Package 'MARSS'

November 2, 2018

Type Package

Title Multivariate Autoregressive State-Space Modeling

Version 3.10.10

Date 2018-10-30

Depends R (>= 3.1.0)

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

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

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 2018-11-02 05:20:03 UTC

R topics documented:

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

Description

The MARSS package fits constrained and unconstrained multivariate autoregressive time-series models to multivariate time series data. To open the user guide from the command line, type `RShowDoc("UserGuide",package="MARSS")`. To open an overview page with package information and links to the R scripts in the User Guide, type `RShowDoc("index",package="MARSS")`.

The main function is `MARSS` which is used to fit a specified model to data and estimate the model parameters. MARSS model specification is based on "form" (an argument to a MARSS() call). The form tells MARSS() what to expect in the model list (model is a MARSS argument) and how to translate that into the base model form used in the fitting algorithms.

The default MARSS model form is "marxss", which is a model of the following form:

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

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

x(1) ~ MVN(x0, V0) or x(0) ~ MVN(x0, V0)

The parameters, hidden state processes (x), and observations (y) are matrices:

- x(t) is m x 1
- y(t) is n x 1 (m<=n)
- Z is n x m
- B is m x m
- U is m x 1
- Q is g x g (typically m x m)
- G is m x g
- A is n x 1
- R is h x h (typically n x n)
- H is n x h

- C is m x q
- D is n x p
- c(t) is q x 1
- d(t) is q x 1
- x0 is m x 1
- V0 is m x m

All parameters can be time-varying.

The package functions estimate the model parameters using an EM algorithm (primarily but see MARSSoptim). Parameters may be constrained to have shared elements (elements which are constrained to have the same value) or fixed elements (with the other elements estimated). The states and smoothed state estimates are provided via a Kalman filter and smoother. Bootstrapping, confidence interval estimation, bias estimation, model selection and simulation functions are provided. The main user interface to the package is the top-level function MARSS.

Details

Important MARSS functions:

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

MARSSsimulate Produces simulated data from a MARSS model.

MARSSkem Estimates MARSS parameters using an EM algorithm.

MARSSkf Kalman filter and smoother.

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

MARSSaic Calculates AICc, AICc, and various bootstrap AICs.

MARSSboot Creates bootstrap MARSS parameter estimates.

MARSSparamCIs Computes confidence intervals for maximum-likelihood estimates of MARSS parameters.

Author(s)

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

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

kellie(dot)wills(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.

Type `RShowDoc("index",package="MARSS")` to see all the package documentation, tutorials, and R scripts from the User Guide.

allowed *MARSS Function Defaults and Allowed Methods*

Description

Defaults and allowed fitting methods for the MARSS function are specified in the file MARSSsettings.R. These are hidden thus to see them preface with MARSS:::.

Details

allowed.methods is a vector with the allowed method arguments for the MARSS function. kem.methods and optim.methods are vectors of method arguments that fall in each of these two categories; used by MARSS. alldefaults is a list that specifies the defaults for MARSS arguments if they are not passed in.

augment.marssMLE *Return the model predicted values, residuals, and optionally confidence intervals*

Description

Return a data.frame with the observations or states fitted values, residuals, and upper and lower confidence intervals if requested.

Usage

```
augment.marssMLE(x, type.predict = c("observations", "states"),
                              interval = c("none", "confidence"),
                            conf.level = 0.95, form=attr(x[["model"]], "form"))
```

Arguments

x	a marssMLE object
type.predict	Type of prediction: for the observations (y) or for the states (x). Read the details below for states. tidy would be the more common function for returning state estimates.
interval	Type of interval: none or confidence interval. If the latter, 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`.

type.predict observations

This returns a familiar model predicted value of the response (y) and the difference between the model prediction and the actual data y(t) 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 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 augment 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, one.step.ahead=TRUE)`. The innovations are returned by `MARSSkf`.

type.predict states

The states are estimated. 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` to return the estimate of the state.

`augment(MLEobj, type.predict="states")` returns a model prediction of $\hat{x}(t)$ given $\hat{x}(t-1)$. The residuals returned are for w(t), the difference between $\hat{x}(t)$ and the prediction of $\hat{x}(t)$. 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.

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 the estimated states
# Don't use augment.  States are not data.
d <- tidy(MLEobj, type="states")
ggplot(data = d) +
  geom_line(aes(t, estimate)) +
```

```
geom_ribbon(aes(x=t, ymin=conf.low, ymax=conf.high), linetype=2, alpha=0.1) +
facet_grid(~term) +
xlab("Time Step") + ylab("Count")
```

---

checkMARSSInputs *Check inputs to MARSS call*

Description

This is a helper function to check the inputs to a MARSS() call for any errors. Not exported.

Usage

```
checkMARSSInputs( MARSS.inputs, silent=FALSE )
```

Arguments

MARSS.inputs	A list comprised of the needed inputs to a MARSS call: data, inits, model, control, method, form)
silent	Suppresses printing of progress bars, error messages, warnings and convergence information.

Details

This is a helper function to check that all the inputs to a MARSS call are properly specified.

If arguments inits or control are not provided by the user, they will be set by the alldefaults[[method]] object specified in MARSSsettings. Argument model specifies the model structure using a list of matrices; see MARSS or the User Guide for instructions on how to specify model structure. If model is left off, then the function MARSS.form() is used to determine the default model structure.

Value

If the function does not stop due to errors, it returns an updated list with elements

data	Data supplied by user.
model	Not changed. Will be updated by the MARSS.form function (e.g. MARSS.marxss).
inits	A list specifying initial values for parameters to be used at iteration 1 in iterative maximum-likelihood algorithms.
method	The method used for estimation.
form	The equation form used to convert wrapper object to a marssMODEL object.
control	See Arguments.

Author(s)

Kellie Wills, NOAA, Seattle, USA.

kellie(dot)wills(at)noaa(dot)gov

See Also

MARSS marssMODEL checkModelList

---

checkModelList *Check model List Passed into MARSS Call*

Description

This is a helper function to check the model list passed in to a MARSS() call for any errors. Not exported.

Usage

