Package ‘MARSS’

February 19, 2015

**Type** Package

**Title** Multivariate Autoregressive State-Space Modeling

**Version** 3.9

**Date** 2013-3-15

**Depends** R (>= 2.15.0)

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

**Suggests** Hmisc, maps, xtable, stringr

**Author** Eli Holmes, Eric Ward, 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), confidence 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 http://faculty.washington.edu/eeholmes/workshops.shtml See the NEWS file for update information.

**License** GPL-2

**LazyData** yes

**BuildVignettes** yes

**NeedsCompilation** no

**Repository** CRAN

**Date/Publication** 2014-03-21 01:22:51
R topics documented:

MARSS-package ........................................ 3
allowed .................................................. 4
checkMARSSInputs ....................................... 5
checkModelList .......................................... 6
coe.marssMLE ........................................... 7
CSEGriskfigure ......................................... 8
CSEGtmufigure .......................................... 9
graywhales .............................................. 11
harborSeal ............................................... 12
is.marssMODEL .......................................... 13
isleRoyal .................................................. 15
loggerhead ............................................... 16
MARSS ..................................................... 17
marss.conversion ....................................... 23
MARSS.dfa ............................................... 24
MARSS.marxss .......................................... 26
MARSSaic ................................................ 29
MARSSapplynames ....................................... 31
MARSSboot ............................................... 31
MARSShatyt ............................................. 34
MARSShessian .......................................... 35
MARSSinfo ................................................ 36
MARSSinits ............................................. 37
MARSSinnovationsboot .................................. 38
MARSSkem ............................................... 40
MARSSkemcheck ........................................ 44
MARSSkf .................................................. 44
MARSSmcinit .......................................... 48
marssMLE .............................................. 50
marssMLE-class ....................................... 52
marssMODEL .......................................... 53
MARSSoptim ............................................ 55
MARSSparamCIs ......................................... 57
MARSSsimulate ......................................... 59
MARSSvectorizeparam .................................. 60
plankton ................................................ 61
predict.marssMLE ...................................... 63
print.marssMLE ........................................ 65
print.marssMODEL ...................................... 67
residuals.marssMLE .................................... 68
SalmonSurvCUI ......................................... 69
stdInnov ................................................ 70
utility.functions ....................................... 71

Index 74
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:

\[
x(t+1) = B x(t) + U + C c(t) + w(t), \quad \text{where } w(t) \sim \text{MVN}(0,Q)
\]
\[
y(t) = Z x(t) + A + D d(t) + v(t), \quad \text{where } v(t) \sim \text{MVN}(0,R)
\]
\[
x(1) \sim \text{MVN}(x0, V0) \text{ or } x(0) \sim \text{MVN}(x0, V0)
\]

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 m x m
- A is n x 1
- R is n x n
- 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


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

---

## 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.
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, MCBounds, 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, MCBounds, 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.
MCbounds        A list of the bounds for the Monte Carlo initialization routine.
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
checkModelList

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
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

```R
## 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" Returns the parameter matrix for that parameter with fixed values at their fixed values and the estimated values at their estimated values.
- **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$model, "form")`, however you can specify a different form. `form="marss"` should always work since this is the base model form in which the model objects are stored (in `marssMLE$marss`). This is mainly useful if you want to see your model transformed into `marss` form as seeing this can be helpful for debugging numerical problems.
- **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(holmes@noaa.gov)
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", "parametric", "innovations", or "none". See MARSSparamCIs.
CI.sim      Number of simulations for bootstrap confidence intervals (positive integer).

Details

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

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.


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

See Also

`MARSSboot marssMLE CSEGtmufigure`

Examples

```r
  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 Anal-
ysis. 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

```r
  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 case study.

**Author(s)**

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

eli(holmes(at)noaa(dot)gov

**References**


**See Also**

CSEGriskfigure

**Examples**

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

Example data sets for use in case studies 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

- 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 kestrellogged 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.


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**

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


Examples

```r
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 marss-MODEL model object.

**Usage**

```r
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.
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

\[ x(t+1) = B(t) x(t) + U(t) + w(t), \text{ where } w(t) \sim \text{MVN}(0,Q(t)) \]
\[ y(t) = Z(t) x(t) + A(t) + v(t), \text{ where } v(t) \sim \text{MVN}(0,R(t)) \]
\[ x(1) \sim \text{MVN}(x0, V0) \]

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

\[ x(t+1) = \text{kron}(x(t),I)(f_b(t)+D_b(t)b) + (f_u(t)+D_u(t)u) + w(t), \text{ where } w(t) \sim \text{MVN}(0,Q) \]
\[ \text{vec}(Q) = f_q(t)+D_q(t)q \]
\[ y(t) = \text{kron}(x(t),I)(f_z(t)+D_z(t)z) + (f_a(t)+D_a(t)a) + v(t), \text{ where } v(t) \sim \text{MVN}(0,R) \]
\[ \text{vec}(R) = f_r(t)+D_r(t)r \]
\[ x(1) \sim \text{MVN}(f_p+D_p p, V0) \]
\[ \text{vec}(V0) = f_l+D_l l \]

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:

\[ Z \ n x m \]
\[ B, Q, \text{ and } V0 \ m x m \]
\[ U \text{ and } x0 \ m x 1 \]
\[ A \ n x 1 \]
\[ R \ n x n \]

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
isleRoyal

See Also
MARSS, MARSS.marxss, marssMODEL

isleRoyal  Isle Royale Wolf and Moose Data

Description
Example data set for estimation of species interaction strengths. These are data on the number of wolves and moose on Isle Royal, Michigan. The data are unlogged. The covariate data are the average Jan-Feb, average Apr-May and average July-Sept temperature (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


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)
Description

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

Usage

