Package: marssTMB (via r-universe)

October 2, 2024

Title Fast fitting of MARSS models with TMB

Description Companion to the MARSS package. Fast fitting of MARSS models with TMB. See the MARSS documentation. All the model syntax and features are the same as for the MARSS package.

Version 0.0.14

Maintainer Elizabeth E. Holmes <eli.holmes@noaa.gov>

URL https://atsa-es.github.io/marssTMB, https://github.com/atsa-es/marssTMB

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

License GPL (>= 2)

Depends R (>= 3.5.0), MARSS (>= 3.11.5)

Imports TMB (>= 1.9.3), stats, utils

Suggests ggplot2, dplyr, tidyr, knitr, rmarkdown, testthat

LinkingTo TMB, Matrix

VignetteBuilder knitr

ByteCompile true

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.3

SystemRequirements GNU make

Remotes atsa-es/MARSS@*release

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

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

RemoteRef HEAD

RemoteSha 32308960f26d91d58036a35393013180f9c28dda

Contents

`dfaTMB` *Tim Cline's original code to Fit a DFA model with TMB.*

Description

This can be called to fit a DFA with his syntax, but generally no work should be done here.

Usage

```
dfaTMB(
  y,
  model = list(m = 1, R = "diagonal and equal"),
  inits = list(R = 0.05),
  EstCovar = FALSE,
  Covars = NULL,
  indivCovar = FALSE,
  Dmat = NULL,
  Dfac = NULL,
  EstSE = FALSE,
  silent = TRUE,
  fun.opt = c("nlminb", "optim", "nlminb+optim"),
  method = c("Nelder-Mead", "BFGS", "CG", "L-BFGS-B", "SANN", "Brent"),
  control = NULL,
  form = c("dfa", "marxss")
)
```

Arguments

`y`	Vector of observations n x T.
`model`	list with • R "diagonal and equal", "unconstrained", "diagonal and unequal" • m number of states (x)
`inits`	list of initial conditions
`EstCovar`	TRUE/FALSE
`Covars`	An optional matrix, dimensioned nD x T, where nD is the number of covariates

indivCovar	flag for structure of covariates
Dmat	D initial matrix
Dfac	What elements of D are fixed
EstSE	TRUE / FALSE, whether to return the Hessian from the TMB sdreport
silent	Show TMB output when fitting, defaults to TRUE
fun.opt	function to use for optimization: stats::nlminb() or stats::optim()
method	to pass to optim call; ignored for fun="nlminb"
control	a list with the control settings for the optimatization function. See details for the defaults.
form	The equation form used in the marssTMB() call. The default is "dfa".

Details

The control defaults for stats::nlminb() are iter.max = 2000 and eval.max = 2000. For stats::optim(), the defaults are reltol = 1e-12 and maxit = 2000.

Value

A list with Optimization, Estimates, Fits, and AIC

Author(s)

Tim Cline wrote most of this while a graduate student in the Fish 507 Time Series Analysis course. Eli Holmes later modified it to replicate the MARSS(x, form="dfa") model.

Examples

```
library(MARSS)
data(lakeWAplankton, package = "MARSS")
phytoplankton <- c("Cryptomonas", "Diatoms", "Greens", "Unicells", "Other.algae")
dat <- as.data.frame(lakeWAplanktonTrans) |>
  subset(Year >= 1980 & Year <= 1989) |>
  subset(select=phytoplankton) |>
  t() |>
  MARSS::zscore()

#fit with MARSS
m1.em <- MARSS(dat, model=list(R='unconstrained', m=1, tinitx=1), form='dfa', z.score=FALSE)
m1.tmb <- dfaTMB(dat, model=list(R='unconstrained', m=1))
c(m1.em$logLik, m1.tmb$logLik)
```

`estimate_marss` *Internal function: MARSS parameter estimation using TMB*

Description

This model is in the general "marss" vectorized form. The diagonals and offdiagonals of R and Q are split apart like Tim Cline did. In `estimate_marss2()`, I use instead the approach in `MARSS::MARSSoptim()` where I just use the `chol()` of these matrices. Technically there are 2 equal solutions since the diagonals appear as the square so -a and a are the same. But I have not observed that this affects the behavior of optim().

