E. E. Holmes, E. J. Ward, and M. D. Scheuerell

Analysis of multivariate time-series using the MARSS package

version 3.5

September 18, 2013

Northwest Fisheries Science Center, NOAA
Seattle, WA, USA

Holmes, E. E., E. J. Ward and M. D. Scheuerell 2012. Analysis of multivariate time-series using the MARSS package. NOAA Fisheries, Northwest Fisheries Science Center, 2725 Montlake Blvd E., Seattle, WA 98112. Contacts eli.holmes@noaa.gov, eric.ward@noaa.gov, and mark.scheuerell@noaa.gov

Disclaimer: E. E. Holmes, E. J. Ward, and M. D. Scheuerell are NOAA scientists employed by the U.S. National Marine Fisheries Service. The views and opinions presented here are solely those of the authors and do not necessarily represent those of our employer.

Preface

The initial motivation for our work with MARSS models was a collaboration with Rich Hinrichsen. Rich developed a framework for analysis of multi-site population count data using MARSS models and bootstrap AICb (Hinrichsen and Holmes, 2009). Our work (EEH and EJW) extended Rich's framework, made it more general, and led to the development of a parametric bootstrap AICb for MARSS models, which allows one to do model-selection using datasets with missing values (Ward et al., 2010; Holmes and Ward, 2010). Later, we developed additional algorithms for simulation and confidence intervals. Discussions with Mark Scheuerell led to an extensive revision of the EM algorithm and to the development of a general EM algorithm for constrained MARSS models (Holmes, 2012). Discussions with Mark also led to a complete rewrite of the model specification so that the package could be used for MARSS models in general—rather than simply the form of MARSS model used in our applications. Many collaborators have helped test the package; we thank especially Yasmin Lucero, Kevin See, and Brice Semmens. Development of the code into a R package would not have been possible without Kellie Wills, who wrote much of the code outside of the algorithm functions. Finally, we thank the participants of our MARSS workshops and MARSS users who have contacted us regarding issues that were unclear in the manual, errors, or suggestions regarding new applications. Discussions with these users have helped us improve the manual and go in new directions.

The case studies were developed for workshops on analysis of multivariate time-series data given at the Ecological Society of America meetings since 2005 and taught by us along with Yasmin Lucero, Stephanie Hampton, and Brice Semmens. The case study on extinction estimation and trend estimation was initially developed by Brice Semmens and later extended by us for this user guide. The algorithm behind the TMU figure in Chapter 9 was developed during a collaboration with Steve Ellner (Ellner and Holmes, 2008).

The authors are research scientists at the Northwest Fisheries Science Center (NWFSC). This work was conducted as part of our jobs at the NWFSC, a research center for NOAA Fisheries which is a United States federal government agency. A CAMEO grant from the National Science Foundation and NOAA Fisheries provided the initial impetus for the development of the package as part of a research project with Stephanie Hampton, Lindsay Scheef, and Steven Katz on analysis of marine plankton time series. During the initial stages of this work, EJW was supported on a post-doctoral fellowship from the National Research Council and MDS was partially supported by a PECASE award from the White House Office of Science and Technology Policy.

You are welcome to use the code and adapt it with attribution. You should use citation Holmes et al. (2012) for the MARSS package. It may not be used in any commercial applications nor may it be copyrighted. Use of the EM algorithm should cite Holmes (2012). Links to more code and publications on MARSS applications can be found by following the links at EEH's website

`http://faculty.washington.edu/eeholmes`. Links to our papers that use these methods can also be found at the same website.

Contents

1 The MARSS package 1
- 1.1 What does the MARSS package do? 2
- 1.2 What does MARSS output and how do I get the output? 4
- 1.3 How to get started (quickly) 4
- 1.4 Important notes about the algorithms 4
- 1.5 Troubleshooting 7
- 1.6 Other related packages 8

2 Overview of the package functions 11
- 2.1 The `MARSS()` function: inputs 11
- 2.2 The `MARSS()` function: outputs 12
- 2.3 Core functions for fitting a MARSS model 12
- 2.4 Functions for a fitted marssMLE object 13
- 2.5 Functions for marssMODEL objects 13

3 The MARSS() function 15
- 3.1 $\mathbf{u}$, $\mathbf{a}$ and π model structures 16
- 3.2 $\mathbf{Q}$, $\mathbf{R}$, Λ model structures 17
- 3.3 $\mathbf{B}$ model structures 19
- 3.4 $\mathbf{Z}$ model 19
- 3.5 Default model structures 20

4 Algorithms used in the MARSS package 23
- 4.1 Kalman filter and smoother 23
- 4.2 The exact likelihood 24
- 4.3 Maximum-likelihood parameter estimation 25
- 4.4 Parametric and innovations bootstrapping 26
- 4.5 Simulation and forecasting 27
- 4.6 Model selection 27

5 Examples . . . 29
5.1 Fixed and estimated elements in parameter matrices . . . 29
5.2 Different numbers of state processes . . . 30
5.3 Time-varying parameters . . . 39
5.4 Printing and summarizing models and model fits . . . 40
5.5 Confidence intervals on a fitted model . . . 42
5.6 Vectors of just the estimated parameters . . . 44
5.7 Kalman filter and smoother output . . . 44
5.8 Degenerate variance estimates . . . 45
5.9 Bootstrap parameter estimates . . . 47
5.10 Random initial conditions . . . 48
5.11 Data simulation . . . 49
5.12 Bootstrap AIC . . . 49
5.13 Convergence . . . 50