```r
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 "loggerheadNoisy" 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

```r
str(loggerhead)
str(loggerheadNoisy)
```
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 unfamiliar. 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=marxss):

\[
x(t) = ((t(x(t-1)))(x) I_m)(f_b(t)+D_b(t)beta) + (f_u(t)+D_u(t)upsilon) + w(t), \text{ where } w(t) \sim MVN(0,Q(t))
\]

\[
y(t) = ((t(x(t)))(x) I_n)(f_z(t)+D_z(t)zeta) + (f_z(t)+D_z(t)alpha) + v(t), \text{ where } v(t) \sim MVN(0,R(t))
\]

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

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 \( p \times p \) 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:
\[ x(t) = B(t) \ x(t-1) + u(t) + C(t)c(t) + w(t), \text{ where } w(t) \sim \text{MVN}(0,Q(t)) \]
\[ y(t) = Z(t) \ x(t) + a(t) + D(t)d(t) + v(t), \text{ where } v(t) \sim \text{MVN}(0,R(t)) \]
\[ x(1) \sim \text{MVN}(x_0, V_0) \text{ or } x(0) \sim \text{MVN}(x_0, V_0) \]

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 \( \text{x} \) 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,**

- \texttt{inits=NULL,}
- \texttt{model=NULL,}
- \texttt{miss.value=as.numeric(NA),}
- \texttt{method = "kem",}
- \texttt{form = "marxss",}
- \texttt{fit=TRUE,}
- \texttt{silent = FALSE,}
- \texttt{control = NULL,}
- \texttt{Mbounds = NULL,}
- \texttt{fun.kf = "MARSSkfas",}

\texttt{...}**

### Arguments

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

- **y**: A \( n \times T \) matrix of \( n \) time series over \( T \) time steps.

- **inits**: A list with the same form as the list outputted by \texttt{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 (\texttt{method="kem"}) (see **MARSSkem**) and the BFGS algorithm (\texttt{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() 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.
- **MCInit** If TRUE, do a Monte Carlo search of the initial condition space. (T/F)
- **numInitSteps** Number of random initial value draws if MCInit=TRUE (ignored otherwise). (positive integer)
- **silent** 1 or TRUE, 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.
- **MCbounds** Bounds on the uniform distributions from which initial values will be drawn if MCInit=TRUE (ignored otherwise). The Q (and analogously R) random matri-
ces are created by a random draw from a wishart distribution where \(df=\text{bounds}[1]\) and \(S=\text{diag}(\text{bounds}[2],m)\).

**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 MARSSkss which is a native R implementation of the Kalman filter and smoother in Shumway and Stoffer. The KFAS filter is much faster. MARSSkfas modifies the input and output in order to output the lag-one covariance smoother needed for the EM algorithm (per page 321 in Shumway and Stoffer (2000).

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 (\(\text{method}="\text{ken}"\)) or via a quasi-Newton algorithm provided by function \text{optim}(\text{method}="\text{BFGS}"). The function \text{MARSS()} allows the user to specify models using the \text{model} argument. See \text{MARSS.marxss} for the format of the \text{model} argument for the default marxss form. See also the User Guide (reference and link below) or Quick Start Guide.

A call to \text{MARSS()} returns an object of class marssMLE. The MARSS package has print, coef, residuals, and predict functions that will handle marssMLE objects. See \text{print.marssMLE}, \text{coef.marssMLE}, \text{residuals.marssMLE}, and \text{predict.marssMLE}. The help page for \text{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). \text{MARSS()} allows the user to specify the form of their model using the argument \text{form}. The default form is a multivariate lag-1 (time-varying) state-space model which is denoted using \text{form}="marxss". Look at \text{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 (\text{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 the Monte Carlo initial conditions search using \text{MCInit=TRUE} in the \text{control} list. 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 \text{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, 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.
- **y.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


