MARSS Quick Start Guide

The default MARSS model (`form="marxss"`) is written as follows:

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

c and **d** are inputs (aka, exogenous variables or covariates or indicator variables) and must have no missing values. They are not treated as 'data' in the likelihood but as inputs. In most cases, **G** and **H** are fixed (not estimated) and must have no missing values, but see the User Guide on situations when they can be estimated.

The MARSS package is designed to handle linear constraints within the parameter matrices: **B**, **u**, **C**, **Q**, **Z**, **a**, **D**, **R**, $\boldsymbol{\pi}$, and $\boldsymbol{\Lambda}$ (and in limited situations **G** and **H**). Linear constraint means you can write the elements of the matrix as a linear equation of all the other elements.

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

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_t = \begin{bmatrix} b & 0 \\ 0 & b \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_{t-1} + \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}_t, \quad \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}_t \sim \text{MVN}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} q_{11} & q_{12} \\ q_{12} & q_{22} \end{bmatrix} \right), \quad \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_0 \sim \text{MVN}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right)$$

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

To fit with MARSS, we translate this model into equivalent matrices (or arrays if time-varying) in R. 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.

```
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)
```

Try printing these out and you will see the one-to-one correspondence between the model in R and the math version of the model. For `form="marxss"` (the default), matrix names in the model list must be B, U, C, c, Q, Z, A, D, d, R, x0, and V0, just like in equation (1). The `tinitx` element tells MARSS whether the initial state for x is at $t = 1$ (`tinitx=1`) or $t = 0$ (`tinitx=0`). The data must be entered as a $n \times T$ matrix; a dataframe is not a matrix nor is a vector nor is a time-series object. MARSS has a number of text shortcuts for common parameter forms, such as "diagonal and unequal"; see the User Guide for the possible shortcuts. You can leave off matrix names and the defaults will be used. Type ?MARSS.marxss to see the defaults for `form="marxss"`.

The call to MARSS is

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

The `R`, `Q` and `V0` variances can be set to zero to specify partially deterministic systems. This allows you to write MAR-p models in MARSS form for example. See the User Guide for examples.

Linear constraints

Your model can have simple linear constraints within all the parameters except Q, R and V0. 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 +− for subtraction. Anything after '*' is a parameter. So '1*1' has a parameter called '"1"'. Example, let's change the **B** matrix and **Q** matrices in the previous model to:

$$\mathbf{B} = \begin{bmatrix} b - 0.1 & 0 \\ 0 & b + 0.1 \end{bmatrix} \quad \mathbf{Q} = \begin{bmatrix} q_{11} & 0 \\ 1 & 0 \end{bmatrix} \quad \mathbf{Z} = \begin{bmatrix} z_1 - z_2 & 2 * z_1 \\ 0 & z_1 \\ z_2 & 0 \end{bmatrix}$$

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("q11",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 with and best to call `toLatex()` on your model to make sure you and `MARSS()` agree on what model you a trying to fit:

```
fit <- MARSS(data, model=model.list)
toLatex(fit$model)
```

Important

- Specification of a properly constrained model with a unique solution is the responsibility of the user because MARSS has no way to tell if you have specified an insufficiently constrained model.
- The code in the MARSS package is not particularly fast and 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, much slower. "BFGS" can be notoriously sensitive to initial conditions. You can run EM a few iterations and then pass to "BFGS", and it will do better. ■marss.call3, eval=FALSE■= fit1 <- MARSS(data, model=model.list, control=list(minit=10, maxit=10)) fit2 <- MARSS(data, model=model.list, method="BFGS", inits=fit1)

Time-varying parameters and inputs

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

$$\begin{aligned} \mathbf{x}_t &= \mathbf{B}_t\mathbf{x}_{t-1} + \mathbf{u}_t + \mathbf{C}_t\mathbf{c}_t + \mathbf{G}_t\mathbf{w}_t, \quad \mathbf{W}_t \sim \text{MVN}(0, \mathbf{Q}_t) \\ \mathbf{y}_t &= \mathbf{Z}_t\mathbf{x}_t + \mathbf{a}_t + \mathbf{D}_t\mathbf{d}_t + \mathbf{H}_t\mathbf{v}_t, \quad \mathbf{V}_t \sim \text{MVN}(0, \mathbf{R}_t) \\ \mathbf{x}_{t_0} &\sim \text{MVN}(\boldsymbol{\pi}, \boldsymbol{\Lambda}) \end{aligned} \tag{2}$$

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

Inputs are specified in exactly the same manner. $\mathbf{C}$ and $\mathbf{D}$ are the estimated parameters and $\mathbf{c}$ and $\mathbf{d}$ are the inputs. Let's say you have temperature data and you want to include a linear effect of temperature that is different for each $\mathbf{x}$ time series:

```
C1 <- matrix(c("temp1","temp2"),2,1)
model.list <- list(B=B1,U=U1,C=C1,c=temp,Q=Q1,Z=Z1,A=A1,R=R1,x0=pi1,V0=V1,tinitx=0)
```

If you want a factor effect, then you'll need to recode your factor as a matrix with T columns and a row for each factor. Then you have 0 or 1 if that factor applies in time period t. $\mathbf{C}$ then has a column for each estimated factor effect. See the Covariate chapter in the user guide.

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 `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`.

Tips and Tricks

Use `plot(fit)` (or `autoplot(fit)`) to see a series of plots and diagnostics for your model. 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. Let's say you specified your model with some text short-cuts, like `Q="unconstrained"`, but you want the list matrix form 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 form (class SSModel), thus you can use all the functions available in the KFAS package on your model.

Need more information?

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 a series of

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 MARSS is included in the user guide. Lectures and more examples on fitting multivariate models can be found at `https://nwfsc-timeseries.github.io/`.