6 Incorporating covariates into MARSS models . . . 51
6.1 Covariates as inputs . . . 51
6.2 Examples using plankton data . . . 51
6.3 Including observation error . . . 57
6.4 Covariates with missing values or observation error . . . 58
6.5 Modeling seasonal effects . . . 60

7 Lag-p models with MARSS . . . 69
7.1 Background . . . 69
7.2 MAR(2) models . . . 69
7.3 MAR(p) models . . . 73
7.4 MARSS(p): models with observation error . . . 74

8 Case study instructions . . . 79

9 Case Study 1: Count-based population viability analysis (PVA) using corrupted data . . . 81
9.1 Background . . . 81
9.2 Simulated data with process and observation error . . . 82
9.3 Maximum-likelihood parameter estimation . . . 85
9.4 Probability of hitting a threshold $\Pi(x_d, t_e)$ . . . 90
9.5 Certain and uncertain regions . . . 94
9.6 More risk metrics and some real data . . . 96
9.7 Confidence intervals . . . 98
9.8 Comments . . . 99

10 Case study 2: Combining multi-site data to estimate regional population trends . . . 101
10.1 Harbor seals in the Puget Sound, WA. . . . 101
10.2 A single well-mixed Puget Sound population . . . 103
10.3 Different observation error structures . . . 108

10.4 Two subpopulations, north and south . . . 111
10.5 Other population structures . . . 115
10.6 Discussion . . . 117

11 Case Study 3: Identifying spatial population structure and covariance . . . 121
11.1 Harbor seals on the U.S. west coast . . . 121
11.2 How many distinct subpopulations? . . . 122
11.3 Is Hood Canal separate? . . . 126

12 Case Study 4: Dynamic factor analysis (DFA) . . . 131
12.1 Overview of dynamic factor analysis . . . 131
12.2 The data . . . 133
12.3 Setting up the model in MARSS . . . 135
12.4 Using model selection to determine the number of trends . . . 140
12.5 Using varimax rotation to determine the loadings and trends . . . 143
12.6 Examining model fits . . . 145
12.7 Adding covariates . . . 147
12.8 Questions and further analyses . . . 148

13 Case Study 5: Analyzing noisy animal tracking data . . . 149
13.1 A simple random walk model of animal movement . . . 149
13.2 Loggerhead sea turtle tracking data . . . 150
13.3 Estimate locations from bad tag data . . . 151
13.4 Using specialized packages to analyze tag data . . . 154
13.5 Questions and further analyses . . . 156

14 Case Study 6: Detection of outliers and structural breaks . . . 159
14.1 River flow in the Nile River . . . 159
14.2 Different models for the Nile flow levels . . . 159
14.3 Observation and state residuals . . . 164

15 Case Study 7: Estimation of species interaction strengths with and without covariates . . . 169
15.1 Background . . . 169
15.2 Two-species example using wolves and moose . . . 170
15.3 Analysis a four-species plankton community . . . 178

16 Case Study 8: Combining data from multiple time series . . . 191
16.1 Overview . . . 191
16.2 Salmon spawner surveys . . . 192
16.3 American kestrel abundance indices . . . 195

A Textbooks and articles that use MARSS modeling for population modeling . . . 201

B Package MARSS: Warnings and errors 205

References 211

Index 215

1

The MARSS package

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 A). 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[1].

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

$$\mathbf{x}_t = \mathbf{B}_t\mathbf{x}_{t-1} + \mathbf{u}_t + \mathbf{C}_t\mathbf{c}_t + \mathbf{w}_t, \text{ where } \mathbf{w}_t \sim \text{MVN}(0, \mathbf{Q}_t) \tag{1.1a}$$

$$\mathbf{y}_t = \mathbf{Z}_t\mathbf{x}_t + \mathbf{a}_t + \mathbf{D}_t\mathbf{d}_t + \mathbf{v}_t, \text{ where } \mathbf{v}_t \sim \text{MVN}(0, \mathbf{R}_t) \tag{1.1b}$$

$$\mathbf{x}_1 \sim \text{MVN}(\pi, \Lambda) \text{ or } \mathbf{x}_0 \sim \text{MVN}(\pi, \Lambda) \tag{1.1c}$$

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 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.

[1] Fitting via the BFGS algorithm is also provided using R 's optim function, but this is not the focus of the package.

$\mathbf{v}$ 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 $m \times m$ and $n \times n$ variance-covariance matrices.
π is either a parameter or a fixed prior. It is a $m \times 1$ matrix.
Λ 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.
$\mathbf{c}$ and $\mathbf{d}$ are inputs (no missing values) and are $p \times T$ and $q \times T$ matrices.

In some fields, the $\mathbf{u}$ and $\mathbf{a}$ terms are routinely set to 0 or the model is written in such a way that they are incorporated into $\mathbf{B}$ or $\mathbf{Z}$. However, in other fields, the $\mathbf{u}$ and $\mathbf{a}$ terms are the main objects of interest, and the model is written to explicitly show them. We include them throughout our discussion, but they can be set to zero if desired.

The 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 7. 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 errors 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 case studies in this User Guide and textbooks on MARSS models for examples of how a wide variety of autoregressive models can be written in MARSS form.

1.1 What does the MARSS package do?

Written in an unconstrained form[2], a MARSS model can be written out as follows. Two state processes ($\mathbf{x}$) and three observation processes ($\mathbf{y}$) are used here for example's sake.