See Also

marssMLE MARSSkem MARSSoptim MARSS-package print.MARSS MARSS.marxss MARSS.dfa

Examples

```r
#harborSealWA is a n=5 matrix of logged population counts
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 #print with se's and CIs

#fit a model with 3 hidden states (default)
kemfit = MARSS(dat, silent=TRUE) #suppress printing
kemfit #print information on the marssMLE object

#fit a model with 3 correlated hidden states
# with one variance and one covariance
#maxit set low to speed up example, but more iterations are need 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 data
# 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

Description

The Dynamic Factor Analysis model in MARSS is

\[ x(t+1) = x(t) + w(t), \text{ where } w(t) \sim \text{MVN}(0,I) \]
\[ y(t) = Z(t) x(t) + D(t)d(t) + v(t), \text{ where } v(t) \sim \text{MVN}(0,R(t)) \]
\[ x(1) \sim \text{MVN}(0, 5*I) \]
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", "MCbounds", "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 necessary 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 (list) 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 using form="dfa".
Value

A object of class marxssMLE. See print.marxssMLE for a discussion of the various output available for marxssMLE objects (coefficients, residuals, Kalman filter and smoother output, imputed values for missing data, etc.). See MARSSsimulate for simulating from marxssMLE 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


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)

---

**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:

\[
x(t) = B(t) x(t-1) + u(t) + C(t)c(t) + w(t), \text{ where } w(t) \sim \text{MVN}(0,Q(t))
\]

\[
y(t) = Z(t) x(t) + a(t) + D(t)d(t) + v(t), \text{ where } v(t) \sim \text{MVN}(0,R(t))
\]

\[
x(0) \sim \text{MVN}(x0, V0)
\]

Note, marxss is a model form. A model form is defined by a collection of form functions discussed in marxssMODEL. 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", "MCbounds", "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, 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 unsolvable 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 m x m matrix to use a custom fixed matrix.

- **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 n x n matrix to use a custom fixed matrix.

- **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 specifica-
tion 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.

- **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(Z=1, B=1, U=0, Q=0.05, A=0, R=0.05, x0=-99, V0=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=NULL, trace=0, safe=FALSE, allow.degen=TRUE, min.degen.iter=50, degen.lim=1e-04, MCInit=FALSE, numInits = 500, numInitSteps = 10, min.iter.conv.test=15, conv.test.deltaT=9, conv.test.slope.tol= 0.5, boundsInits=list(B=c(0,1), U=c(-1,1), Q = c(sqrt(0.1),0.1,0.1), Z=c(0,1), A=c(-1,1), R = c(sqrt(0.1),0.1,0.1) ))

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.

### Usage

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

### Author(s)

Eli Holmes, NOAA, Seattle, USA.
MARSSaic

See Also

marssModel MARSS.dfa

Examples