```
checkModelList( model, defaults, this.form.allows)
```

Arguments

model	A list from which a marssMODEL model will be constructed.
defaults	A list with the defaults for the elements in the model list in case the user leaves out any.
this.form.allows	A list of what inputs are allowed for each element in the model list.

Details

This is a helper function to check that all the model list that will be passed to a MARSS.form function to make the marssMODEL object. If elements in the list are left off, they will be filled in by defaults.

Value

If the function does not stop due to errors, it returns an updated model list with missing elements filled in by the defaults.

Author(s)

Eli Holmes, NOAA, Seattle, USA.

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

See Also

MARSS marssMODEL checkModelList

coef.marssMLE *Coefficient function for MARSS MLE objects*

Description

The MARSS fitting function, MARSS, outputs marssMLE objects. coef(marssMLE), where marssMLE is one's output from a MARSS call, will print out the estimated parameters. The default output is a list with the estimated parameters for each MARSS parameter, however coef 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, coef uses the model form specified in the call to MARSS to determine how to display the coefficients. This information is in attr(marssMLE$model,"form"), however you can specify a different form. The default form is "marxss" which is a MARSS model with inputs, see MARSS.marxss. However, the MARSS EM algorithm transforms the model into form "marss"; see MARSS.marss. The marss form of the model is stored (in marssMLE$marss). You can look at the coefficients in marss form by passing in form="marss" (this will return marssMLE$par). This is mainly useful is for debugging numerical problems since the error reports will be for the model in 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)
```

`fitted.marssMLE` *fitted function for MARSS MLE objects*

Description

The MARSS fitting function, MARSS, outputs marssMLE objects. `fitted(marssMLE)`, where marssMLE is the output from a MARSS call, will return the modeled value of y(t) or x(t). See details.

Usage

```
  ## S3 method for class 'marssMLE'
fitted(object, ..., one.step.ahead=FALSE, type=c("observations", "states"))
```

Arguments

`object`	A marssMLE object.
`...`	Other arguments, not used.
`one.step.ahead`	If TRUE, expected value of the states is computed conditioned on the data up to t-1
`type`	Fitted values for the observations or the states.

Details

observation fitted values

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

$\hat{x}(t)$ is the expected value of the states at time t. If `one.step.ahead=FALSE`, $\hat{x}(t)$ is the expected value conditioned on all the data, i.e. xtT returned by MARSSkf. If `one.step.ahead=TRUE`, $\hat{x}(t)$ is the expected value conditioned on 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.

state fitted values

The model predicted $\hat{x}(t)$ given $\hat{x}(t-1)$ is $B(t)\hat{x}(t-1)+u(t)$, where the model is written in marss form.

$\hat{x}(t)$ is the expected value of the states at time t. If `one.step.ahead=FALSE`, $\hat{x}(t)$ is the expected value conditioned on all the data, i.e. xtT returned by `MARSSkf`. If `one.step.ahead=TRUE`, $\hat{x}(t)$ is the expected value conditioned on the data up to time t-1, i.e. xtt1 returned by `MARSSkf`. 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, which is want one typically wants not the fitted values, then you'll want either the states estimate conditioned on all the data or conditioned on the data up to time t-1. These are returned by `MARSSkf` in `xtT` and `xtt1` respectively. Which one you want depends on your objective and application. You can also use the `tidy.marssMLE` function to return a data.frame with the xtT, standard errors and confidence intervals.

Value

A T column matrix of the fitted values with one row for each observation (or state) time series.

Author(s)

Eli Holmes, NOAA, Seattle, USA.

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

`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 .

AICbp Parametric bootstrap Akaike information criterion if .

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)
```

---

graywhales *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)
```

`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.marssMODEL` *Test Model Objects*

Description

These are model objects and utility functions for model objects in the package MARSS-package. Users would not normally work directly with these functions. `is.marssMODEL()` ensures model consistency. `MARSS_formname()` translates a model list as passed in call to MARSS() into a marssMODEL model object.

Usage

```
is.marssMODEL(MODELobj, method="kem")
```

Arguments

`MODELobj`	An object of class marssMODEL.
`method`	Method used for fitting in case there are special constraints for that method.

Details

A `marssMODEL` object is an R representation of a MARSS model along with the data. Data in a `marssMODEL` object consists of multivariate time series data in which time is across columns and the n observed time series are in the n different rows.

