Quick Start Guide

Contents

1 Summary

Put data (y) in a $n \times T$ matrix or ts/mts object.

Specify each of the parameters in a list for the `model` argument.

Fit with

```
fit <- MARSS(y, model=list(...), method=c("kem", "BFGS", "TMB"))
```

Choose one method from `c("kem", "BFGS", "TMB")`.

Specification of a properly constrained model with a unique solution is the responsibility of the user because the {MARSS} package has no way to tell if you have specified an insufficiently constrained model.

2 The MARSS model

The {MARSS} package fits multivariate autoregressive state-space (MARSS) models of the form:

$$\begin{gathered} \boldsymbol{x}_t = \mathbf{B}_t \boldsymbol{x}_{t-1} + \mathbf{U}_t + \mathbf{C}_t \mathbf{c}_t + \mathbf{G}_t \boldsymbol{w}_t, \text{ where } \boldsymbol{W}_t \sim \text{MVN}(0, \mathbf{Q}_t) \\ \boldsymbol{y}_t = \mathbf{Z}_t \boldsymbol{x}_t + \mathbf{A}_t + \mathbf{D}_t \mathbf{d}_t + \mathbf{H}_t \boldsymbol{v}_t, \text{ where } \boldsymbol{V}_t \sim \text{MVN}(0, \mathbf{R}_t) \\ \boldsymbol{X}_1 \sim \text{MVN}(\boldsymbol{\xi}, \boldsymbol{\Lambda}) \text{ or } \boldsymbol{X}_0 \sim \text{MVN}(\boldsymbol{\xi}, \boldsymbol{\Lambda}) \end{gathered} \tag{2.1}$$

c and **d** are inputs (aka, exogenous variables or covariates or indicator variables) and must have no missing values. The **R**, **Q** and **Λ** variances can can have zeros on the diagonal to specify partially deterministic systems. This allows you to write MAR(p) models in MARSS form for example. See the User Guide. The {MARSS} package is designed to handle linear constraints within the parameter matrices. Linear constraint means you can write the elements of the matrix as a linear equation of all the other elements. See section below on linear constraints.

Example: a mean-reverting random walk model with three observation time series:

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_t = \begin{bmatrix} b & 0 \\ 0 & b \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_{t-1} + \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}_t, \quad \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}_t \sim \text{MVN}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} q_{11} & q_{12} \\ q_{12} & q_{22} \end{bmatrix} \right) \tag{2.2}$$

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}_t = \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_t + \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}_t, \quad \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}_t \sim MVN\left(\begin{bmatrix} a_1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} r_{11} & 0 & 0 \\ 0 & r & 0 \\ 0 & 0 & r \end{bmatrix} \right) \tag{2.3}$$

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_0 \sim \text{MVN}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right) \tag{2.4}$$

2.1 Model specification

Model specification is via a list with the structure of each of the model parameters: **B**, **U**, **C**, **Q**, **Z**, **A**, **D**, **R**, $\boldsymbol{\xi}$, and$\boldsymbol{\Lambda}$. There is a one-to-one correspondence between the model in R and the math version of the model. The model written in matrix form is translated into equivalent matrices (or arrays if time-varying) in R code. Matrices that combine fixed and estimated values are specified using a list matrix with numerical values for fixed values and character names for the estimated values.

For 2.2, this would be:

```
B1 <- matrix(list("b",0,0,"b"),2,2)
U1 <- matrix(0,2,1)
Q1 <- matrix(c("q11","q12","q12","q22"),2,2)
Z1 <- matrix(c(1,0,1,1,1,0),3,2)
A1 <- matrix(list("a1",0,0),3,1)
R1 <- matrix(list("r11",0,0,0,"r",0,0,0,"r"),3,3)
pi1 <- matrix(0,2,1); V1=diag(1,2)
model.list <- list(B=B1,U=U1,Q=Q1,Z=Z1,A=A1,R=R1,x0=pi1,V0=V1,tinitx=0)
```

The `tinitx` element tells MARSS whether the initial state for x is at $t = 1$ (`tinitx=1`) or $t = 0$ (`tinitx=0`).