[2] meaning all the elements in a parameter matrices are allowed to be different

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

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

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

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, estimated, values in a matrix have the same value. Here is an example of a MARSS model with fixed and shared parameter elements:

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_t = \begin{bmatrix} a & 0 \\ 0 & a \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_{t-1} + \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}_t, \quad \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}_t \sim \text{MVN}\left(\begin{bmatrix} 0.1 \\ u \end{bmatrix}, \begin{bmatrix} q_{11} & q_{12} \\ q_{21} & q_{22} \end{bmatrix} \right)$$

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

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

The MARSS package fits models via maximum likelihood. The MARSS package is unusual among packages for fitting MARSS models in that fitting is performed via an EM algorithm (Holmes, 2012), although fitting via the BFGS algorithm is also provided using R 's optim function. The EM algorithm gives robust estimation for datasets replete with missing values and for high-dimensional models with various constraints. The EM algorithm is also often used to provide initial conditions for the BFGS algorithm (or an MCMC routine) in order to improve the performance of those algorithms. In addition, the MARSS package supplies functions for bootstrap and approximate confidence intervals, parametric and non-parametric bootstrapping, model selection (AIC and bootstrap AIC), simulation, and bootstrap bias correction.

1.2 What does MARSS output and how do I get the output?

MARSS models are used in many different ways and different users will want different types of output. Some users will want the parameter estimates while others want the smoothed states and others want to use MARSS to interpolate missing values and want the expected values of missing data. The best way to find out how to get output is to type `?print.MARSS` at the command line after installing MARSS. This help page discusses how to get parameter estimates in different forms, the smoothed and filtered states, all the Kalman filter and smoother output, all the expectations of y (missing data), confidence intervals and bias estimates for the parameters, and standard errors of the states. If you are looking only for Kalman filter and smoother output, see the relevant section in the Chapter 4 and `?MARSSkf` at the R command line.

1.3 How to get started (quickly)

If you already work with models in the form of Equation 1.1, 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 5. Your data need to be a matrix (not dataframe) with time going across the columns and any non-data columns (like year) removed. The MARSS functions assume discrete time steps and you will need a column for each time step. Replace any missing time steps with a missing value holder (e.g. NA). Write your model down on paper and identify which parameters correspond to which parameter matrices in Equation 1.1. Call the `MARSS()` function (Chapter 3) using your data and using the `model` argument to specify the structure of each parameter.

1.4 Important notes about the algorithms

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—with correspondingly an infinite number of solutions.

Specifying a properly constrained model with a unique solution is imperative. How do you know if the model is properly constrained? If you are using a MARSS model form that is widely used, then you can probably assume that it is properly constrained. If you go to papers where someone developed the model or method, the issue of constraints necessary to ensure "identifiability" will likely be addressed if it is an issue. Are you fitting novel MARSS models? Then you will need to do some study on identifiability in this class of models using textbooks (see the textbook list at end of this User Guide). Note,

often textbooks do not address identifiability explicitly, rather it's addressed implicitly by only showing a model constructed in such a way that it is identifiable. In our work, if we suspect identification problems, we will often first do a Bayesian analysis with flat priors and look for oddities in the posteriors, such as ridges, plateaus or bimodality.

All the EM code in the MARSS package is in native R . Thus the model fitting is slow (relatively). The classic Kalman filter/smoother algorithm, as shown in Shumway and Stoffer (2006, p. 331-335), is based on the original smoother presented in Rauch (1963). This Kalman filter is provided in function `MARSSkfss`, but the default Kalman filter and smoother used in the MARSS package is based on the algorithm in Kohn and Ansley (1989) and papers by Koopman et al. This Kalman filter and smoother is provided in the KFAS package (Helske 2012). Table 2 in Koopman (1993) indicates that the classic algorithm is 40-100 times slower than the algorithm given in Kohn and Ansley (1989), Koopman (1993), and Koopman et al. (1999). The MARSS package function `MARSSkfas` provides a translator between the model objects in MARSS and those in KFAS so that the KFAS functions can be used. `MARSSkfas` also includes a lag-one covariance smoother algorithm as this is not output by the KFAS functions, and it provides proper formulation of the priors so that one can use the KFAS functions when the prior on the states is set at $t = 0$ instead of $t = 1$ (and no, simply off-setting your data to start at t=2 and sending that to to a $t_{init} = 1$ Kalman filter would not give the correct output).

EM algorithms will quickly get in the vicinity of the maximum likelihood, but the final approach to the maximum is generally slow relative to quasi-Newton methods. On the flip side, EM algorithms are quite robust to initial conditions choices and can be extremely fast at getting close to the MLE values for high-dimensional models. The MARSS package also allows one to use the BFGS method to fit MARSS models, thus one can use an EM algorithm to "get close" and then the BFGS algorithm to polish off the estimate. Restricted maximum-likelihood algorithms are also available for AR(1) state-space models, both univariate (Staples et al., 2004) and multivariate (Hinrichsen, 2009). REML can give parameter estimates with lower variance than plain maximum-likelihood algorithms. However, the algorithms for REML when there are missing values are not currently available (although that will probably change in the near future). Another maximum-likelihood method is data-cloning which adapts MCMC algorithms used in Bayesian analysis for maximum-likelihood estimation (Lele et al., 2007).

Missing values are seamlessly accommodated with the MARSS package. Simply specify missing data with NAs. The likelihood computations are exact and will deal appropriately with missing values. However, no innovations[3] bootstrapping can be done if there are missing values. Instead parametric bootstrapping must be used.