Usage

```
estimate_marss(
  MLEobj,
  method = c("TMB", "nlminb_TMB", "BFGS_TMB"),
  opt.control = NULL,
  ...
)
```

Arguments

`MLEobj`	A properly formatted MARSS model as output by `MARSS()`
`method`	Normally passed in as MLEobj$method, but allows user to pass in a new method if they want to use MLEobj with another method. Allowed values are "TMB", "nlminb.TMB", "BFGS.TMB".
`opt.control`	Normally this is passed in as MLEobj$control, but if the MLEobj was set up using a different method, then you will need to set the opt.control options. See details.
`...`	not used

Details

Minimal error checking is done in this function. Normal calling function is `MARSS::MARSS()` with `method="TMB"`.

The main thing this does is

- collapse the 3D fixed and free matrices into 2D
- separate out the diag and offdiag parameter elements of R and Q

Restrictions

- V0 fixed (not estimated)
- Q and R cannot be time-varying (at the moment)

opt.control is what is passed to the control argument in nlminb() or optim(). If you use MARSS(x, method="TMB"), this will be set to appropriate defaults which you can see via MLEobj$control. But if you call estimate_marss() with a MLEobj from a call such as MARSS(x, method="kem") (so not a TMB method), you will need to set opt.control if you want values different from the base defaults for those functions. Note as a shortcut for nlminb(), you can set both eval.max, iter.max to the same value with opt.control=list(maxit=1000). Note, if you pass in opt.control, this will replace all values currently in MLEobj$control that are associated with the optimizer function.

The defaults set in MARSS::MARSS() are

- nlminb: eval.max = 5000, iter.max = 5000 and trace = 0.
- optim: maxit = 5000 and trace = 0

All other controls for the optimization function are left at NULL.

Value

A list with the objective and optimization objects.

- obj is the raw output from the TMB::MakeADFun() call.
- op is the raw output from the optimization call (optim or nlminb). Note that the function is minimizing the negative log-likelihood so the sign will be opposite of the log-likelihood reported by MARSS()
- opt.control is the controls sent to the optimization function.
- method method used for optimization

Author(s)

Eli Holmes.

See Also

MARSS::MARSSoptim(), MARSS::MARSSkem()

Examples

```
library(MARSS)
data(lakeWAplankton, package = "MARSS")
phytoplankton <- c("Cryptomonas", "Diatoms", "Greens", "Unicells", "Other.algae")
dat <- as.data.frame(lakeWAplanktonTrans) |>
  subset(Year >= 1980 & Year <= 1989) |>
  subset(select=phytoplankton) |>
  t() |>
  MARSS::zscore()

# set-up the model
mod <- MARSS(dat, model=list(m=3, tinitx=1), form="dfa", fit=FALSE, silent=TRUE)
# fit
fit <- estimate_marss(mod)
```

`estimate_marss2` *Internal function: MARSS parameter estimation using TMB*

Description

This model is in the general "marss" vectorized form. In `estimate_marss2()`, I use the approach in `MARSS::MARSSoptim()` where I just use the `chol()` of these matrices. Technically there are 2 equal solutions since the diagonals appear as the square so -a and a are the same. But I have not observed that this affects the LL of optim() but it definitely seems to slow things down. In `estimate_marss()`, the diagonals and offdiagonals of R and Q are split apart like Tim Cline did. I had some problems with the splitting approach for some models with Q unconstrained, though now it seems fixed.