The base MARSS model (form=marss) is

x(t+1) = B(t) x(t) + U(t) + w(t), where w(t) ~ MVN(0,Q(t))

y(t) = Z(t) x(t) + A(t) + v(t), where v(t) ~ MVN(0,R(t))

x(1) ~ MVN(x0, V0)

The marssMODEL(form=marss) object describes this MARSS model but written in vec form:

x(t+1) = kron(x(t),I)(f_b(t)+D_b(t)b) + (f_u(t)+D_u(t)u) + w(t), where w(t) ~ MVN(0,Q)

vec(Q) = f_q(t)+D_q(t)q

y(t) = kron(x(t),I)(f_z(t)+D_z(t)z) + (f_a(t)+D_a(t)a) + v(t), where v(t) ~ MVN(0,R)

vec(R) = f_r(t)+D_r(t)r

x(1) ~ MVN(f_p+D_p p, V0)

vec(V0) = f_l+D_l l

In the marssMODEL(form=marss) object, f(t) + D(t)m, is the vec of a matrix M(t), so f_b(t)+D_b(t)b would be vec(B(t)). The estimated parameters are in the column vectors: b, u, q, z, a, r, p, and l. Each matrix M(t) is f(t)+D(t)m so is the sum of a fixed part f(t) and the linear combination, D(t), of the free (or estimated) part m.

The vec form of the MARSS model is specified by 3D matrices for each f and D for each parameter: B, U, Q, Z, A, R, x0, V0. The number of columns in the D matrix for a parameter determines the number of estimated values for that parameter. The first dimension for f (`fixed`) and D (`free`) must be:

Z n x m

B, Q, and V0 m x m

U and x0 m x 1

A n x 1

R n x n

The third dimension of f (`fixed`) and D (`free`) is either 1 (if not time-varying) or TT (if time-varying). The second dimension of f (`fixed`) is always 1, while the second dimension of D (`free`) depends on how many values are being estimated for a matrix. It can be 0 (if the matrix is fixed) or up to the size of the matrix (if all elements are being estimated).

Value

A vector of error messages or NULL is no errors.

Author(s)

Eli Holmes, NOAA, Seattle, USA.

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

See Also

`MARSS`, `MARSS.marxss`, `marssMODEL`

`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 averager 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)
```

MARSS *MARSS Model Specification and Estimation*

Description

This is the main MARSS function for fitting multivariate autoregressive state-space (MARSS) models. Scroll down to the bottom to see some short examples. To open the user guide from the command line, type `RShowDoc("UserGuide",package="MARSS")`. To open a guide to show you how to get started quickly, type `RShowDoc("Quick_Start",package="MARSS")`. To open an overview page with package information and how to find all the R code for the user guide chapters, type `RShowDoc("index",package="MARSS")`. To get info on the axillary functions (like for bootstrapping and confidence intervals) go to `MARSS-package`. To see a discussion of how to get output from your model fits, go to `print.MARSS`. 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 background section on this page is focused on fitting MARSS models in vectorized form. This form will almost certainly look unfamilar. MARSS works by converting the users' (more familiar model form) into the vectorized form which allows general linear constraints. You should go to the help page for the form of the model you are fitting to get background on that model form. Currently the MARSS package has two model forms: marxss and dfa.

`MARSS.marxss` This is the default form. X_t = B_t*X_t-1 + U_t + C_t*c_t + w_t; Y_t=Z_t*X_t + A_t + D_t*d_t + v_t. Any parameters can be set to zero. Most users will want this help page.

`MARSS.dfa` This is a model form to allow easier specification of models for Dynamic Factor Analysis. X_t = X_t-1 + U_t + w_t; Y_t=Z_t*X_t + A_t + D_t*d_t + v_t. The Z parameters has a specific form and the Q is set at i.i.d with variance of 1.

The rest of this help page discusses the vectorized form of a MARSS model.

The MARSS package fits time-varying state-space models that can be transformed into the form (termed form=marss):

x(t) = (t(x(t-1)) (x) I_m)(f_b(t)+D_b(t)beta) + (f_u(t)+D_u(t)upsilon) + w(t), where w(t) ~ MVN(0,Q(t))

y(t) = (t(x(t)) (x) I_n)(f_z(t)+D_z(t)zeta) + (f_a(t)+D_a(t)alpha) + v(t), where v(t) ~ MVN(0,R(t))

x(1) ~ MVN(x0, V0) or x(0) ~ MVN(x0, V0)

where beta, upsilon, zeta, and alpha are column vectors of estimated values, the f are column vectors of inputs, and the D are matrices of inputs. The f and D are potentially time-varying. (x) means kronecker product and I_p is a p x p identity matrix. The function `MARSS()` is used to fit MARSS models using the argument `form` to specify the type of state-space model being fit.

Most commonly used multivariate autoregressive state-space models can be reformulated into the form above. The user is not required to specify their model in the marss form (which is unfamiliar and unwieldy). Instead `MARSS()` uses the `form` argument to specify a more familiar state-space form. The user specifies their model in that (more familiar) form. `MARSS()` calls a helper function `MARSS_form` to translate the user's model into form=marss.

The default MARSS form is "marxss" which is the state-space model:

x(t) = B(t) x(t-1) + 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)

See `MARSS.marxss` for arguments and defaults information.

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 the `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()`.

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`).
	A n x T matrix of n time series over T time steps.
`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

MARSS provides an interface to the base `MARSS-package` functions and allows specification and fitting of MARSS models. The available estimation methods are maximum-likelihood via an EM algorithm (`method="kem"`) or via a quasi-Newton algorithm provided by function `optim` (`method="BFGS"`). The function `MARSS()` allows the user to specify models using the `model` argument. See `MARSS.marxss` for the format of the `model` argument for the default `marxss` form. See also the User Guide (reference and link below) or Quick Start Guide.

A call to `MARSS()` returns an object of class `marssMLE`. The MARSS package has print, coef, and residuals functions that will handle `marssMLE` objects. See `print.marssMLE`, `coef.marssMLE`, and `residuals.marssMLE`. The help page for `print.marssMLE` summarizes all the different output available for `marssMLE` objects and describes what the output is mathematically. Thus, this page is the first place to start and will direct you to the appropriate other method functions (like `coef`).

Many different types of multivariate time-series models can be converted to the MARSS form (see the User Guide). `MARSS()` allows the user to specify the form of their model using the argument `form`. The default form is a multivariate lag-1 (time-varying) state-space model which is denoted using `form="marxss"`. Look at `MARSS.marxss`) to see how the model argument for the marxss form is specified. The model argument is a list, but the elements of that and how it is converted to a marssMODEL(form=marss) object (needed for the internal algorithms) depends intimately on the equation form. Thus you will need to refer to that appropriate help page (`MARSS.formname`) for your equation form.

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) since 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.

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), go here `print.MARSS`.

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`, an 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. See `print.marssMLE` or `coef.marssMLE` for information on outputing the model estimates. This will be in form "marss". Use `print` or `coef` to output the estimated parameters in the form in the MARSS() call (e.g. the default "marxss" form).
`states`	The expected value of x conditioned on the data.
`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 isn't normally added to a marssMLE object since it is verbose, but can be computed using `MARSSkf(marssMLE)`.

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(marssMLE).

Author(s)

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

eli(dot)holmes(at)noaa(dot)gov, kellie(dot)wills(at)noaa(dot)gov

References

The 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 MARSS-package print.MARSS 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 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
kemfit

## 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.conversion` *Convert Model Objects between Forms*

Description

These are utility functions for model objects in the package MARSS-package. Users would not normally work directly with these functions.

Usage