MARSS has a number of text shortcuts for common parameter forms, such as `"diagonal and unequal"`; see below for the possible shortcuts.

3 Data and fitting

3.1 Data

The data must be entered as a $n \times T$ matrix, or a ts (or mts) object or vector (which will be converted to a $n \times T$ matrix).

3.2 Fit call

The call to fit the model is.

```
fit <- MARSS(y, model=model.list)
```

See `?MARSS` for all other arguments. The common ones are `method` to change the fitting method from default of EM (slow). See below. `control` is used to pass in a list to control fitting, e.g. `control=list(maxit=1000)` to increase maximum iterations if the fit does not converge.

Example with simulated data:

```
library(MARSS)
set.seed(1234)
x <- rbind(arima.sim(n=50,list(ar=0.95), sd=0.4),
           arima.sim(n=50,list(ar=0.95), sd=.02))
```

```
y <- Z1 %*% x + matrix(rnorm(3*50,0,0.1), 3, 50)
fit <- MARSS(y, model=model.list, silent=TRUE)
tidy(fit)
```

```
##    term     estimate    std.error      conf.low      conf.up
## 1  A.a1 0.0184598413 0.0269628326 -0.0343863395 0.0713060221
## 2 R.r11 0.0188070225 0.0062216241  0.0066128633 0.0310011817
## 3   R.r 0.0115645650 0.0024433098  0.0067757658 0.0163533641
## 4   B.b 0.8605716318 0.0634069752  0.7362962441 0.9848470195
## 5 Q.q11 0.1301323970 0.0288070155  0.0736716842 0.1865931098
## 6 Q.q12 0.0020987299 0.0029632765 -0.0037091853 0.0079066452
## 7 Q.q22 0.0001353807 0.0003058603 -0.0004640944 0.0007348558
```

3.3 Different fitting methods

The EM algorithm in the {MARSS} package is in R and on top of that EM algorithms are famously slow. You can try `method="BFGS"` and see if that is faster. For some models, it will be much faster and for others slower. The companion package {marssTMB} allows you to fit these models with TMB and will be the fastest, often much faster. Definitely if you are doing Dynamic Factor Analysis or working with large data sets, you will want to install {marssTMB} so you can fit with TMB.

```
library(marssTMB)
fit1 <- MARSS(y, model=model.list, silent=TRUE)
fit2 <- MARSS(y, model=model.list, method="BFGS", silent=TRUE)
fit3 <- MARSS(y, model=model.list, method="TMB", silent=TRUE)
cat("logLL of the 3 fits:", c(fit1$logLik, fit2$logLik, fit3$logLik))
```

```
## logLL of the 3 fits: 30.75614 30.82143 28.691
```

`method="BFGS"` and `method="TMB"` are both using quasi-Newton methods to optimize and these can be sensitive to initial conditions. You can run EM a few iterations and run with BFGS, and this will guard against poor initial conditions issues.

```
fit1 <- MARSS(y, model=model.list, control = list(maxit=15))
fit2 <- MARSS(y, model=model.list, method="BFGS", inits = fit1)
```

3.4 Defaults for `model` list

Form of the model list is `list(B=..., U=...)` etc.

3.4.1 `form="marxss"`

For `form="marxss"` (the default), matrix names in the model list must be `B`, `U`, `C`, `c`, `Q`, `Z`, `A`, `D`, `d`, `R`, `x0` ($\boldsymbol{\xi}$), and `V0` ($\boldsymbol{\Lambda}$), just like in Equation 2.1. There are defaults each parameter and you can leave off matrix names and the defaults will be used. Type `?MARSS.marxss` additional information.

- `B="identity"` $m \times m$ identity matrix
- `U="unequal"` Each element in the $m \times 1$ matrix $\mathbf{U}$ is estimated and allowed to be different.
- `Q="diagonal and unequal"` $\mathbf{Q}$ is a diagonal matrix and each element on the diagonal is allowed to be different.
- `Z="identity"` $n \times n$ identity matrix thus in the default model each y is associated with one x.
- `A="scaling"` If $\mathbf{Z}$ is identity, $\mathbf{A}$ is zero. Otherwise, the first y associated with a x is set to 0 and the rest are estimated.
- `R="diagonal and equal"` $\mathbf{R}$ is a diagonal matrix and each element on the diagonal is the same.
- `x0="unequal"` Each element in the $m \times 1$ matrix $\boldsymbol{\xi}$ is estimated and allowed to be different.
- `V0="zero"` $\boldsymbol{\Lambda}$ is set to zero thus $\boldsymbol{x}_0$ is treated as an estimated parameter.