```r
## 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**

```r
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 $\text{par}$ 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


See Also

MARSSboot

Examples

dat = t(harborSealWA)
dat = dat[2:3,]
kem = MARSS(dat, model=list(Z=matrix(1,2,1),
"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 $mas$ 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


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[2:4,]
# maxit set low to speed up the example
kem = MARSS(dat, model=list(U="equal",Q=diag(.01,3)),
   control=list(maxit=50))
hess.list = MARSSboot(kem, param.gen="hessian", nboot=5)
# (no missing values allowed)
boot.list = MARSSboot(kem, output="all", sim="innovations", nboot=5)

# Parametric bootstrap CIs for data with missing values
dat = t(harborSealWA)
dat = dat[2:4,]
kem = MARSS(dat, model=list(Q=diag(.01,3)),
   control=list(maxit=50))
boot.list = MARSSboot(kem, output="all", sim="parametric", nboot=5)
MARSShatyt  

Compute Expected Value of Y, Y, and YX

Description

Computes the expected value of random variables involving Y for the EM algorithm. This function is not exported. Users should 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.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>y1T</td>
<td>Estimates E[Y(t)</td>
</tr>
<tr>
<td>ytt1</td>
<td>Estimates E[Y(t)</td>
</tr>
<tr>
<td>01T</td>
<td>Estimates E[Y(t) t</td>
</tr>
<tr>
<td>yxtT</td>
<td>Estimates E[Y(t) t</td>
</tr>
<tr>
<td>yxt1T</td>
<td>Estimates E[Y(t) t</td>
</tr>
<tr>
<td>errors</td>
<td>Any error messages due to ill-conditioned matrices.</td>
</tr>
<tr>
<td>ok</td>
<td>(T/F) Whether errors were generated.</td>
</tr>
</tbody>
</table>

Author(s)

Eli Holmes, NOAA, Seattle, USA.

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

References


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. The variance-covariance parameters are transformed via a Cholesky decomposition so that the variance-covariance matrices remain positive definite. The function returns a marssMLE object in this TRANSFORMED form. It appends $Hessian, $parMean, $parSigma in this transformed state. This is a utility function in the MARSS-package.

Usage

MARSShessian(MLEobj)
MARSShessian.backtrans(MLEobj.hessian, par.hessian)

Arguments

MLEobj An object of class marssMLE. This object must have a $par element containing MLE parameter estimates from e.g. MARSSkem.
MLEobj.hessian An object of class marssMLE but with any variance-covariance matrices in their Cholesky decomposition form. This requires a marssMODEL with transformed fixed elements and transformed par element in the marssMLE object.
par.hessian A vector of parameter estimates where any variance matrix elements are in the Cholesky decomposed form.

Details

Uses fdHess from package nlme to numerically estimate the Hessian matrix (the matrix of partial 2nd derivatives of the parameter estimates). Hessian CIs are based on the asymptotic normality of ML estimates under a large-sample approximation.
Value

MARSShessian() returns a transformed marssMLE object passed in along with additional components: Hessian, gradient, parMean and parSigma computed by the MARSShessian function.

MARSShessian.backtrans(MLEobj.hessian, par.hessian) takes a transformed marssMLE object (as output by MARSShessian along with a parameter vector in transformed form) and returns a parameter vector in regular (non-transformed form).

Author(s)

Eli Holmes, NOAA, Seattle, USA.
eli(dot)holmes(at)noaa(dot)gov

See Also

MARSSparamCIs marssMLE

Examples

dat = t(harborSeal)
dat = dat[,c(2,11),]
MLEobj = MARSS(dat)
# variance terms are TRANSFORMED by the Cholesky decomposition
MLEobj.hessian = MARSShessian(MLEobj)

# show the approx Hessian, again variance terms are transformed
MLEobj.hessian$Hessian

# generate a (Cholesky transformed) 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)

# back transform so variance terms are non-transformed
MARSShessian.backtrans(MLEobj.hessian, hess.params)

MARSSinfo

Information for MARSS Error Messages and Warnings

Description

Prints out more information for MARSS error messages and warnings.

Usage

MARSSinfo(number)
**MARSSinits**

**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**

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

**Arguments**

MLEobj

An object of class **marssMLE**.

inits

A list of up to 8 values to construct starting matrices for each parameter in a MARSSmodel.

**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$elem` values will be overridden and replaced with the fixed value. Default is `array(0,dim=c(n or m,1)).`
• **elem** = Q, R, B
  A numeric vector or matrix. If length equals the length of the modelObj$fixed$elem then inits$elem will be constructed by array(inits$elem), dim = dim(modelObj$fixed$elem)). If length is 1 or equals m or n then inits$elem will be constructed into a diagonal matrix by the command diag(inits$elem, m or n). If elem is fixed in the model, any inits$elem values will be overridden and replaced with the fixed value. Default is diag(0,0.05,m or n) for Q and R. Default is diag(PNPUL,m or n) for Q and R. Default is diag(QL,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 8 matrices A, R, B, U, Q, x0, V0, Z, specifying initial values for parameters in a MARSS model.

Author(s)
Eli Holmes, NOAA, Seattle, USA.
eli(dot)holmes(at)noaa(dot)gov

See Also
marssMODEL MARSSkem MARSSoptim

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)
MARSSinnovationsboot

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


See Also

stdInnov MARSSparamCIs MARSSboot
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 scratch but are easier to construct using MARSS with MARSS(..., fit=FALSE).

Options for MARSSkem() may be set using MLEobj$control. The commonly used elements of control are 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.

Examples

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

MCInit  If TRUE, Monte Carlo initialization will be performed by MARSSmcinit.

numInits  Number of random initial value draws to be used with MARSSmcinit. Ignored if MCInit=FALSE.

numInitSteps  Maximum number of EM iterations for each random initial value draw to be used with MARSSmcinit. Ignored if MCInit=FALSE.

boundsInits  Length 6 list. Each component is a length 2 vector of bounds on the uniform distributions (for B, U, and A) from which initial values will be drawn to be used with MARSSmcinit(). For R and Q, the 2 bounds specify the df and S(=diag(bound[2],m)) for a wishart distribution. Ignored if MCInit=FALSE. See Examples.

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

Value

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

method  String "kem".

kf  Kalman filter output.

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

numIter  Number of iterations needed for convergence.