```
marss_to_marxss(x, C.and.D.are.zero=FALSE)
marxss_to_marss(x, only.par = FALSE)
```

Arguments

`x`	An object of class marssMLE.
`C.and.D.are.zero`	If the C and D matrices are all 0, then a marss model can be converted to marxss without further information besides the marss model.
`only.par`	If only.par=TRUE then only the par element is changed and marss is used for the marss object.

Details

As the name of the functions imply, these convert marssMODEL objects of different forms into other forms. form=marss is the base form needed for the internal algorithms, thus other (more user friendly forms) must have a `form_to_marss` function to convert to the base form. The printing functions are customized to show output in the user-friendly form, thus a `marss_to_form` function is needed for `print` and `coef` methods for marssMLE objects.

Value

A marssMODEL object of the appropriate form.

Author(s)

Eli Holmes, NOAA, Seattle, USA.

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

See Also

marssMODEL

`MARSS.dfa` *Multivariate Dynamic Factor Analysis*

Description

The Dynamic Factor Analysis model in MARSS is

x(t+1) = x(t) + w(t), where w(t) ~ MVN(0,I)

y(t) = Z(t) x(t) + D(t)d(t) + v(t), where v(t) ~ MVN(0,R(t))

x(1) ~ MVN(0, 5*I)

Passing in `form="dfa"` to MARSS() invokes a helper function to create that model and creates the Z matrix for the user. Q is by definition identity, x0 is zero and V0 is diagonal with large variance (5). U is zero, A is zero, and covariates only enter the Y equation. Because U and A are 0, the data should have mean 0 (de-meaned) otherwise one is likely to be creating a structurally inadequate model (i.e. the model implies that the data have mean = 0, yet data do not have mean = 0).

Arguments

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

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

`demean` Logical. Default is TRUE, which means the data will be demeaned.

- `z.score` Logical. Default is TRUE, which means the data will be z-scored (demeaned and variance standardized to 1).

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

Details

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

- `B` "Identity". Can be "identity", "diagonal and equal", or "diagonal and unequal".
- `U` "Zero". Cannot be changed or passed in via model argument.
- `Q` "Identity". Can be "identity", "diagonal and equal", or "diagonal and unequal".
- `Z` Can be passed in as a (list) matrix if the user does not want a default DFA `Z` matrix. There are many equivalent ways to construct a DFA `Z` matrix. The default is Zuur et al.'s form (see User Guide).
- `A` Default="zero". Can be "unequal", "zero" or a matrix.
- `R` Default="diagonal and equal". Can be set to "identity", "zero", "unconstrained", "diagonal and unequal", "diagonal and equal", "equalvarcov", or a (list) matrix to specify general forms.

- `x0` Default="zero". Can be "unconstrained", "unequal", "zero", or a (list) matrix.
- `V0` Default=diagonal matrix with 5 on the diagonal. Can be "identity", "zero", or a matrix.
- `tinitx` Default=0. Can be 0 or 1. Tells MARSS whether x0 is at t=0 or t=1.
- `m` Default=1. Can be 1 to n (the number of y time-series). Must be integer.

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

Value

A object of class `marssMLE`. See `print.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.

Usage

```
MARSS(y, inits=NULL, model=NULL, miss.value=as.numeric(NA), method = "kem", form = "dfa",
```

Author(s)

Eli Holmes, NOAA, Seattle, USA.

References

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

See Also

`MARSS` `MARSS.marxss`

Examples

```
## Not run:
#See the Dynamic Factor Analysis chapter in the User Guide
RShowDoc("UserGuide",package="MARSS")

## End(Not run)
```

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

Description

The form of MARSS models for users is marxss, the MARSS models with inputs. See MARSS.marxss. In the internal algorithms, 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) = B(t) x(t-1) + 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(0) ~ MVN(x0, V0)

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 = "marxs
```

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) = B(t) x(t-1) + 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(0) ~ MVN(x0, V0)

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 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 = "marxs
```

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)
```

MARSSaic *AIC for MARSS Models*

Description

Calculates AIC, AICc, a parametric bootstrap AIC (AICbp) and a non-parametric bootstrap AIC (AICbb).