- `tinitx=0` The initial condition for $\boldsymbol{x}$ is set at $t = 0$.

3.4.2 `form="dfa"`

Special form for fitting DFA models. Pass in `form="dfa"` to the `MARSS()` call. Typically only these would be in the model list:

- `m=1` Number of factors.
- `R="diagonal and equal"` $\mathbf{R}$ is a diagonal matrix and each element on the diagonal is the same.
- `d="zero"` If there are p covariates, pass in as a $p \times T$ matrix.
- `D="unconstrained"` if covariates passed in.

Defaults.

- `Z` A special unconstrained matrix with the upper triangle (without the diagonal) set to zero.
- `Q="identity"` $\mathbf{Q}$ is a diagonal matrix and each element on the diagonal is allowed to be different.
- `x0="zero"` Each element in the $m \times 1$ matrix $\boldsymbol{\xi}$ is estimated and allowed to be different.
- `V0=diag(5,n)` $\mathbf{\Lambda}$ is set to a diagonal matrix with 5 on the diagonal.
- `tinitx=0` The initial condition for $\boldsymbol{x}$ is set at $t = 0$.
- `B="identity"` $m \times m$ identity matrix
- `U="zero"`
- `A="zero"`

4 Showing the model fits and getting the parameters

There are `plot`, `autoplot`, `print`, `summary`, `coef`, `fitted`, `residuals` and `predict` functions for marssMLE objects. `?print.MARSS` will show you how to get standard output from your fitted model objects and where that output is stored in the marssMLE object. Type `?coef.MARSS` to see the different formats for displaying the estimated parameters. To see plots of your states and fits plus diagnostic plots, use `plot(fit)` or, better, `ggplot2::autoplot(fit)`. For summaries of the residuals (model and state), use the `residuals` function. See `?residuals.marssMLE`. To produce predictions and forecasts from a MARSS model, see `?predict.marssMLE`.

5 Tips and Troubleshooting

5.1 Tips

Use `ggplot2::autoplot(fit)` (or `plot(fit)`) to see a series of standard plots and diagnostics for your model. Use `tidy(fit)` for parameter estimates or `coef(fit)`. Use `fitted(fit)` for model ($\boldsymbol{y}$) estimates and `tsSmooth(fit)` for states ($\boldsymbol{x}$) estimates. You can also use `fit$states` for the states.

Let's say you specified your model with some text short-cuts, like `Q="unconstrained"`, but you want the list matrix for for a next step. `a <- summary(fit$model)` returns that list (invisibly). Because the model argument of `MARSS()` will understand a list of list matrices, you can pass in `model=a` to specify the model. `MARSSkfas(fit, return.kfas.model=TRUE)` will return your model in {KFAS} format (class SSModel), thus you can use all the functions available in the {KFAS} package on your model.

5.2 Troubleshooting

Try `MARSSinfo()` if you get errors you don't understand or fitting is taking a long time to converge. When fitting a model with `MARSS()`, pass in `silent=2` to see what `MARSS()` is doing. This puts it in verbose mode. Use `fit=FALSE` to set up a model without fitting. Let's say you do `fit <- MARSS(..., fit=FALSE)`. Now you can do `summary(fit$model)` to see what `MARSS()` thinks you are trying to fit.

You can also try `toLatex(fit$model)` to make a LaTeX file and pdf version of your model (saved in the working directory). This loads the {Hmisc} package (and all its dependencies) and requires that you are able

to process LaTeX files.

6 More information and tutorials

Many example analyses can be found in the MARSS User Guide (pdf). In addition, recorded lectures and more examples on fitting multivariate models can be found at our course website and in the ATSA course eBook html.