Usage

```
estimate_marss2(
  MLEobj,
  method = c("TMB", "nlminb_TMB", "BFGS_TMB"),
  opt.control = NULL,
  ...
)
```

Arguments

`MLEobj`	A properly formatted MARSS model as output by `MARSS()`
`method`	Normally passed in as MLEobj$method, but allows user to pass in a new method if they want to use MLEobj with another method. Allowed values are "TMB", "nlminb.TMB", "BFGS.TMB".
`opt.control`	Normally this is passed in as MLEobj$control, but if the MLEobj was set up using a different method, then you will need to set the opt.control options. See details.
`...`	not used

Details

Minimal error checking is done in this function. Normal calling function is `MARSS::MARSS()` with `method="TMB"`.

The main thing this does is

- collapse the 3D fixed and free matrices into 2D
- separate out the diag and offdiag parameter elements of R and Q

Restrictions

- V0 fixed (not estimated)
- Q and R cannot be time-varying (at the moment)

opt.control is what is passed to the control argument in nlminb() or optim(). If you use MARSS(x, method="TMB"), this will be set to appropriate defaults which you can see via MLEobj$control. But if you call estimate_marss() with a MLEobj from a call such as MARSS(x, method="kem") (so not a TMB method), you will need to set opt.control if you want values different from the base defaults for those functions. Note as a shortcut for nlminb(), you can set both eval.max, iter.max to the same value with opt.control=list(maxit=1000). Note, if you pass in opt.control, this will replace all values currently in MLEobj$control that are associated with the optimizer function.

The defaults set in MARSS::MARSS() are

- nlminb: eval.max = 5000, iter.max = 5000 and trace = 0.
- optim: maxit = 5000 and trace = 0

All other controls for the optimization function are left at NULL.

Value

A list with the objective and optimization objects.

- obj is the raw output from the TMB::MakeADFun() call.
- op is the raw output from the optimization call (optim or nlminb). Note that the function is minimizing the negative log-likelihood so the sign will be opposite of the log-likelihood reported by MARSS()
- opt.control is the controls sent to the optimization function.
- method method used for optimization

Author(s)

Eli Holmes.

See Also

MARSS::MARSSoptim(), MARSS::MARSSkem()

Examples

```
library(MARSS)
data(lakeWAplankton, package = "MARSS")
phytoplankton <- c("Cryptomonas", "Diatoms", "Greens", "Unicells", "Other.algae")
dat <- as.data.frame(lakeWAplanktonTrans) |>
  subset(Year >= 1980 & Year <= 1989) |>
  subset(select=phytoplankton) |>
  t() |>
  MARSS::zscore()

# set-up the model
mod <- MARSS(dat, model=list(m=3, tinitx=1), form="dfa", fit=FALSE, silent=TRUE)
# fit
fit <- estimate_marss(mod)
```

MARSSfit.TMB *MARSS parameter estimation using TMB*

Description

This takes a MARSS::marssMLE object (fitted or not) and estimates parameters. Note this is the TMB method for the MARSS::MARSSfit generic. The typical use would be to call as `MARSS(data, method="TMB")`.

Usage

```
## S3 method for class 'TMB'
MARSSfit(x, fun = 1, ...)
```

Arguments

x	A properly formatted MARSS::marssMLE object ready for fitting.
fun	A debugging option to switch between estimate_marss() and estimate_marss2()
...	not used

Details

Restrictions

- V0 fixed (not estimated)
- Q and R cannot be time-varying (at the moment)

`opt.control` is what is passed to the control argument in `nlminb()` or `optim()`. If you use `fit <- MARSS(data, method="TMB")`, this will be set to appropriate defaults which you can see via `fit$control`. But if you call `estimate_marss()` with a marssMLE object from a call such as `MARSS(data, method="kem")` (so not a TMB method), you will need to set `opt.control` if you want values different from the base defaults for those functions. Note as a shortcut for `nlminb()`, you can set both `eval.max`, `iter.max` to the same value with `opt.control=list(maxit=1000)`. Note, if you pass in `opt.control`, this will replace all values currently in `fit$control` that are associated with the optimizer function.