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"))
```

MARSSapplynames *Names for marssMLE Object Components*

Description

Puts names on the par, start, par.se, init components of `marssMLE` objects. This is a utility function in the `MARSS-package` and is not exported.

Usage

```
MARSSapplynames(MLEobj)
```

Arguments

`MLEobj`	An object of class `marssMLE`.

Details

The X.names and Y.names are attributes of marssMODEL objects (which would be in `$marss` and `$model` in the marssMLE object). These names are applied to the par elements in the marssMLE object.

Value

The object passed in, with row and column names on matrices as specified.

Author(s)

Eli Holmes, NOAA, Seattle, USA.

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

See Also

`marssMLE` `marssMODEL`

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 numerically estimating the Hessian matrix (the matrix of partial 2nd derivatives of the parameter estimates). 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

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
```

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

Description

Calculates the observed Fisher Information analytically via the recursion by Harvey (1989). This is the same as the Hessian of the 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. 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.

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 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

`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 for the EM algorithm. Users can also use `print( MLEobj, what="Ey")` to access this output. See `print.marssMLE`.

Usage

```
MARSShatyt( MLEobj )
```

Arguments

`MLEobj`	A `marssMLE` object with the `par` element of estimated parameters, `model` element with the model description and data.

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:TT) while y(2) is the unobserved data. y(1,1:t) is the observed data from time 1 to t.

`ytT`	Estimates E[Y(t) | Y(1,1:TT)=y(1,1:TT)] (n x T matrix).
`ytt1`	Estimates E[Y(t) | Y(1,1:t-1)=y(1,1:t-1)] (n x T matrix).
`OtT`	Estimates E[Y(t) t(Y(t) | Y(1)=y(1)] (n x n x T array).
`yxtT`	Estimates E[Y(t) t(X(t) | Y(1)=y(1)] (n x m x T array).
`yxt1T`	Estimates E[Y(t) t(X(t-1) | Y(1)=y(1)] (n x m x T array).
`yxttpT`	Estimates E[Y(t) t(X(t+1) | Y(1)=y(1)] (n x m x T array).
`errors`	Any error messages due to ill-conditioned matrices.
`ok`	(T/F) 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 Also

`MARSS` `marssMODEL` `MARSSkem`

`MARSShessian` *MARSS 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 log-likelihood function at the MLEs. It appends `$Hessian`, `$parMean`, `$parSigma` to the marssMLE object.

Usage

```
MARSShessian( MLEobj, method="Harvey1989" )
MARSSFisherI( MLEobj, method="Harvey1989" )
```

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

Method 'fdHess' uses fdHess from package nlme to numerically estimate the Hessian matrix (the matrix of partial 2nd derivatives of the parameter estimates). 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.

Hessian CIs are based on the asymptotic normality of ML estimates under a large-sample approximation.

MARSSFisherI() is a (non-exported) utility function called by MARSShessian(). It calls either MARSShessian.numerical or MARSSharveyobsFI to return the Hessian based on the value of method.

Value

MARSShessian() attaches Hessian, parMean and parSigma to the MLE object.

Author(s)

Eli Holmes, NOAA, Seattle, USA.

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

See Also

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

MARSS 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="fdHess")
```

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 parameter estimates). 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 *Information for 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

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 MARSS and look at the help file for their model form.

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 specifying 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.

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 a base function in the MARSS-package. 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).
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 *Maximum Likelihood Estimation for Multivariate Autoregressive State-Space 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 used 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, it generally means 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 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`

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)
```

MARSSkemcheck *Model Checking for MLE objects Passed to MARSSkem*

Description

This is a helper function in the MARSS-package that checks that the model can be handled by the MARSSkem algorithm. It also returns the structure of the model as a list of text strings.

Usage

```
MARSSkemcheck( MLEobj )
```

Arguments

MLEobj	An object of class marssMLE.

Value

A list with of the model elements A, B, Q, R, U, x0, Z, V0 specifying the structure of the model using text strings).

Author(s)

Eli Holmes, NOAA, Seattle, USA.

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

See Also

marssMODEL MARSSkem

MARSSkf *Kalman Filtering and Smoothing for Time-varying MARSS models*

Description

Provides Kalman filter and smoother output for MARSS models with (or without) time-varying parameters. This is a base function in the MARSS-package. MARSSkf is a small helper function to select which Kalman filter/smoother function to use based on which function was requested (in MLEobj$fun.kf). The default function is MARSSkfas.