[3] referring to the non-parametric bootstrap developed by Stoffer and Wall (1991).

You should be aware that maximum-likelihood estimates of variance in MARSS models are fundamentally biased, regardless of the algorithm used. This bias is more severe when one or the other of **R** or **Q** is very small, and the bias does not go to zero as sample size goes to infinity. The bias arises because variance is constrained to be positive. Thus if **R** or **Q** is essentially zero, the mean estimate will not be zero and thus the estimate will be biased high while the corresponding bias of the other variance will be biased low. You can generate unbiased variance estimates using a bootstrap estimate of the bias. The function `MARSSparamCIs()` will do this. However be aware that adding an *estimated* bias to a parameter estimate will lead to an increase in the variance of your parameter estimate. The amount of variance added will depend on sample size.

You should also be aware that mis-specification of the prior on the initial states (π and Λ) can have catastrophic effects on your parameter estimates if your prior conflicts with the distribution of the initial states implied by the MARSS model. These effects can be very difficult to detect because the model will appear to be well-fitted. Unless you have a good idea of what the parameters should be, you might not realize that your prior conflicts.

The most common problems, we have found with priors on $\mathbf{x}_0$ are the following. Problem 1) The correlation structure in Λ (whether the prior is diffuse or not) does not match the correlation structure in $\mathbf{x}_0$ implied by your model. For example, you specify a diagonal Λ (independent states), but the implied distribution has correlations. Problem 2) The correlation structure in Λ does not match the structure in $\mathbf{x}_0$ implied by constraints you placed on π. For example, you specify that all values in π are shared, yet you specify that Λ is diagonal (independent). Unfortunately, using a diffuse prior does not help with these two problems because the diffuse prior still has a correlation structure and can still conflict with the implied correlation in $\mathbf{x}_0$. One way to get around these problems is to set Λ=0 (a $m \times m$ matrix of zeros) and estimate $\pi \equiv \mathbf{x}_0$ only. Now π is a fixed but unknown (estimated) parameter, not the mean of a distribution. In this case, Λ does not exist in your model and there is no conflict with the model. Unfortunately estimating π as a parameter is not always robust. If you specify that Λ=0 and specify that π corresponds to $\mathbf{x}_0$, but your model "explodes" when run backwards, you cannot estimate π because you cannot get a good estimate of $\mathbf{x}_0$. Sometimes this can be avoided by specifying that π corresponds to $\mathbf{x}_1$ so that it can be constrained by the data $\mathbf{y}_1$. In summary, if the implied correlation structure of your initial states is independent (diagonal variance-covariance matrix), you should generally be ok with a diagonal and high variance prior or with treating the initial states as parameters (with $\Lambda = 0$). But if your initial states have an implied correlation structure that is not independent, then proceed with caution.

There is a large class of models in the statistical finance literature that have the form

$$\mathbf{x}_{t+1} = \mathbf{B}\mathbf{x}_t + \Gamma\eta_t$$
$$\mathbf{y}_t = \mathbf{Z}\mathbf{x}_t + \eta_t$$

For example, ARMA(p,q) models can be written in this form. The MARSS model framework in this package will not allow you to write models in that form. You can put the η_t into the $\mathbf{x}_t$ vector and set $\mathbf{R} = 0$ to make models of this form using the MARSS form, but the EM algorithm in the MARSS package won't let you estimate parameters because the parameters will drop out of the full likelihood being maximized in the algorithm.

1.5 Troubleshooting

Numerical errors due to ill-conditioned matrices are not uncommon when fitting MARSS models. The Kalman and EM algorithms need inverses of matrices. If those matrices become ill-conditioned, for example all elements are close to the same value, then the algorithm becomes unstable. Warning messages will be printed if the algorithms are becoming unstable and you can set `control$trace=1`, to see details of where the algorithm is becoming unstable. Whenever possible, you should avoid using shared π values in your model[4]. The way our algorithm deals with Λ tends to make this case unstable, especially if $\mathbf{R}$ is not diagonal. In general, estimation of a non-diagonal $\mathbf{R}$ is more difficult, more prone to ill-conditioning, and more data-hungry.

You may also see non-convergence warnings, especially if your MLE model turns out to be degenerate. This means that one of the elements on the diagonal of your $\mathbf{Q}$ or $\mathbf{R}$ matrix are going to zero (are degenerate). It will take the EM algorithm forever to get to zero. BFGS will have the same problem, although it will often get a bit closer to the degenerate solution. If you are using `method="kem"`, MARSS will warn you if it looks like the solution is degenerate. If you use `control=list(allow.degen=TRUE)`, the EM algorithm will attempt to set the degenerate variances to zero (instead of trying to get to zero using an infinite number of iterations). However, if one of the variances is going to zero, first think about why this is happening. This is typically caused by one of three problems: 1) you made a mistake in inputting your data, e.g. used -99 as the missing value in your data but did not replace these with NAs before passing to MARSS, 2) your data are not sufficient to estimate multiple variances or 3) your data are inconsistent with the model you are trying fit.

