the true value), and the standard error of that estimate is given by $\tilde{\mathbf{V}}_t$ (`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 `tidy.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
```

Intervals for the observation process

For observation process, the expected values and intervals are shown for either new data (`type="fitted.ytT"`) or the observed data set (`type="ytT"`). Details on these are below after this discussion of intervals for the observation process

The types of intervals you want for data (Y part of the MARSS equation) depends on what you are trying to do.

- Get the model predictions of the expected value of new Y or some underlying mean YUse `type="fitted.ytT"`. This returns the fitted values (model predictions = Z x(t)+A) for Y_t conditioned on all the data. Confidence intervals and prediction intervals are returned. The former is the interval for the mean of new data and the latter is the interval for new data (not the mean but data themselves).
- Get the distribution of new data at time t that would be generated by the processSame as above.
- Compare your data to model predictionsIn this case, you want the distribution of the model residuals for the data. Use `augment.marssMLE` with `type="observations"`. You want the standard errors for the observed data minus the fitted values which is what `augment.marssMLE` gives.
- Get estimates and variance of missing data in your data setUse `type="ytT"`. The observed data will have an expected value equal to the observed data and variance of 0, while the missing data will have an expected value and variance conditioned on all the observed data. Note, if R is diagonal then the missing data values (and intervals) will be the same as for `type="fitted.ytT"` but if R is non-diagonal and some y at time t are missing and some are not, then the expected values will be very different.
- Do a leave-one-out cross-validationIn this case, you want the distribution of the model residuals for those left-out values. Use `augment.marssMLE` with `type="observations"`. You want the standard errors for the left-out data minus the fitted values which is what `augment.marssMLE` gives.
- One-step-ahead predictionsUse `fitted.marssMLE` with `type="ytt1"` or `type="xtt1"`. Confidence (mean prediction) and prediction intervals (new data) are returned.
- Y prediction conditioned on data up to t-1Same as one-step-ahead.

type="fitted.ytT"

For `type="fitted.ytT"`, `tidy.marssMLE` returns the analogous information for the Y part of the MARSS equation for an I.I.D. NEW DATA SET y'. The expected value and variance of y' is conditioned on the data you did observe y. It is important to note that y' is independent and identical (meaning i.i.d. in a statistical sense) to y except it has no missing values. Do not plot your observed data on these intervals. You need residuals intervals in that case. See `augment.marssMLE` for those.

The expected value of a new data set $\mathbf{Y'}_t$ conditioned on the observed data $\mathbf{Y} = \mathbf{y}(\mathbf{1}:\mathbf{T})$, is $Enewy = Z_t\hat{x}_t + D_t d_t + a_t$, where $\hat{x}_t$ is the expected value of $\mathbf{X}_t$ conditioned on the data up to T. The variance of a new data set $\mathbf{Y'}_t$ conditioned on the observed data 1:T, is $var.newy = R_t + Z_t\hat{V}_t Z_t^\top$, where $\hat{V}_t$ is the variance of $\mathbf{X}_t$ conditioned on the data up to T. The variance of the expected value of the new data set $\mathbf{Y'}_t$ is $var.Enewy = Z_t\hat{V}_t Z_t^\top$.

We compute the prediction interval for y', an interval that will cover the new data for alpha/2 percent of new data sets. The equation for the $\alpha/2$ confidence interval is (`qnorm(alpha/2)*sd.newy + Enewy`), where `sd.newy` is the square root of the diagonal of $var.newy$. The confidence interval for the expected value of y' is `qnorm(alpha/2)*se.Enewy + Enewy`, where `se.Enewy` is the square root of the diagonal of $var.Enewy$.

type="ytT" for missing data estimation

This returns the expected value and variance of $\mathbf{Y}_t$ (the data set you DID observe) 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 `tidy.marssMLE` for the missing values takes into account all the information in the data, specifically the correlation with other data at time t if R is not diagonal. Do not use `type="fitted.ytT"` for interpolating missing data as those 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 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 $E[Y(t)]$ conditioned on the data while the latter uses variance of $Y(t)$ 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 Y(t) conditioned on the observed data (in the section 'Computing the expectations in the update equations').

Parameter uncertainty Currently the intervals calculations for the states and observations use the point estimates of the model parameters and thus solve the intervals for the 'known' parameters case.

Value

A data.frame with estimates, sample standard errors, and confidence (or prediction) intervals.

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), ]
MLEobj <- MARSS(dat)

library(broom)

# A data frame of the estimated parameters
tidy(MLEobj)

# Make a plot of the estimated states
library(ggplot2)
d <- tidy(MLEobj, type = "xtT")
ggplot(data = d) +
  geom_line(aes(t, estimate)) +
  geom_ribbon(aes(x = t, ymin = conf.low, ymax = conf.high), linetype = 2, alpha = 0.3) +
```

```
  facet_grid(~.rownames) +
  xlab("Time Step") + ylab("Count")

# Make a plot of the estimates for the missing values
library(ggplot2)
d <- tidy(MLEobj, type = "ytT")
ggplot(data = d) +
  geom_point(aes(t, estimate)) +
  geom_line(aes(t, estimate)) +
  geom_point(aes(t, y), color = "blue") +
  geom_ribbon(aes(x = t, ymin = conf.low, ymax = conf.high), alpha = 0.3) +
  geom_line(aes(t, pred.low), linetype = 2) +
  geom_line(aes(t, pred.high), linetype = 2) +
  facet_grid(~.rownames) +
  xlab("Time Step") + ylab("Count") +
  ggtitle("Blue=data, Black=estimate, grey=CI, dash=prediction interval")

# Make a plot of the fitted y(t), i.e., to 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)
# Use augment() for model residuals (data - fitted values)
library(ggplot2)
d <- tidy(MLEobj, type = "fitted.ytT")
ggplot(data = d) +
  geom_line(aes(t, estimate), size=1) +
  geom_point(aes(t, y), color = "blue") +
  geom_ribbon(aes(x = t, ymin = conf.low, ymax = conf.high), alpha = 0.3) +
  geom_line(aes(t, pred.low), linetype = 2) +
  geom_line(aes(t, pred.high), 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")
```

`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"), 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"), 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 or tex. If blank, both are output.
replace	Replace existing file if present.
simplify	If TRUE, then if B or 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.

eli(dot)holmes(at)noaa(dot)gov

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)
```

zscore *z-score a vector or matrix*

Description

Removes the mean and standardizes the variance to 1.

Usage

```
zscore(x)
```

Arguments

x	n x T matrix of numbers

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

$$\Sigma$$

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.

eli(dot)holmes(at)noaa(dot)gov, eric(dot)ward(at)noaa(dot)gov

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