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 KFAS 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 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 traditional 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. R package version 0.9.11. 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 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

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
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 <- MARSSkfas( MLEobj )
```

---

marssMLE *Maximum Likelihood MARSS Estimation Object*

Description

An 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 hessian. 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 to extract output.

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 8 matrices Z, A, R, B, U, Q, x0, V0, 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)

Kellie Wills, NOAA, Seattle, USA.

kellie(dot)wills(at)noaa(dot)gov

See Also

`marssMODEL` `MARSSkem`

marssMLE-class *Class "marssMLE"*

Description

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

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, NOAA, Seattle, USA

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

Kellie Wills, NOAA, Seattle, USA.

kellie(dot)wills(at)noaa(dot)gov

marssMODEL *Class "marssMODEL"*

Description

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

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. If called with marssMLE, 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")`: ...

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 via the `optim` function. The `MARSSoptim` function is called when `MARSS` is called with `method="BFGS"`. This is a base 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` 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, 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 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. If this is the case, using V0=0 and estimating the x0 elements (with control$kf.x0="x10") can often help.

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 what 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 Hessian matrix (the matrix of partial 2nd derivatives of the parameter estimates) 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. Using a Hessian approximation for variances is approximate. Bootstrap estimates of parameter bias are reported if method "parametric" or "innovations" is specified.

Note, these are added to the par (etc) 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.

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
```

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

Arguments

MLEobj	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.
`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)
```

MARSSvectorizeparam *Vectorize or Replace the par List*

Description

Converts `MLEobj[["what"]]` to a vector or assigns a vector to `MLEobj[["what"]]`. This is a utility function in the `MARSS-package` for marssMODEL objects of form="marss" and is not exported. Users achieve this functionality with `coef.marssMLE`.

Usage

```
MARSSvectorizeparam(MLEobj, parvec = NA, what="par")
```

Arguments

`MLEobj`	An object of class `marssMLE`.
`parvec`	NA or a vector. See Value.
`what`	What part of the MLEobj is being replaced or vectorized. Need to be a par list.

Details

Utility function to generate parameter vectors for optimization functions, and to set `MLEobj[[what]]` using a vector of values. The function bases the unlisting and naming order on `names(MLEobj$marss$fixed)`. Appends matrix name to the row names in the par list.

Value

If parvec=NA, a vector of the elements of the `what` element. Otherwise, a `marssMLE` object with `MLEobj[["what"]]` set by parvec.

Author(s)

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

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

See Also

`marssMLE`

Examples

```
dat <- t(harborSealWA)
dat <- dat[2:4,]
kem <- MARSS(dat)
paramvec = MARSS:::MARSSvectorizeparam(kem)
paramvec
```

model.frame.marssMODEL *model.frame method for marssMLE and marssMODEL objects*

Description

`model.frame(marssMLE)` or `model.frame(marssMODEL)`, where marssMLE is the output from a `MARSS` call marssMODEL is the model element of a marssMLE object, will return a data.frame with the data (y) and inputs/covariates (c and d elements) for a MARXSS model. See `MARSS.marxss`. This is mainly a utility function to help with the `broom` functions (`tidy`, `augment` and `glance`).

Usage

```
  ## S3 method for class 'marssMODEL'
model.frame(formula, ...)
```

Arguments

`formula`	A marssMODEL object.
`...`	Other arguments not used.

Value

A data.frame witht the data and inputs (c and d) in a MARXSS model. See `MARSS.marxss`.

Author(s)

Eli Holmes, NOAA, Seattle, USA.

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

plankton *Plankton Data Sets*

Description

Example 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 *plotting functions for 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, ...)
  ## S3 method for class 'marssMLE'
autoplot(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, ...)
```

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)
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.
conf.level	Confidence level for CIs.
decorate	TRUE/FALSE. Add smoothing lines to residuals plots or qqline to qqplots.
...	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)
fit <- MARSS(t(harborSealWA[,-1]), model=list(Z=as.factor(c(1,1,1,1,2)), R="diagonal and equal"))
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)
```

---

print.marssMLE *Printing functions for MARSS MLE objects*

Description

The MARSS fitting function, MARSS, outputs marssMLE objects. `print(marssMLE)`, where marssMLE 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(marssMODEL)`, where `marssMODEL` 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 Standardized Residuals*

Description

Calculates the standardized (or auxiliary) residuals sensu Harvey, Koopman and Penzer (1998).

Usage

```
## S3 method for class 'marssMLE'
residuals(object,..., Harvey=FALSE, normalize=FALSE)
```

Arguments

`object`	An object of class `marssMLE`.
`...`	Not used.
`Harvey`	TRUE/FALSE
`normalize`	TRUE/FALSE

Details

This function returns the conditional expected value (mean) and variance of the model and state residuals. 'conditional' means in this context, conditioned on the observed data and a set of parameters. If there are no missing values in the data, E(Y|y)=y, and the model residual is the difference between y and the model prediction. If the data are missing, then the model residual is 0 because E(Y)=model prediction and the difference between these is 0. The variance however is non-zero.

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 difference in the algorithm only concerns model residuals and their variance for missing values. The residual variance for missing data is not normally used but is needed for leave-one-out analyses when a non-missing y(t) is removed and then its residual is compared the model residual variance at time t.

The residuals matrix (and se and variance) has a value for each time step. The residuals in column t is the residual associated with the data at time t and the process (x) transition from x(t) to x(t+1). In the MARSS package, the process equation is written $x(t) = f(x(t-1)) + w(t)$, and the residual in column t is then v(t) w(t+1). In many other texts, the process equation is written is written $x(t) = f(x(t-1)) + w(t-1)$ and the residual in column t is denoted v(t) w(t). Regardless, the meaning is the same; the state residual in column t is associated with the transition from x(t) to x(t+1) not the transition from x(t-1) to x(t).

model.residuals

The model residuals v_t are the difference between the data and the predicted data at time t:

$$v_t = y_t - Zx_t - a$$

In a state-space model, x_t is stochastic and the model residuals are a random variable. y_t is also stochastic, though often observed unlike x_t. The model residual random variable is:

$$V_t = Y_t - ZX_t - a$$

The unconditional mean and variance of V_t is 0 and R. This function (`residuals(MLEobj)`) returns the *conditional* mean and variance of V_t.

`model.residuals` is the expected value of V_t conditioned on the data and parameter set Θ (all the parameters including Z, a, B, u, R and Q).

$$E(V_t|y_1^T, \Theta) = E(Y_t|y_1^T, \Theta) - ZE(X_t|y_1^T, \Theta) - a$$

If there are no missing data, this becomes

$$E(V_t|y_1^T, \Theta) = y_t - ZE(X_t|y_1^T, \Theta) - a$$

y_t are data at time t and $E(X_t|y_1^T, \Theta)$ is the Kalman smoother estimate of the states x_t at time t, i.e. the expected value of the states conditioned on all the data and the parameter set Θ. Thus 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
```