The algorithms in the MARSS package are designed for cases where the $\mathbf{Q}$ and $\mathbf{R}$ diagonals are all non-minuscule. For example, the EM update equation for $\mathbf{u}$ will grind to a halt (not update $\mathbf{u}$) if $\mathbf{Q}$ is tiny (like 1E-7). Conversely, the BFGS equations are likely to miss the maximum-likelihood when $\mathbf{R}$ is tiny because then the likelihood surface becomes hyper-sensitive to π. The solution is to use the degenerate likelihood function for the likelihood calculation and the EM update equations. MARSS will implement this automatically when $\mathbf{Q}$ or $\mathbf{R}$ diagonal elements are set to zero and will try setting $\mathbf{Q}$ and $\mathbf{R}$ terms

[4] An example of a π with shared values is $\pi = \left[\begin{smallmatrix} a \\ a \\ a \end{smallmatrix}\right]$.

to zero automatically if `control$allow.degen=TRUE`. One odd case can occur when $\mathbf{R}$ goes to zero (a matrix of zeros), but you are estimating π. If `model$tinitx=1`, then π must be $\mathbf{y}_1$ as $\mathbf{R}$ goes to zero, but as $\mathbf{R}$ goes to zero, the log-likelihood will go (correctly) to infinity. But if you set $\mathbf{R} = 0$, the log-likelihood will be finite. The reason is that $\mathbf{R} \approx 0$ and $\mathbf{R} = 0$ specify different likelihoods. In the first, the determinant $\mathbf{R}$ will appear, and this goes to positive infinity as $\mathbf{R}$ goes to zero. In the second case, $\mathbf{R}$ does not appear in the likelihood and so the determinant of $\mathbf{R}$ does not appear. If some elements of the diagonal of $\mathbf{R}$ are going to zero, you should be suspect of the parameter estimates. Sometimes the structure of your data, e.g. one data value followed by a long string of missing values, is causing an odd spike in the likelihood at $\mathbf{R} \approx 0$. Try manually setting $\mathbf{R}$ equal to zero to get the correct log-likelihood[5].

1.6 Other related packages

Packages that will do Kalman filtering and smoothing are many, but packages that estimate the parameters in a MARSS model, especially constrained MARSS models, are much less common. The following are those with which we are familiar, however there are certainly more packages for estimating MARSS models in engineering and economics of which we are unfamiliar. The MARSS package is unusual in that it uses an EM algorithm for maximizing the likelihood as opposed to a Newton-esque method (e.g. BFGS). The package is also unusual in that it allows you to specify the initial conditions at $t = 0$ or $t = 1$, allows degenerate models (with some of the diagonal elements of $\mathbf{R}$ or $\mathbf{Q}$ equal to zero). Lastly, model specification in the MARSS package has a one-to-one relationship between the model list in MARSS and the model as you would write it on paper as a matrix equation. This makes the learning curve a bit less steep. However, the MARSS package has not been optimized for speed and probably will be really slow if you have time-series data with a lot of time points.

DLM DLM is an R package for fitting MARSS models. Our impression is that it is mainly Bayesian focused but it does allow MLE estimation via the `optim()` function. It has a book, Dynamic Linear Models with R by Petris et al., which has many examples of how to write MARSS models for different applications.

sspir sspir an R package for fitting ARSS (univariate) models with Gaussian, Poisson and binomial error distributions.

dse dse (Dynamic Systems Estimation) is an R package for multivariate Gaussian state space models with a focus on ARMA models.

[5] The likelihood returned when $\mathbf{R} \approx 0$ is not incorrect. It is just not the likelihood that you probably want. You want the likelihood where the $\mathbf{R}$ term is dropped because it is zero.

SsfPack SsfPack is a package for Ox/Splus that fits constrained multivariate Gaussian state space models using mainly (it seems) the BFGS algorithm but the newer versions support other types of maximization. SsfPack is very flexible and written in C to be fast. It has been used extensively on statistical finance problems and is optimized for dealing with large (financial) data sets. It is used and documented in Time Series Analysis by State Space Methods by Durbin and Koopman, An Introduction to State Space Time Series Analysis by Commandeur and Koopman, and Statistical Algorithms for Models in State Space Form: SsfPack 3.0, by Koopman, Shephard, and Doornik.

Brodgar The Brodgar software was developed by Alain Zuur to do (among many other things) dynamic factor analysis, which involves a special type of MARSS model. The methods and many example analyses are given in Analyzing Ecological Data by Zuur, Ieno and Smith. This is the one package that we are aware of that also uses an EM algorithm for parameter estimation.

eViews eViews is a commercial economics software that will estimate at least some types of MARSS models.

KFAS The KFAS R package provides a fast Kalman filter and smoother. Examples in the package show how to estimate MARSS models using the KFAS functions and R 's `optim()` function. The MARSS package uses the filter and smoother functions from the KFAS package.

S+FinMetrics S+FinMetrics is a S-plus module for fitting MAR models, which are called vector autoregressive (VAR) models in the economics and finance literature. It has some support for state-space VAR models, though we haven't used it so are not sure which parameters it allows you to estimate. It was developed by Andrew Bruce, Doug Martin, Jiahui Wang, and Eric Zivot, and it has a book associated with it: Modeling Financial Time Series with S-plus by Eric Zivot and Jiahui Wang.

kftrack The kftrack R package provides a suite of functions specialized for fitting MARSS models to animal tracking data.

2

Overview of the package functions

The MARSS package is object-based. It has two main types of objects: a model object (class marssMODEL) and a maximum-likelihood fitted model object (class marssMLE). A marssMODEL object specifies the structure of the model to be fitted. It is an R code version of the MARSS equation (Equation 1.1). A marssMLE object specifies both the model and the information necessary for fitting (initial conditions, controls, method). If the model has been fitted, the marssMLE object will also have the parameter estimates and (optionally) confidence intervals and bias.

