MARSS Package Manual

E. E. Holmes

2019-11-27

Contents

Preface

The **MARSS** R package allows you to fit **constrained** multivariate autoregressive state-space models.

This manual covers the **MARSS R** package: what it does, how to set up your models, how to structure your input, and how to get different types of output. For vignettes showing how to use MARSS models to analyze data, see the companion book *MARSS Modeling for Environmental Data* by Holmes, Scheurell, and Ward.

Installation

To install and load the **MARSS** package from CRAN:

```
install.packages("MARSS")
library(MARSS)
```

The latest release on GitHub may be ahead of the CRAN release. To install the latest release on GitHub:

```
install.packages("devtools")
library(devtools)
install_github("nwfsc-timeseries/MARSS@*release")
library(MARSS)
```

The master branch on GitHub is not a 'release'. It has work leading up to a GitHub release. The code here may be broken though usually preliminary work is done on a development branch. To install the master branch:

```
install_github("nwfsc-timeseries/MARSS")
```

If you are on a Windows machine and get an error saying 'loading failed for i386' or similar, then try

```
options(devtools.install.args = "--no-multiarch")
```

To install an **R** package from Github, you need to be able to build an **R** package on your machine. If you are on Windows, that means you will need to install Rtools. On a Mac, installation should work fine; you don't need to install anything.

Author

Elizabeth E. Holmes is a research scientist at the Northwest Fisheries Science Center (NWFSC) a US Federal government research center. This work was conducted as part of her job for NOAA Fisheries, as such the work is in the public domain and cannot be copyrighted.

Links to more code and publications can be found on our academic websites:

- http://faculty.washington.edu/eeholmes

Citation

Holmes, E. E. 2019. MARSS Manual. NOAA Fisheries, Northwest Fisheries Science Center, 2725 Montlake Blvd E., Seattle, WA 98112. Contact eli.holmes@noaa.gov.

Part 1. Overview

Here the **MARSS** package and MARSS models are briefly introduced. Part 2 shows a series of short examples and Part 3 goes into output from **MARSS** fitted objects and MARSS models in general.

Chapter 1

Overview

MARSS stands for Multivariate Auto-Regressive(1) State-Space. The **MARSS** package is an ,**R** package for estimating the parameters of linear MARSS models with Gaussian errors. This class of model is extremely important in the study of linear stochastic dynamical systems, and these models are important in many different fields, including economics, engineering, genetics, physics and ecology (Appendix **??**). The model class has different names in different fields, for example in some fields they are termed dynamic linear models (DLMs) or vector autoregressive (VAR) state-space models. The **MARSS** package allows you to easily fit time-varying constrained and unconstrained MARSS models with or without covariates to multivariate time-series data via maximum-likelihood using primarily an EM algorithm. The EM algorithm in the **MARSS** package allows you to apply linear constraints on all the parameters within the model matrices. Fitting via the BFGS algorithm is also provided in the package using ,**R**'s `optim` function, but this is not the focus of the **MARSS** package.

MARSS, `MARSS()` and MARSS. MARSS model refers to the class of models which the **MARSS** package fits using, primarily, an EM algorithm. In the text, **MARSS** refers to the ,**R** package. Within the package, the main fitting function is `MARSS()`. When the class of model is being discussed, rather than the package or the function, "MARSS" is used.

A full MARSS model, with Gaussian errors, takes the form:

$$\begin{aligned} \mathbf{x}_t &= \mathbf{B}_t\mathbf{x}_{t-1} + \mathbf{u}_t + \mathbf{C}_t\mathbf{c}_t + \mathbf{G}_t\mathbf{w}_t,\ \mathbf{w}_t \sim \text{MVN}(0, \mathbf{Q}_t) \\ \mathbf{y}_t &= \mathbf{Z}_t\mathbf{x}_t + \mathbf{a}_t + \mathbf{D}_t\mathbf{d}_t + \mathbf{H}_t\mathbf{v}_t,\ \mathbf{v}_t \sim \text{MVN}(0, \mathbf{R}_t) \\ \mathbf{x}_1 &\sim \text{MVN}(\boldsymbol{\pi}, \boldsymbol{\Lambda}) \text{ or } \mathbf{x}_0 \sim \text{MVN}(\boldsymbol{\pi}, \boldsymbol{\Lambda}) \end{aligned} \tag{1.1}$$

The $\mathbf{x}$ equation is termed the state process and the $\mathbf{y}$ equation is termed the observation process. Data enter the model as the $\mathbf{y}$; that is the $\mathbf{y}$ is treated as the data although there may be missing data. The $\mathbf{c}_t$ and $\mathbf{d}_t$ are inputs (aka, exogenous variables, covariates or indicator variables). The $\mathbf{G}_t$ and $\mathbf{H}_t$ are also typically inputs (fixed values with no missing values).