The MARSS User Guide starts with some tutorials on MARSS models and walks through many examples showing how to write multivariate time-series models in MARSS form. The User Guide also has vignettes: how to write AR(p) models in state-space form, dynamic linear models (regression models where the regression parameters are AR(p)), multivariate regression models with regression parameters that are time-varying and enter the non-AR part of your model or the AR part, detecting breakpoints using state-space models, and dynamic factor analysis. All of these can be written in MARSS form. It also has a series of vignettes on analysis of multivariate biological data.

Background on the algorithms used in the {MARSS} package is included in the User Guide.

7 Shortcuts and all allowed model structures

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.

The model list elements can have the following values:

7.1 `Z`

Defaults to `"identity"`. Can be 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 $n \times m$ list matrix for general specification of both fixed and shared elements within the matrix. May also be specified as a numeric $n \times m$ matrix to use a custom fixed $\mathbf{Z}$. `"onestate"` gives a $n \times 1$ matrix of 1s. The text shortcuts all specify $n \times n$ matrices.

7.2 `B`

Defaults to `"identity"`. Can be 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 $m \times m$ matrix to use custom fixed $\mathbf{B}$, but in this case all the eigenvalues of $\mathbf{B}$ must fall in the unit circle.

7.3 `U` and `x0`

Defaults to `"unequal"`. Can be a text string, `"unconstrained"`, `"equal"`, `"unequal"` or `"zero"`. May be specified as a $m \times 1$ list matrix for general specification of both fixed and shared elements within the matrix. May also be specified as a numeric $m \times 1$ matrix to use a custom fixed $\mathbf{U}$ or $\boldsymbol{x}_0$.

7.4 `A`

Defaults to `"scaling"`. Can be 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 $n \times 1$ matrix to use a custom fixed $\mathbf{A}$. Care must be taken so that the model is not under-constrained and unsolvable model. The default, `"scaling"`, only applies to $\mathbf{Z}$ matrices that are design matrices (only 1s and 0s and all rows sum to 1). When a column in $\mathbf{Z}$ has multiple

1s, the first row in the **A** matrix associated with those **Z** rows is 0 and the other associated **A** rows have an estimated value. This treats **A** as an intercept where one intercept for each $\boldsymbol{x}$ (hidden state) is fixed at 0 and any other intercepts associated with that $\boldsymbol{x}$ have an estimated intercept. This ensures a solvable model when **Z** is a design matrix.

7.5 `Q`, `R` and `V0`

Can be 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 matrix to use a custom fixed matrix.

`V0` defaults to `"zero"`, which means that $\boldsymbol{x}_0$ is treated as an estimated parameter.

7.6 `D` and `C`

Defaults to `"zero"` or if no covariates, defaults to `"unconstrained"`. Can also be any of the options available for the variance matrices.

7.7 `d` and `c`

Defaults to `"zero"`. Numeric matrix. No missing values allowed. Must have 1 column or the same number of columns as the data, The numbers of rows in must match the corresponding **D** or **C** matrix.

7.8 `G` and `H`

Defaults to `"identity"`. Can be specified as a numeric matrix or array for time-varying cases. Size must match the corresponding **Q** or **R** matrix.

8 Covariates, Linear constraints and time-varying parameters

8.1 Covariates

Inputs, aka covariates, are in **c** and **d**. The are passed in via the model list and must be a numeric matrix (no missing values). **C** and **D** are the estimated parameters, aka covariate effects.

Let's say you have temperature data and you want to include a linear effect of temperature that is different for each $\boldsymbol{x}$ time series:

```
temp <- matrix(rnorm(50, seq(0,1,1/50), 0.1),nrow=1)
C1 <- matrix(c("temp1","temp2"),2,1)
model.list$C <- C1
model.list$c <- temp
```

Fit as normal:

```
fit <- MARSS(y, model=model.list, method="BFGS")
```

A seasonal effect can be easily included via sine/cosine pairs. The `fourier()` function in the {forecast} package simplifies this. You will need to make your data into a ts/mts object.

```
yts <- ts(t(y), frequency = 12) # requires time down the rows
fcov <- forecast::fourier(yts,1) |> t()
```

```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

If you want a factor effect, then you can use the `seasonaldummy()` function in {forecast}