convergence  Did estimation converge successfully?

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

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

convergence=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 for your final fit initial parameter values be set using Monte Carlo initialization (MLEobj$control$MCInit = TRUE), particularly if the model is not a good fit to the data. This requires more computation time, but reduces the chance of the algorithm terminating at a local maximum and not reaching the true MLEs. For many models and for draft analyses, this is unnecessary, but answers should be checked using an initial conditions search before reporting final values.

MARSSkem() calls a Kalman filter/smoothed (MARSSk) 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 structurally inflexible model is inadequate (much lower likelihood).

Convergence testing is done via a combination of two tests. The first test (abstol test) is the test that the change in the absolute value of the log-likelihood from one iteration to another is less than some tolerance value (abstol). The second test (log-log test) is that the slope of a plot of the log of the parameter value or log-likelihood versus the log of the iteration number is less than some tolerance. Both of these must be met to generate the Success! parameters converged output. If you want to circumvent one of these tests, then set the tolerance for the unwanted test to be high. That will guarantee that that test is met before the convergence test you want to use is met. The tolerance for the abstol test is set by control$abstol and the tolerance for the log-log test is set by control$conv.test.slope.tol. Anything over 1 is huge for both of these.

Author(s)
Eli Holmes and Eric Ward, NOAA, Seattle, USA.
eli(holmes@noaa.gov, eric(ward@noaa.gov

References

See Also
MARSSmcinit, 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.
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/smouter 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 )
MARSSkf

Arguments

MLEobj  A marssMLE object with the par element of estimated parameters, marss element with the model description (in marss form) and data, and control element for the fitting algorithm specifications. control$debugkf specifies that detailed error reporting will be returned (only used by MARSSkf). model$diffuse=TRUE specifies that a diffuse prior be used (only used by MARSSkf). 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 MARSSkf. If set, only the log-likelihood is returned using the KFAS function logLik. This is much faster if only the log-likelihood is needed.

return.lag.one Used by MARSSkf. If set to FALSE, the smoothed lag-one covariance values are not returned (Vtt1T is set to NULL). This speeds up MARSSkf 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 MARSSkf. 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.

MARSSkf uses the (Fortran-based) Kalman filter and smoother function (KF5) 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. MARSSkf 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), MARSSkf 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 MARSSkf 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). "P" elements are called "R" 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.

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xtT</td>
<td>State first moment conditioned on y(1:T): E[x(t)</td>
</tr>
<tr>
<td>VtT</td>
<td>State variance conditioned on y(1:T): E[(x(t)-xtT(t))(x(t)-xtT(t))'</td>
</tr>
<tr>
<td>Vtt1T</td>
<td>State lag-one covariance E[(x(t)-xtT(t))(x(t-1)-xtT(t-1))'</td>
</tr>
<tr>
<td>x0T</td>
<td>Initial state estimate E[x(i)</td>
</tr>
<tr>
<td>V0T</td>
<td>Estimate of initial state covariance matrix E[x(i)x(i)'</td>
</tr>
<tr>
<td>J</td>
<td>(m x m x T) Kalman smoother output. Only for MARSSkfss. (ref S&amp;S eqn 6.49)</td>
</tr>
<tr>
<td>J0</td>
<td>J at init time (t=0 or t=1) (m x m x T). Kalman smoother output. Only for MARSSkfss.</td>
</tr>
</tbody>
</table>
xtt State first moment conditioned on $y(1:t)$: $E[x(t) \mid y(1:t)]$ (m x T). Kalman filter output. (S&S eqn 6.17 with $s=t$)

xtt1 State first moment conditioned on $y(1:t-1)$: $E[x(t) \mid y(1:t-1)]$ (m x T). Kalman filter output. (S&S eqn 6.17 with $s=t-1$)

Vtt State variance conditioned on $y(1:t)$: $E[(x(t)-xtt(t))(x(t)-xtt(t))' \mid y(1:t)]$ (m x m x T array). Kalman filter output. $P_{t\mid t}$ in S&S (S&S eqn 6.18 with $s=t$, $t1=t2=t$). State second moment $E[x(t)x(t)' \mid y(1:t)] = Vtt(t)+xtt(t)xtt(t)'$

Vtt1 State variance conditioned on $y(1:t-1)$: $E[(x(t)-xtt1(t))(x(t)-xtt1(t))' \mid y(1:t-1)]$ (m x m x T array). Kalman filter output. $P_{t\mid t}$ in S&S (S&S eqn 6.18 with $s=t-1$, $t1=t2=t$). State second moment $E[x(t)x(t)' \mid y(1:t-1)] = Vtt1(t)+xtt1(t)xtt1(t)'$

Kt Kalman gain (m x m x T). Kalman filter output (ref S&S eqn 6.23). Only for MARSSkfss.

Innov Innovations $y(t) - E[y(t) \mid y(1:t-1)]$ (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 $\log L(y(1:T) \mid \theta)$ (ref S&S eqn 6.62)

kfas_model The model in KFAS model form (class SSMmodel). Only for MARSSkfss.

effects Any error messages.

Author(s)

Eli Holmes, NOAA, Seattle, USA. eli(dot)holmes(at)noaa(dot)gov

References


MARSSmcinit

Monte Carlo Initialization

Description

Performs a Monte Carlo search for optimal initial conditions iterative maximization algorithms (MARSSkem and MARSSoptim). This is a utility function in the MARSS-package.

Usage

MARSSmcinit(MLEobj)

Arguments

MLEobj

An object of class marssMLE.

Details

It is recommended that initial parameter values be set using MARSSmcinit(), particularly if the model is not a good fit to the data. This requires more computation time, but reduces the chance of the algorithm terminating at a local maximum and not reaching the true MLEs.

Options for MARSSmcinit() may be set using MLEobj$control, as follows:

MLEobj$control$numInits Number of random initial value draws.

MLEobj$control$numInitSteps Maximum number of EM iterations for each random initial value draw.
MLEobj$control$boundsInits Length 6 list. Each component is a length 2 vector of bounds on the uniform distributions from which initial values will be drawn (for A, B, U, and Z). For R and Q, variance-covariance matrices are generated from a Wishart distribution with df=bound[1] and S=diag(bound[2],m). Note, random initial conditions are only used for parameters that are not fixed.

The default values for these are given in MARSSsettings.R and listed in MARSS.

Value

A list with 8 matrices Z, A, R, B, U, Q, x0, V0, specifying initial values for parameters for iteration 1 of the EM algorithm. Note the output is the initial values for a marssMODEL in marss form.

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


See Also

MARSSkem marssMLE MARSS

Examples