2.1 The `MARSS()` function: inputs

The function `MARSS()` is an interface to the core fitting functions in the MARSS package. It allows a user to fit a MARSS model using a list to describe the model structure. It returns marssMODEL and marssMLE objects which the user can later use in other functions, e.g. simulating or computing bootstrap confidence intervals.

`MLEobj=MARSS(data, model=list(), ..., fit=TRUE)` This function will fit a MARSS model to the data using a model list which is a list describing the structure of the model parameter matrices. In the default model, i.e. if you use `MARSS(dat)` with no `model` argument, **Z** and **B** are the identity matrix, **R** is a diagonal matrix with one variance, **Q** is a diagonal matrix with unique variances, **u** is unique, **a** is scaling, and **C**, **c**, **D**, and **d** are all zero. The output is a marssMLE object where the estimated parameter matrices are in `MLEobj$par`. If `fit=FALSE`, it returns a minimal marssMLE object that is ready for passing to a fitting function (below) but with no `par` element.

2.2 The MARSS() function: outputs

The marssMLE object returned by a MARSS() call include the estimated parameters, states, and expected values of any missing data. Derived statistics, such as confidence intervals and standard errors, can be computed using the functions described below. However the `print` method for marssMLE objects will print or compute all the frequently needed output using the `what=` argument in the print call. Type `?print.MARSS` at the R command line to see the print help file.

estimated parameters `coef(marssMLE)` The coef function can output parameters in a variety of formats. See `?coef.marssMLE`.

residuals `residuals(marssMLE)` See `?residuals.marssMLE` for a discussion of standardized residuals in the context of MARSS models.

Kalman filter and smoother output The smoothed states are in `marssMLE$states` but the full Kalman filter and smoother output is available from `MARSSkf(marssMLE)`. See `?MARSSkf` for a discussion of the Kalman filter and smoother outputs.

expected value of missing y `MARSShatyt(marssMLE)` See `?MARSShatyt` for a discussion of the expectations involving y.

2.3 Core functions for fitting a MARSS model

The following core functions are designed to work with 'unfitted' marssMLE objects, that is a marssMLE object without the `par` element. Users do not normally need to call the `MARSSkem` or `MARSSoptim` functions since `MARSS()` will call those. Below, `MLEobj` means the argument is a marssMLE object. Note, these functions can be called with a marssMLE object with a `par` element, but these functions will overwrite that element.

`MLEobj=MARSSkem(MLEobj)` This will fit a MARSS model via the EM algorithm to the data using a properly specified marssMLE object, which has data, the marssMODEL object and the necessary initial condition and control elements. See the appendix on the object structures in the MARSS package. `MARSSkem` does no error-checking. See `is.marssMLE()`. `MARSSkem` uses `MARSSkf` described below.

`MLEobj=MARSSoptim(MLEobj)` This will fit a MARSS model via the BFGS algorithm provided in `optim()`. This requires a properly specified marssMLE object, such as would be passed to `MARSSkem`.

`MLEobj=MARSSmcinit(MLEobj)` This will perform a Monte Carlo initial conditions search and update the marssMLE object with the best initial conditions from the search.

`is.marssMLE(MLEobj)` This will check that a marssMLE object is properly specified and ready for fitting. This should be called before `MARSSkem` or `MARSSoptim` is called. This function is not typically needed if using `MARSS()` since `MARSS()` builds the model object for the user and does error-checking on model structure.

2.4 Functions for a fitted marssMLE object

The following functions use a marssMLE object that has a populated `par` element, i.e. a marssMLE object returned from one of the fitting functions (`MARSS`, `MARSSkem`, `MARSSoptim`). Below `modelObj` means the argument is a marssMODEL object and `MLEobj` means the argument is a marssMLE object. Type `?function.name` to see information on function usage and examples.

`kf=MARSSkf(MLEobj)` This will compute the expected values of the hidden states given data via the Kalman filter (to produce estimates conditioned on $1:t-1$) and the Kalman smoother (to produce estimates conditioned on $1:T$). The function also returns the exact likelihood of the data conditioned on `MLEobj$par`. A variety of other Kalman filter/smoother information is also output (`kf` is a list of output); see `?MARSSkf` for details.

`MLEobj=MARSSaic(MLEobj)` This adds model selection criteria, AIC, AICc, and AICb, to a marssMLE object.

`boot=MARSSboot(MLEobj)` This returns a list containing bootstrapped parameters and data via parametric or innovations bootstrapping.

`MLEobj=MARSShessian(MLEobj)` This adds a numerically estimated Hessian matrix to a marssMLE object.

`MLEobj=MARSSparamCIs(MLEobj)` This adds standard errors, confidence intervals, and bootstrap estimated bias for the maximum-likelihood parameters using bootstrapping or the Hessian to the passed-in marssMLE object.

`sim.data=MARSSsimulate(MLEobj)` This returns simulated data from a MARSS model specified via a list of parameter matrices in `MLEobj$parList` (this is a list with elements `Q`, `R`, `U`, etc).

`paramVec=MARSSvectorizeparam(MLEobj)` This returns the estimated (and only the estimated) parameters as a vector. This is useful for storing the results of simulations and for writing functions that fit MARSS models using R's `optim` function.

`new.MLEobj=MARSSvectorizeparam(MLEobj, paramVec)` This will return a marssMLE object in which the estimated parameters (which are in `MLEobj$par` along with the fixed values) are replaced with the values in `paramVec`.