The model.residuals for the missing data is 0 (in res2) because that is the expected value of V_t when data are missing.

`var.residuals` returned by the function is the conditional variance of the residuals. Rows 1 to n are the conditional variance of the model residuals. This is the variance of V_t conditioned on the data and the parameter set Θ. The unconditional variance (no data) would just be R. See Holmes 2014.

If `Harvey=TRUE`, there will be no variance calculation for the missing y. If `Harvey=FALSE`, the variance of the model residuals at the t with missing y are computed (via Holmes 2014). The interpretation is that although the data are missing, you can still imagine that these data exist and you need the variance of residuals. For example, if you are doing a leave-one-out cross-validation, the data exist and you need their variance because you are going to compute some diagnostics using the left-out data. For outlier diagnostics and shock detection, the variances for the missing values are not needed.

state.residuals

The state residuals 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:

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

Like the model residual, the state residual w_t is a random variable since x_t is a random variable:

$$W_t = X_t - BX_{t-1} - u$$

The unconditional mean and variance of W_t is 0 and Q. `residuals(MLEobj)` returns the *conditional* mean and variance of W_t.

`state.residuals` is the expected value of W_t conditioned on the data (all the data 1 to T) and parameter set Θ.

$$E(W_t|y_1^T, \Theta) = E(X_t|y_1^T, \Theta) - BE(X_{t-1}|y_1^T, \Theta) - u$$

Thus 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
```

The state residuals always exist since the expected value of the states exist without data and will be identical with `Harvey=TRUE` or `Harvey=FALSE`. Generally speaking, $E(W_t|y_1^T)$ is not 0 even if there are missing data. Note that the state residual at the last time step (T) will be NA because it is the residual associated with x(T) to 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.

standardized residuals

`residuals.marssMLE` will return the standardized residuals sensu Harvey et al. (1998) for you in `std.residuals` for outlier and shock detection. These are the model and state residuals scaled by the inverse square root of the missing values corrected variance of the residuals. Note the standardized model residuals are set to NA when there are missing data (if there is no data point, there is no model residual). The standardized state residuals however always exist since the expected value of the states exist without data.

The interpretation of the 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 transformed residuals can look rather non-intuitive.

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

$$y_t = Zx_t + a + v_t$$

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

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).

`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

`model.residuals`
: The smoothed model residuals E(V(t)|y(1:T),Theta), where Theta is the set of model parameters. Sometimes called the smoothations. This is different than the Kalman filter innovations which are E(V(t)|y(1:t-1),Theta).

`state.residuals`
: The smoothed state residuals E(X(t)|y(1:T))-E(X(t)|E(x(t-1)|y(1:T))).

`residuals`
: The model residuals as a (n+m) x TT matrix with `model.residuals` on top and `state.residuals` below. `model.residuals` is $\hat{(}\eta_t)$ on page 112 of Harvey, Koopman and Penzer (1998).

`var.residuals`
: The variance of the model residuals and state residuals as a (n+m) x (n+m) x TT matrix with the model residuals in rows 1 to n.

`std.residuals`
: The standardized model residuals as a (n+m) x TT matrix. This is `residuals` divided by the square root of `var.residuals`—although the matrix equivalent of that equation.

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. This report shows how to compute the residuals from the Kalman smoother output without using the Harvey et al. (1998) algorithm. The variance of the hatvt for the missing values can then be computed.

See Also

MARSSkem marssMLE

Examples

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

#state 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)
#standardize to variance of 1
state.resids2 <- (solve(t(chol(Q))) %*% state.resids2)
#compare the two
cbind(t(state.resids1[,-30]),t(state.resids2))

#standardized (by variance) model & state residuals
residuals(MLEobj)$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)
```

stdInnov *Standardized Innovations*

Description

Standardizes Kalman filter innovations. This is a helper function called by MARSSinnovationsboot in the MARSS-package. Not exported.

Usage

```
stdInnov(SIGMA, INNOV)
```

Arguments

SIGMA	n x n x T array of Kalman filter innovations variances. This is output from MARSSkf.
INNOV	n x T matrix of Kalman filter innovations. This is output from MARSSkf.

Details

n = number of observation (y) time series. T = number of time steps in the time series.

Value

n x T matrix of standardized innovations.

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

MARSSboot MARSSkf MARSSinnovationsboot

Examples

```
## Not run:
std.innovations <- stdInnov(kfList$Sigma, kfList$Innov)

## End(Not run)
```

`tidy.marssMLE` *Return estimated parameters and states with summary information*

Description

This returns a data.frame with the estimated parameters (or states) of a MARSS model with optionally standard errors and confidence intervals. This assembles information available via the print and coef functions into a data.frame that summarizes the estimates.