The bolded terms are matrices with the following definitions:

$\mathbf{x}$ is a $m \times T$ matrix of states. Each $\mathbf{x}_t$ is a realization of the random variable $\mathbf{X}_t$ at time t.

$\mathbf{w}$ is a $m \times T$ matrix of the process errors. The process errors at time t are multivariate normal with mean 0 and covariance matrix $\mathbf{Q}_t$.

$\mathbf{y}$ is a $n \times T$ matrix of the observations. Some observations may be missing.

$\mathbf{v}$ is a $n \times T$ column vector of the non-process errors. The observation erros at time t are multivariate normal with mean 0 and covariance matrix $\mathbf{R}_t$.

$\mathbf{B}_t$ **and** $\mathbf{Z}_t$ are parameters and are $m \times m$ and $n \times m$ matrices.

$\mathbf{u}_t$ **and** $\mathbf{a}_t$ are parameters and are $m \times 1$ and $n \times 1$ column vectors.

$\mathbf{Q}_t$ **and** $\mathbf{R}_t$ are parameters and are $g \times g$ (typically $m \times m$) and $h \times h$ (typically $n \times n$) variance-covariance matrices.

$\boldsymbol{\pi}$ is either a parameter or a fixed prior. It is a $m \times 1$ matrix.

$\boldsymbol{\Lambda}$ is either a parameter or a fixed prior. It is a $m \times m$ variance-covariance matrix.

$\mathbf{C}_t$ **and** $\mathbf{D}_t$ are parameters and are $m \times p$ and $n \times q$ matrices.

c and d are inputs (no missing values) and are $p \times T$ and $q \times T$ matrices.

$\mathbf{G}_t$ **and** $\mathbf{H}_t$ are inputs (no missing values) and are $m \times g$ and $n \times h$ matrices.

AR(p) models can be written in the above form by properly defining the $\mathbf{x}$ vector and setting some of the $\mathbf{R}$ variances to zero; see Chapter **??**. Although the model appears to only include i.i.d. errors ($\mathbf{v}_t$ and $\mathbf{w}_t$), in practice, AR(p) errors can be included by moving the error terms into the state model. Similarly, the model appears to have independent process ($\mathbf{v}_t$) and observation ($\mathbf{w}_t$) errors, however, in practice, these can be modeled as identical or correlated by using one of the state processes to model the errors with the $\mathbf{B}$ matrix set appropriately for AR or white noise—although one may have to fix many of the parameters associated with the errors to have an identifiable model. Study the application chapters and textbooks on MARSS models (Appendix **??**) for examples of how a wide variety of autoregressive models can be written in MARSS form.

1.1 Examples of MARSS models

Written in an unconstrained form, meaning all the elements in a parameter matrices are allowed to be different and none constrained to be equal or related, a MARSS model can be written out as follows. Two state processes ($\mathbf{x}$) and three observation processes ($\mathbf{y}$) are used here as an example.

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_t = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_{t-1} + \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}_t, \quad \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}_t \sim \text{MVN}\left(\begin{bmatrix} u_1 \\ u_2 \end{bmatrix}, \begin{bmatrix} q_{11} & q_{12} \\ q_{21} & q_{22} \end{bmatrix} \right)$$

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}_t = \begin{bmatrix} z_{11} & z_{12} \\ z_{21} & z_{22} \\ z_{31} & z_{32} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_t + \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}_t, \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}_t \sim \text{MVN}\left(\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}, \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \right)$$