2.5 Functions for marssMODEL objects

`is.marssMODEL(modelObj)` This will check that the free and fixed matrices in a marssMODEL object are properly specified. This function is not typically needed if using `MARSS()` since `MARSS()` builds the marssMODEL object for the user and does error-checking on model structure.

`summary(modelObj)` This will print the model parameter matrices showing the fixed values (in parentheses) and the location of the estimated elements. The estimated elements are shown as g1, g2, g3, ... which indicates

which elements are shared (i.e., forced to have the same value). For example, an i.i.d. **R** matrix would appear as a diagonal matrix with just g1 on the diagonal.

3

The MARSS() function

From the user perspective, the main package function is `MARSS()`. This fits a MARSS model (Equation 1.1) to a matrix of data:

```
MARSS(data, model=list(), form="marxss"))
```

The model argument is a list with names `B`, `U`, `C`, `c`, `Q`, `Z`, `A`, `D`, `d`, `R`, `x0`, `V0`. Elements can be left off to use default values. The form argument tells `MARSS()` how to use the model list elements. The default is `form="marxss"` which is the model in Equation 1.1.

The data must be passed in as a $n \times T$ matrix; that is time goes across columns. A vector is not a matrix, nor is a dataframe. A data matrix consisting of three time series ($n = 3$) with six time steps might look like

$$\mathbf{y} = \begin{bmatrix} 1 & 2 & NA & NA & 3.2 & 8 \\ 2 & 5 & 3 & NA & 5.1 & 5 \\ 1 & NA & 2 & 2.2 & NA & 7 \end{bmatrix}$$

where NA denotes a missing value.

The argument `model` specifies the structure of the MARSS model. It is a list, where the list elements for each model parameter specifies the form of that parameter.

The most general way to specify model structure is to use a list matrix. The list matrix allows one to combine fixed and estimated elements in one's parameter specification. It allows a one-to-one correspondence between how you write the parameter matrix on paper and how you specify it in R . For example, let's say $\mathbf{Q}$ and $\mathbf{u}$ have the following forms in your model:

$$\mathbf{Q} = \begin{bmatrix} q & 0 & 0 \\ 0 & q & 0 \\ 0 & 0 & 1 \end{bmatrix} \text{ and } \mathbf{u} = \begin{bmatrix} 0.05 \\ u_1 \\ u_2 \end{bmatrix}$$

So $\mathbf{Q}$ is a diagonal matrix with the 3rd variance fixed at 1 and the 1st and 2nd estimated and equal. The 1st element of $\mathbf{u}$ is fixed, and the 2nd and 3rd are estimated and different. You can specify this using a list matrix:

```
Q=matrix(list("a",0,0,0,"a",0,0,0,1),3,3)
U=matrix(list(0.05,"b","c"),3,1)
```

If you print out `Q` and `U`, you will see they look exactly like **Q** and **u** written above. MARSS will keep the fixed values fixed and estimate a, b, and c.

List matrices allow the most flexible model structures, but MARSS also has text shortcuts for a number of common model structures. Below, the possible ways to specify each model parameter are shown, using $m = 3$ (the number of hidden state processes) and $n = 3$ (number of observation time series).

3.1 u, a and π model structures

u, **a** and π are all row matrices and the options for specifying their structures are the same. The most general way to specify structure is to use a list matrix, but there are text shortcuts for the common structures. **a** has one special option, `"scaling"` described below. The allowable structures are shown using **u** as an example. Note that you should be careful about specifying shared structure in π because you need to make sure the structure in Λ matches. For example, if you require that all the π values are shared (equal) then Λ cannot be a diagonal matrix since that would be saying that the π values are independent, which they are clearly not if you force them to be equal.

`U=matrix(list(),m,1)`: This is the most general form and allows one to specify fixed and estimated elements in **u**. Each character string in **u** is the name of one of the **u** elements to be estimated. For example if `U=matrix(list(0.01,"u","u"),3,1)`, then **u** in the model has the following structure:

$$\begin{bmatrix} 0.01 \\ u \\ u \end{bmatrix}$$

`U=matrix(c(),m,1)`, where the values in `c()` are all character strings: each character string is the name of an element to be estimated. For example if `U=matrix(c("u1","u1","u2"),3,1)`, then **u** in the model has the following structure:

$$\begin{bmatrix} u_1 \\ u_1 \\ u_2 \end{bmatrix}$$

and two values being estimated. `U=matrix(list("u1","u1","u2"),3,1)` has the same effect.

`U="unequal"` or `U="unconstrained"`: Both of these stings indicate that each element of **u** is estimated. If $m = 3$, then **u** would have the form:

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix}$$

`U="equal"`: There is only one value in **u**:

$$\begin{bmatrix} u \\ u \\ u \end{bmatrix}$$

`U=matrix(c(),m,1)`, where the values in `c()` all numerical values: **u** is fixed and has no estimated values. If `U=matrix(c(0.01,1,-0.5),3,1)`, then **u** in the model has the following structure:

$$\begin{bmatrix} 0.01 \\ 1 \\ -0.5 \end{bmatrix}$$

`U=matrix(list(0.01,1,-0.5),3,1)` would have the same effect.