```r
## Not run:
# Note doing a Monte-Carlo search takes a long, long time
dat = t(harborSeal)
dat = dat[c(2,nrow(dat)),]
fit1=MARSS(dat, control=list(MCInit=TRUE))
fit1
# Show the inits that were used
fit1$start
# Try fewer initial start locations
# and different mean variance (0.1 instead of 1) for R and Q
cntl.list = list(MCInit=TRUE, numInits=10, numInitSteps = 10,
    boundsInits=list(Q=c(1,0.1),R=c(1,0.1)))
fit2=MARSS(dat, control=cntl.list)
fit2
# Show the inits that were used
fit2$start
# ignore the values for Z, B, and V0; those parameters are fixed

## End(Not run)
```
**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**

```r
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.
MCInit  If TRUE, do a Monte Carlo search of the initial condition space.
numInit  Number of random initial value draws if MCInit=TRUE (ignored otherwise).
numInitSteps  Number of EM iterations for each random initial value draw if MCInit=TRUE (ignored otherwise).
boundsInit  Bounds on the uniform distributions from which initial values will be drawn if MCInit=TRUE (ignored otherwise).
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.
yt  Expected values of the y. This is just y for non-missing y.
y.se  Standard errors on the y. This will be 0 for non-missing y.
kf  A list containing Kalman filter/smoothed 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.
marssMLE-class

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

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
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. neglogLik is a helper function for MARSSoptim that returns the negative log-likelihood given a vector of the estimated parameters and a marssMLE object. When possible, the Kalman filter and smoother functions from the KFAS R package are used.

Usage

MARSSoptim(MLEobj)
neglogLik(x, MLEobj)

Arguments

MLEobj An object of class marssMLE.

x An vector of the estimated parameters as output by coef(MLEobj,type="vector").

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$control$mcInit If TRUE, Monte Carlo initialization will be performed by MARSSmcinit.

MLEobj$control$numInits Number of random initial value draws to be used with MARSSmcinit. Ignored if control$MCInit=FALSE.

MLEobj$control$numInitSteps Maximum number of EM iterations for each random initial value draw to be used with MARSSmcinit. Ignored if control$MCInit=FALSE.

MLEobj$control$boundsInits Length 6 list. Each component is a length 2 vector of bounds on the uniform distributions from which initial values will be drawn to be used with MARSSmcinit(). Ignored if control$MCInit=FALSE. See Examples.

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. MLEobj$par$V0 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 MLEobj$par$V0 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 visa-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(holmes(at)noaa(dot)gov

See Also

MARSS MARSSkem marssMLE optim

Examples

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

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

MARSSparamCIs

Standard Errors, Confidence Intervals and Bias for MARSS Parameters
Description

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

Usage

MARSSparamCIs(MLEobj, method = "hessian", alpha = 0.05, nboot=1000)

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.

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 done on these matrices in their equivalent Cholesky decomposition form (see `marsshessian`). 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

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),
"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.
MARSSvectorizeparam

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

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

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 phosphorous 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 phosphorous was added. The phosphorous input rates for weeks with no sampling (and no experimental phosphorous input) have been filled with 0.1 in the "by week" data.

Usage

```
data(ivesDataLP)
data(ivesDataByWeek)
data(lakeWAplankton)
```

Format

The data are provided as a matrix with time running down the rows.

Source


- 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)
```
predict.marssMLE  

**Compute the Prediction Intervals, Expected Values, and Standard Errors for States (X) and Observation (Y) from MARSS fits**

**Description**

This function is still in beta development. Not exported in version 3.5. Computes the expected values of states (X component) and observations (Y component) for MARSS models fit with MARSS. A list of newdata, which includes the inputs (e.g. c and d in a marxss model) must be passed in if they are included in the model. Optionally data (Y) can be passed in in newdata. This can be the original data or new data and can have missing values. To simulate data, use `simulate.marssMLE`.

**Usage**

```r
## S3 method for class 'marssMLE'
predict(object, ..., 
  n.ahead=1, 
  t.start=NULL, 
  newdata=list(), 
  se.fit=TRUE, 
  nboot=1000, 
  param.gen="hessian", 
  verbose=FALSE, 
  prediction.intervals=TRUE)
```

**Arguments**

- `object`  
  A marssMLE object `marssMLE`.
- `...`  
  Not used.
- `n.ahead`  
  Number of time steps ahead to predict (starting from `t.start`).
- `t.start`  
  A what t to start the prediction relative to the start (t=1) of the data to which the model was fit. The default is to start at t=T+1, where T is the length of the data to which the model was fit. For example, if T=10, t.start=1 and n.ahead = 5, prediction is for time steps 1, 2, 3, 4, 5 so is not past the end of the data.
- `newdata`  
  Data (explanatory variables) to be used for prediction if included in the model.
- `se.fit`  
  Logical: should standard errors of prediction be returned?
- `prediction.intervals`  
  Logical. Whether to return prediction intervals. Calculation of prediction intervals can be slow (if `param.gen="MLE"`) and will exit with an error if the Hessian cannot be computed (if `param.gen="hessian"`).
- `param.gen`  
  "MLE" or "hessian". Default="hessian". The prediction interval calculation requires samples of the parameters from their estimated distribution. You can generate these via the estimated Hessian matrix (fast but assumes normality) or via parametric bootstrapped (terribly slow).
predict.marssMLE

*nboot* Number of bootstrapped parameter sequences to generate for the prediction interval calculation.

*verbose* If TRUE, the boot straps used for the prediction interval calculation are returned.

**Details**

`newdata` needs to be consistent with the model fitted. Thus if the model included inputs (in X or Y component) then these must also be included in `newdata` and have the proper size and form. If `newdata$data` is not passed in, then data (Y component) are assumed to be all missing and the expected values of the data (Ys) will be returned. The data is assumed to be all missing even if `t.start` is less than `T` and thus the prediction period overlaps the data to which the model was fit. If that data should be included in the predictions, then it must be passed in in `newdata`. The number of time steps in components of `newdata` must be consistent with `nt.ahead`.

Prediction of the state (X) component is via the Kalman smoother (i.e. conditioned on all the data, if there is any) using `MARSSkf`. Prediction of the observation (Y) component is via `MARSShatyt`, and is also smoothed (i.e. conditioned on all the data, if there is any). The standard errors of prediction exclude the uncertainty in the estimation of the model parameters. You could use bootstrapping via `MARSSboot` if you wanted to include that.

**Value**

- **E.x**
  A matrix of the expected values of the states (X) given `newdata`. This is the Kalman smoother `xtT` output using `newdata` as the input for data and any inputs. See `MARSSkf`.

- **E.y**
  A matrix of the expected values of the observations (Y) given `newdata`. This is `ytT` output using `newdata` as the input for data and any inputs. See `MARSShatyt`.

- **x.se**
  A matrix of the estimated standard errors of the X predictions.

- **y.se**
  A matrix of the estimated standard errors of the Y predictions.

- **newdata**
  The `newdata` list that was constructed from the inputs.

- **pred.MLEobj**
  The `marssMLE` object that was constructed with `newdata`. This can be passed to `MARSSkf` if the full Kalman filter and smoother output is needed.

**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`
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