```
yts <- ts(t(y), frequency = 12) # monthly data
mcov <- forecast::seasonaldummy(yts) |> t() # month factor
```

8.2 Linear constraints

Your model can have simple linear constraints within all the parameters except $\mathbf{Q}$, $\mathbf{R}$ and $\mathbf{\Lambda}$. For example $1 + 2a - 3b$ is a linear constraint. When entering this value for you matrix, you specify this as `"1+2*a+-3*b"`. NOTE: +'s join parts so `+-3*b` to specify $-3b$. Anything after `*` is a parameter. So `1*1` has a parameter called `"1"`. Example, let's change the $\mathbf{B}$ and $\mathbf{Q}$ matrices in the previous model to:

$$\mathbf{B} = \begin{bmatrix} b-0.1 & 0 \\ 0 & b+0.1 \end{bmatrix} \quad \mathbf{Q} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \mathbf{Z} = \begin{bmatrix} z_1 - z_2 & 2z_1 \\ 0 & z_1 \\ z_2 & 0 \end{bmatrix}$$

$\mathbf{Q}$ *is fixed because* $\mathbf{Z}$ *is estimated and estimating both creates a statistically confounded model (both scale the variance of* $\boldsymbol{x}$*).*

This would be specified as (notice `"1*z1+-1*z2"` for `z1-z2`):

```
B1 <- matrix(list("-0.1+1*b",0,0,"0.1+1*b"),2,2)
Q1 <- matrix(list(1,0,0,1),2,2)
Z1 <- matrix(list("1*z1+-1*z2",0,"z2","2*z1","z1",0),3,2)
model.list <- list(B=B1,U=U1,Q=Q1,Z=Z1,A=A1,R=R1,x0=pi1,V0=V1,tinitx=0)
```

Fit as usual:

```
fit <- MARSS(y, model=model.list, silent = TRUE)
```

8.3 Time-varying parameters

The default model form allows you to pass in a 3-D array for a time-varying parameter (T is the number of time-steps in your data and is the 3rd dimension in the array):

$$\begin{gathered} \boldsymbol{x}_t = \mathbf{B}_t \boldsymbol{x}_{t-1} + \mathbf{U}_t + \mathbf{C}_t \mathbf{c}_t + \mathbf{G}_t \boldsymbol{w}_t, \quad \boldsymbol{W}_t \sim \text{MVN}(0, \mathbf{Q}_t) \\ \boldsymbol{y}_t = \mathbf{Z}_t \boldsymbol{x}_t + \mathbf{A}_t + \mathbf{D}_t \mathbf{d}_t + \mathbf{H}_t \boldsymbol{v}_t, \quad \boldsymbol{V}_t \sim \text{MVN}(0, \mathbf{R}_t) \\ \boldsymbol{x}_{t_0} \sim \text{MVN}(\boldsymbol{\xi}, \mathbf{\Lambda}) \end{gathered} \tag{8.1}$$

Zeros are allowed on the diagonals of $\mathbf{Q}$, $\mathbf{R}$ and $\mathbf{\Lambda}$. NOTE(!!), the time indexing. Make sure you enter your arrays such that the correct parameter (or input) at time t lines up with $\boldsymbol{x}_t$; e.g., it is common for state equations to have $\mathbf{B}_{t-1}$ lined up with $\boldsymbol{x}_t$ so you might need to enter the $\mathbf{B}$ array such that your $\mathbf{B}_{t-1}$ is entered at `Bt[,,t]` in your R code.

The length of the 3rd dimension must be the same as your data. For example, say in your mean-reverting random walk model (the example on the first page) you wanted $\mathbf{B}(2,2)$ to be one value before $t = 20$ and another value after but $\mathbf{B}(1,1)$ to be time constant. You can pass in the following:

```
TT <- dim(y)[2]
B1 <- array(list(),dim=c(2,2,TT))
B1[,,1:20] <- matrix(list("b",0,0,"b_1"),2,2)
B1[,,21:TT] <- matrix(list("b",0,0,"b_2"),2,2)
```

Notice the specification is one-to-one to your $\mathbf{B}_t$ matrices on paper.