The defaults set in `MARSS::MARSS()` are

- `nlminb`: `eval.max` = `5000`, `iter.max` = `5000` and `trace` = `0`.
- `optim`: `maxit` = `5000` and `trace` = `0`

All other controls for the optimization function are left at NULL. You can set other controls in the call `MARSS(..., control=list(...))`.

Value

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

- `method`: From the call or argument method if user passed that in.
- `kf`: Kalman filter output.
- `iter.record`: If `x$control$trace = TRUE`, then this is the `$message` value from `stats::optim()` or `stats::nlminb()` plus the output from the `TMB::MakeADFun()` call and the output from the optimization function.
- `numIter`: Number of iterations needed for convergence.
- `convergence`: Did estimation converge successfully?
  - `convergence=0`: Converged in less than `x$control$maxit` iterations and no evidence of degenerate solution.
  - `convergence=3`: No convergence diagnostics were computed because all parameters were fixed thus no fitting required.
  - `convergence=-1`: No convergence diagnostics were computed because the MLE object was not fit (called with fit=FALSE). This is not a convergence error just information. There is not par element so no functions can be run with the object.
  - `convergence=1`: Maximum number of iterations `x$control$maxit` was reached before convergence.
  - For other convergence errors, see`stats::optim()` or `stats::nlminb()`.
- `logLik`: Log-likelihood.
- `states`: State estimates from the Kalman smoother.
- `states.se`: Confidence intervals based on state standard errors, see caption of Fig 6.3 (p. 337) in Shumway & Stoffer (2006).
- `errors`: Any error messages.

Author(s)

Eli Holmes.

See Also

`MARSS::MARSSoptim()`, `MARSS::MARSSkem()`

`marssTMB_CheckPackageVersions`

Check TMB and Matrix versions

Description

Helper function to provide instructions if there is a mis-match

Usage

```
marssTMB_CheckPackageVersions(silent = TRUE)
```

Arguments

silent	Default is TRUE. If FALSE, then gives the full instructions

---

var_to_cholvar *Return R, Q and V0 with free and par for chol of matrix*

Description

The free, fixed and par (and start) return the vec of the var-cov matrix M: fixed + free%*%p = vec(M) This function transforms the free, fixed, par and start so that they return the chol of the varcov matrix: vec(chol(M)).

Usage

```
var_to_cholvar(MLEobj)
```

Arguments

MLEobj	A properly formatted MARSS model as output by MARSS()

Details

Why does this function solve for the new par rather than just putting the chol values in? Because the user might have scrambled the order of the parameter list. I won't know which par correspond to which elements of the var-cov matrix. The free matrix is what does the sorting/permutation of the estimated parameters into their correct places. Also the matrix might be block-diagonal, partially fixed, etc.

Restrictions

- Q and R cannot be time-varying (at the moment)

Value

A new MARSS::marssMLE object with new fixed, free, par and start

Author(s)

Eli Holmes.

Examples

```
dat <- t(harborSealWA)[2:4, ]
fit <- MARSS(dat, model=list(Q="unconstrained"))
fitchol <- var_to_cholvar(fit)
Qchol = coef(fitchol, type="matrix")$Q
Q = coef(fit, type="matrix")$Q
Q
crossprod(Qchol)
```

```
Q = coef(fit, type="matrix", what="start")$Q
Qchol = coef(fitchol, type="matrix", what="start")$Q
Q
crossprod(Qchol)

fit <- MARSS(dat, model=list(Q=diag(0.1,3)+0.01), control=list(maxit=15))
fitchol <- var_to_cholvar(fit)
Qchol = coef(fitchol, type="matrix")$Q
Q = coef(fit, type="matrix")$Q
Q
crossprod(Qchol)
Q = coef(fit, type="matrix", what="start")$Q
Qchol = coef(fitchol, type="matrix", what="start")$Q
Q
crossprod(Qchol)
```

Index