```r
## S3 method for class 'marssMLE'
print(x, digits = max(3,getOption("digits")-4), ..., what="fit", form=NULL, silent=FALSE)
```

## Arguments

- `x` A marssMLE object.
- `digits` Number of digits for printing.
- `...` Other arguments for print.
- `what` What to print. Default is "fit". If you input what as a vector, print returns a list. See examples.
  - "model" The model parameters with names for the estimated parameters. The output is customized by the form of the model that was fit. This info is in `attr(x$model, "form")`.
  - "par" A list of only the estimated values in each matrix. Each model matrix has its own list element. Standard function: `coef(x)`
  - "start" or "inits" The values that the optimization algorithm was started at. Note, `x$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
  - "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 data conditioned on the smoothed states estimate at time t. Standard function: residuals(x)$model.residuals

• "state.residuals" The smoothed state residuals. E(x(t)|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

• 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/smisher 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 MARSSHaty for a discussion of these expectations. This output is not normally attached to the output from a MARSS() call--except yT 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(holmes(at)noaa(dot)gov

Examples

dat = t(harborSeal)
dat = dat[c(2,11),]
MLEobj = MARSS(dat)
print(MLEobj)
print.marssMODEL

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 summary information on the model.

Usage

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

Arguments

x                  A marssMODEL object.

...             Other arguments for print.

Value

A print out of information and invisibly a list with the structure of each parameter matrix. If you assign the print call to a value, then you can reference the output.

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 auxilliary) residuals sensu Harvey, Koopman and Penzer (1998).

Usage

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

Arguments

object  An object of class marssMLE.
...
Not used.

Details

Uses the algorithm on page 112 of Harvey, Koopman and Penzer (1998) to compute the standardized model residuals.

Value

A list with the following components

model.residuals  The smoothed model residuals y(t)-E(y(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 y(t)-E(y(t)|y(1:t-1),Theta).

state.residuals  The smoothed stated 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 model.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 as a (n+m) x (n+m) x TT matrix. This is var(hat(model.residuals)).
std.residuals  The standardized model residuals as a \((n+m) \times TT\) matrix. This is residuals divided by the square root of \(\text{var.residuals}\) — although the code is using the matrix equivalent of that equation.

**Author(s)**

Eli Holmes, NOAA, Seattle, USA.

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

**References**


**See Also**

MARSSkem marssMLE

**Examples**

```r
dat = t(harborSeal)
dat = dat[c(2,11),]
MLEobj = MARSS(dat)

#not standardized model residuals
residuals(MLEobj)$model.residuals

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

### 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)
```

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
utility.functions

References


See Also

MARSSboot MARSSkf MARSSinnovationsboot

Examples

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

## End(Not run)
```