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_0 \sim \text{MVN}\left(\begin{bmatrix} \pi_1 \\ \pi_2 \end{bmatrix}, \begin{bmatrix} \nu_{11} & \nu_{12} \\ \nu_{21} & \nu_{22} \end{bmatrix} \right) \quad \textit{or} \quad \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_1 \sim \text{MVN}\left(\begin{bmatrix} \pi_1 \\ \pi_2 \end{bmatrix}, \begin{bmatrix} \nu_{11} & \nu_{12} \\ \nu_{21} & \nu_{22} \end{bmatrix} \right)$$

However not all parameter elements can be estimated simultaneously. Constraints are required in order to specify a model with a unique solution. The MARSS package allows you to specify constraints by fixing elements in a parameter matrix or specifying that some elements are estimated—and have a

linear relationship to other elements. Here is an example of a MARSS model with fixed and estimated parameter elements:

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_t = \begin{bmatrix} a & 0 \\ 0 & a \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_{t-1} + \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}_t, \quad \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}_t \sim \text{MVN}\left(\begin{bmatrix} 0.1 \\ u \end{bmatrix}, \begin{bmatrix} q_{11} & q_{12} \\ q_{12} & q_{22} \end{bmatrix} \right)$$

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}_t = \begin{bmatrix} d & d \\ c & c \\ 1+2d+3c & 2+3d \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_t + \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}_t, \quad \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}_t \sim \text{MVN}\left(\begin{bmatrix} a_1 \\ a_2 \\ 0 \end{bmatrix}, \begin{bmatrix} r & 0 & 0 \\ 0 & r & 0 \\ 0 & 0 & r \end{bmatrix} \right)$$

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

Notice that some elements are fixed (in this case to 0, but could be any fixed number), some elements are shared (have the same value), and some elements are linear combinations of other estimated values (c, $1+2d+3c$ and $2+3d$ are linear combinations of c and d).

Chapter 2

How to get started (quickly)

If you already work with models in the form of Equation **??**, you can immediately fit your model with the **MARSS** package. Install the **MARSS** package and then type `library(MARSS)` at the command line to load the package. Look at the Quick Start Guide and then skim through Chapter **??** to, hopefully, find an example similar to your application. Appendix **??** also has many examples of how to specify different forms for your parameter matrices.

Chapter 3

Getting your data in right format

Your data need to be a matrix (not dataframe nor a **ts** object) with time across the columns ($n \times T$ matrix). The **MARSS** functions assume discrete time steps and you will need a column for each time step. Replace any missing time steps with NA.

A ,**R ts** object (time series object) stores information about the time steps of the data and often seasonal information (the quarter or month). **MARSS** needs this information in matrix form. If you have your data in **ts** form, then you may be using year and season (quarter, month) as covariates to estimate trend and seasonality. Here is how to get your **ts** into the form that MARSS wants.

Here is how to get your **ts** object into the form that MARSS wants.

3.1 Univariate ts object

This converts a univariate **ts** object with year and quarter into a matrix with a row for the response (here called `Temp`), year, and quarter.

```
z <- ts(rnorm(10), frequency = 4, start = c(1959, 2))
dat <- data.frame(Yr = floor(time(z) + .Machine$double.eps),
```

```
      Qtr = cycle(z), Temp=z)
dat <- t(dat)
class(dat)
```

Notice that the class of `dat` is matrix, which is what we want. There are three rows, first is the reponse and the second and third are the covariates, Year and Quarter. When you call MARSS, `dat["Temp",]` is the data. `dat[c("Yr","Qtr"),]` are your covariates.

3.2 Multivariate ts object

In this example, we have two temperature readings, our responses, and a salinity reading, a covariate. The data are monthly.

```
z <- ts(matrix(rnorm(300), 100, 3), start = c(1961, 1),
     frequency = 12, names=c("Temp1","Temp2","Sal"))
dat <- data.frame(Yr = floor(time(z) + .Machine$double.eps),
     Month = cycle(z), z)
dat <- t(dat)
```

When you call MARSS, `dat[c("Temp1","Temp2"),]` are the data and `dat[c("Yr","Month","Sal"),]` are your covariates.

See the *MARMES* chapters that discuss seasonality for examples of how to model season. The brute force method of treating month or quarter as a factor requires estimation of more parameters than is necessary in many cases.