If conf.int=TRUE. For parameters, `MARSSparamCIs` will be run to add confidence intervals to the fitted 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`. For states, the approximate CIs using the standard deviation of the states is used to compute the confidence intervals (`qnorm(alpha/2)*se.fit + fitted`).

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
```

The tidy function is compatible with the `broom` package.

Usage

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

Arguments

`x`	a marssMLE object
`type`	Estimates for the parameters or for the states.
`conf.int`	Whether to include a confidence interval.
`conf.level`	Confidence level if interval is returned.
`form`	If you want the augment 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.
`...`	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).

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)

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

# Make a plot of the estimated states
# Don't use augment.  States are not data.
d <- tidy(MLEobj, type="states")
ggplot(data = d) +
  geom_line(aes(t, estimate)) +
  geom_ribbon(aes(x=t, ymin=conf.low, ymax=conf.high), linetype=2, alpha=0.1) +
  facet_grid(~term) +
  xlab("Time Step") + ylab("Count")
```

`utility.functions` *Matrix Utilities*

Description

Matrix utilities for MARSS functions in the MARSS-package. These are not exported but can be accessed using the `MARSS:::` prefix.

Usage

```
is.blockdiag(x)
is.validvarcov(x, method="kem")
is.identity(x, dim=NULL)
is.diagonal(x, na.rm=FALSE)
is.equaltri(x)
makediag(x, nrow=NA)
takediag(x)
is.design(x, strict=TRUE, dim=NULL, zero.rows.ok=FALSE, zero.cols.ok=FALSE)
is.fixed(x, by.row=FALSE)
is.identity(x, dim=NULL)
is.zero(x)
vec(x)
unvec(x, dim=NULL)
is.wholenumber(x, tol = .Machine$double.eps^0.5)
Imat(x)
rwishart(nu, V)
mystrsplit(x)
convert.model.mat(param.matrix)
```

```
fixed.free.to.formula(fixed,free,dim)
matrix.power(x, n)
sub3D(x,t=1)
pinv(x)
pcholinv(x)
pchol(x)
is.solvable(A,y=NULL)
all.equal.vector(x)
parmat(MLEobj, elem = c("B", "U", "Q", "Z", "A", "R", "x0", "V0", "G", "H", "L"),
       t = 1, dims = NULL, model.loc = "marss")
```

Arguments

x, A, y	A matrix (or vector for 'makediag' or string for 'mystrsplit'.).
na.rm	How to treat NAs in the block diag test.
dim, dims	Matrix dimensions. Some functions will take the vec of a matrix. In this case, the optional dim arg specifies the matrix dimensions.
fixed	A fixed matrix per the MARSS specification for fixed matrix syntax.
free	A free matrix per the MARSS specification for free matrix syntax.
nrow	Number of rows.
tol	Tolerance.
method	kem or BFGS. Used to add extra test for MARSSoptim().
t	The time index or third dimension of a 3D matrix
nu, V	Parameters of a Wishart distribution.
param.matrix	The list matrix version of a time-invariant MARSS model.
n	An interger for the power function.
zero.rows.ok, zero.cols.ok	Means the design matrix can have all zero rows or columns.
strict	Specifies whether the design matrix must be only 0s and 1s.
by.row	For is.fixed, reports whether is.fixed by row rather than for the whole matrix.
MLEobj	A marssMLE object.
elem	The parameter matrix of a marss model to return.
model.loc	Whether to use the marss or model marssMODEL in the marssMLE object.

Details

- is... tests for various matrix properties. isDiagonal() from the Matrix package is used to test numeric matrices for diagonality. is.diagonal() is only used to determine if list matrices (that combine numeric and character values) are diagonal. is.zero tests for near zeroness and give TRUE for is.zero((.5-.3)-(.3-.1)) unlike ==0.
- vec(x) creates a column vector from a matrix per the standard vec math function.
- unvec(c,dim) takes the vector c and creates a matrix with the specified dimensions.
- Imat(nrow) returns the identity matrix of dimension nrow.

- `fixed.free.to.formula` takes a fixed and free pair and constructs a list matrix (or array if time-varying) with formulas in each matrix element.
- `convert.model.mat` takes a list matrix with formulas in each element and converts to a fixed/free pair.
- `sub3D` returns a 2D matrix after subsetting a 3D matrix on the third (time) dimension. Ensures that R always returns a matrix.
- `mystrsplit` is a customized string splitter used by `convert.model.mat`.
- `rwishart` generates random draws from a wishart distribution.
- `matrix.power` is a faster way to get the n-th power of a matrix.
- `pinv` is the pseudoinverse based on singular value decomposition PInv=UD^+V' where a diagonal matrix with non-zero diagonal values of D (from svd) replaced with 1/D.
- `pcholinv` is the inverse based on the Cholsky decomposition but modified to allow 0s on the diagonal of x (with corresponding 0 row/column). These appear as 0 row/columns in the returned inverse.
- `pchol` returns the Cholsky decomposition but modified to allow 0s on the diagonal of x (with corresponding 0 row/column).
- `is.solvable` returns information on the solvability of the linear system y=Ax using the SVD decomposition.
- `all.equal.vector` tests if the all the elements in a vector, matrix, or array are all equal. Works on list matrices too.
- `parmat` constructs the parameter matrix with both the fixed and free values from the vectorized form in a marssMLE object. Users should use `coef(MLEobj)` (See `coef.marssMLE`).

Value

See above.

Author(s)

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

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

---

`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(rnorm(6),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