---

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

```r
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)
is.solvable(A, y==NULL)
all.equal.vector(x)
parmat(MLEobj, elem = c("B", "U", "Q", "Z", "A", "R", "x0", "V0"),
   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 $P_{inv}=UD^+V'$ 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.

• **is.solveable** 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
Index

*Topic **classes**
  marssMLE-class, 52
  marssMODEL, 53

*Topic **datasets**
  graywhales, 11
  harborSeal, 12
  isleRoyal, 15
  loggerhead, 16
  plankton, 61
  SalmonSurvCUI, 69

*Topic **hplot**
  CSEGriskfigure, 8
  CSEGtmufigure, 9

*Topic **package**
  marssMODEL, 3
  all.equal.vector (utility.functions), 71
  convert.model.mat  
    (utility.functions), 71
  is.solvable (utility.functions), 71
  matrix.power (utility.functions), 71
  mystrsplit (utility.functions), 71
  parmat (utility.functions), 71
  pcholinv (utility.functions), 71
  pinv (utility.functions), 71
  sub3D (utility.functions), 71
  alldefaults (allowed), 4
  allowed, 4
  allowed.methods (allowed), 4
  checkMARSSInputs, 5
  checkModellist, 6, 6
  coef (coef.marssMLE), 7
  coef.marssMLE-method (marssMLE-class), 52
  coef.marssMLE, 7, 20, 21, 60, 65, 73
  CSEGriskFigure, 8, 10
  CSEGtmufigure, 9, 9
  describe.marssMODEL (print.marssMODEL), 67
  fdHess, 35
  fixed.free.to.formula  
    (utility.functions), 71
  graywhales, 11
  grouse (graywhales), 11
  harborSeal, 12
  harborSealWA (harborSeal), 12
  Imat (utility.functions), 71
  is.blockdiag (utility.functions), 71
  is.design (utility.functions), 71
  is.diagonal (utility.functions), 71
  is.equaltri (utility.functions), 71
  is.fixed (utility.functions), 71
  is.identity (utility.functions), 71
  is.marssMLE (marssMLE), 50
  is.marssMODEL, 13
  is.marssMODEL_dfa (is.marssMODEL), 13
  is.marssMODEL_marss (is.marssMODEL), 13
  is.marssMODEL_marxss (is.marssMODEL), 13
  is.validvarcov (utility.functions), 71
  is.wholenumber (utility.functions), 71
  is.zero (utility.functions), 71
  isleRoyal, 15
  ivesDataByWeek (plankton), 61
  ivesDataLP (plankton), 61
  kem.methods (allowed), 4
  kestrel (graywhales), 11
  KFAS, 18
  KFS, 46
  lakeWaplankton (plankton), 61
  lakeWaplanktonRaw (plankton), 61
  lakeWaplanktonTrans (plankton), 61
  loggerhead, 16
loggerheadNoisy (loggerhead), 16
logLik, 45

makediag (utility.functions), 71
MARSS, 3–7, 15, 17, 25–27, 33, 37, 39, 40, 48, 49, 55, 57, 59, 63, 65, 66
MARSS-package, 3
marss.conversion, 23
MARSS.dfa, 17, 18, 22, 24, 29
MARSS.marxss, 5, 15, 17, 18, 20, 22, 26, 26
marss_to_marxss (marss.conversion), 23
MARSSaic, 4, 26, 28, 29, 33
MARSSapplynames, 31
MARSSBoot, 4, 9, 26, 28, 30, 31, 38, 39, 59, 60, 64, 71
MARSSHatyt, 21, 34, 64, 66
MARSShessian, 35, 58, 59
MARSSinfo, 36
MARSSinit, 37
MARSSinit_dlm (MARSSinit), 37
MARSSinit_marss (MARSSinit), 37
MARSSinit_marxss (MARSSinit), 37
MARSSinnovationsboot, 38, 59, 70, 71
MARSSkem, 4, 18, 21, 22, 35, 37–39, 40, 44, 45, 48–50, 52, 57, 69
MARSSkemcheck, 44
MARSSkf, 4, 21, 41–43, 44, 56, 64–66, 70, 71
MARSSkfas (MARSSkf), 44
MARSSkfs (MARSSkf), 44
MARSSmcinit, 41, 43, 48, 55
marssMLE, 8, 9, 19–22, 24, 26, 28–37, 39–41, 43–45, 48–50, 50, 52, 55–61, 63–65, 68, 69
marssMLE-class, 52
marssMODEL, 6, 14, 15, 24, 26, 29, 31, 33, 35, 38, 44, 45, 48, 50, 52, 53, 60, 64, 67
MARSSoptim, 3, 4, 18, 19, 21, 22, 28, 37, 38, 43, 45, 48, 50, 55
MARSSparamCIs, 4, 8, 26, 28, 36, 39, 56, 57, 65, 66
MARSSsettings (MARSS), 17
MARSSsimulate, 4, 26, 28, 59
MARSSvectorizeparam, 60
marxss_to_marss (marss.conversion), 23

neglogLik (MARSSoptim), 55
nlme, 35

okanaganRedds (graywhales), 11

optim, 4, 19, 45, 50, 55–57
optim.methods (allowed), 4

plankton, 61
prairiechicken (graywhales), 11
predict (predict.marssMLE), 63
predict.marssMLE-method
  (marssMLE-class), 52
predict.marssMLE, 20, 63
print (print.marssMLE), 65
print.marssMLE-method (marssMLE-class), 52
print.marssMODEL-method (marssMODEL), 53
print.MARSS, 17, 21, 22
print.marssMLE, 20, 21, 26, 28, 34, 50, 58, 65
print.marssMODEL, 67

redstart (graywhales), 11
residuals (residuals.marssMLE), 68
residuals.marssMLE-method
  (marssMLE-class), 52
residuals.marssMLE, 20, 66, 68
rockfish (graywhales), 11
rwishart (utility.functions), 71

SalmonSurvCUI, 69
simulate.marssMLE-method
  (marssMLE-class), 52
simulate.marssMLE, 63
simulate.marssMLE (MARSSsimulate), 59
SSModel, 45, 47
stdInnov, 39, 70
summary.marssMLE-method
  (marssMLE-class), 52
summary.marssMODEL-method (marssMODEL), 53
takediag (utility.functions), 71
unvec (utility.functions), 71
utility.functions, 71
vec (utility.functions), 71

wilddogs (graywhales), 11