`U="zero"`: **u** is all zero:

$$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

The **a** parameter has a special option, `"scaling"`, which is the default behavior. In this case, **a** is treated like a scaling parameter. If there is only one **y** row associated with an **x** row, then the corresponding **a** element is 0. If there are more than one **y** rows associated with an **x** row, then the first **a** element is set to 0 and the others are estimated. For example, say $m = 2$ and $n = 4$ and **Z** looks like the following:

$$\mathbf{Z} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Then the 1st-3rd rows of **y** are associated with the first row of **x**, and the 4th row of **y** is associated with the last row of **x**. Then if **a** is specified as `"scaling"`, **a** has the following structure:

$$\begin{bmatrix} 0 \\ a_1 \\ a_2 \\ 0 \end{bmatrix}$$

3.2 Q, R, Λ model structures

The possible **Q**, **R**, and Λ model structures are identical, except that **R** is $n \times n$ while **Q** and Λ are $m \times m$. The most general way to specify these variance-covariance matrices is using a list matrix. All types of structures can be specified using a list matrix, but there are also text shortcuts for specifying common structures. The structures are shown using **Q** as the example.

`Q=matrix(list(),m,m)`: This is the most general way to specify the parameters and allows there to be fixed and estimated elements. Each character string in the list matrix is the name of one of the **Q** elements to be estimated, and each numerical value is a fixed value. For example if
`Q=matrix(list("s2a",0,0,0,"s2a",0,0,0,"s2b"),3,3)`,
then **Q** has the following structure:

$$\begin{bmatrix} \sigma_a^2 & 0 & 0 \\ 0 & \sigma_a^2 & 0 \\ 0 & 0 & \sigma_b^2 \end{bmatrix}$$

Note that `diag(c("s2a","s2a","s2b"))` will not have the desired effect of producing a matrix with numeric 0s on the off-diagonals. It will have character 0s and MARSS will interpret "0" as the name of an element of **Q** to be estimated. Instead, the following two lines can be used:

```
Q=matrix(list(0),3,3)
diag(Q)=c("s2a","s2a","s2b")
```

`Q="diagonal and equal"`: There is only one process variance value in this case:

$$\begin{bmatrix} \sigma^2 & 0 & 0 \\ 0 & \sigma^2 & 0 \\ 0 & 0 & \sigma^2 \end{bmatrix}$$

`Q="diagonal and unequal"`: There are m process variance values in this case:

$$\begin{bmatrix} \sigma_1^2 & 0 & 0 \\ 0 & \sigma_2^2 & 0 \\ 0 & 0 & \sigma_3^2 \end{bmatrix}$$

`Q="unconstrained"`: There are values on the diagonal and the off-diagonals of **Q** and the variances and covariances are all different:

$$\begin{bmatrix} \sigma_1^2 & \sigma_{1,2} & \sigma_{1,3} \\ \sigma_{1,2} & \sigma_2^2 & \sigma_{2,3} \\ \sigma_{1,3} & \sigma_{2,3} & \sigma_3^2 \end{bmatrix}$$

There are m process variances and $(m^2-m)/2$ covariances in this case, so $(m^2+m)/2$ values to be estimated. Note that variance-covariance matrices are never truly unconstrained since the upper and lower triangles of the matrix must be equal.

`Q="equalvarcov"`: There is one process variance and one covariance:

$$\begin{bmatrix} \sigma^2 & \beta & \beta \\ \beta & \sigma^2 & \beta \\ \beta & \beta & \sigma^2 \end{bmatrix}$$

`Q=matrix(c(), m, m)`, where all values in `c()` are character strings: Each element in **Q** is estimated and each character string is the name of a value to be estimated. Note if $m = 1$, you still need to wrap its value in `matrix()` so that its class is matrix.

`Q=matrix(c(), m, m)`, where all values in `c()` are numeric values: Each element in **Q** is fixed to the values in the matrix.

`Q="identity"`: The **Q** matrix is the identity matrix:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

`Q="zero"`: The **Q** matrix is all zeros:

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Be careful when setting Λ model structures. Mis-specifying the structure of Λ can have catastrophic, but difficult to discern, effects on your estimates. See the comments on priors in Chapter 1.

3.3 B model structures

Like the variance-covariance matrices (**Q**,**R** and Λ), **B** can be specified with a list matrix to allow you to have both fixed and shared elements in the **B** matrix. Character matrices and matrices with fixed values operate the same way as for the variance-covariance matrices. In addition, the same text shortcuts are available: "unconstrained", "identity", "diagonal and equal", "diagonal and unequal", "equalvarcov", and "zero". A fixed **B** can be specified with a numeric matrix, but all eigenvalues must fall within the unit circle; meaning `all(abs(eigen(B)$values)<=1)`.

3.4 Z model

Like **B** and the variance-covariance matrices, **Z** can be specified with a list matrix to allow you to have both fixed and estimated elements in **Z**. If **Z** is a square matrix, many of the same text shortcuts are available: "diagonal and equal", "diagonal and equal", and "equalvarcov". If **Z** is a design matrix[1], then a special shortcut is available using `factor()` which allows you to specify which **y** rows are associated with which **x** rows. See Chapter 5 and the case studies for more examples.

[1] a matrix with only 0s and 1s and where the row sums are all equal to 1

`Z=factor(c(1,1,1))`: All **y** time series are observing the same (and only) hidden state trajectory x ($n = 3$ and $m = 1$):

$$\mathbf{Z} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

`Z=factor(c(1,2,3))`: Each time series in **y** corresponds to a different hidden state trajectory. This is the default **Z** model and in this case $n = m$:

$$\mathbf{Z} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

`Z=factor(c(1,1,2))`: The first two time series in **y** corresponds to one hidden state trajectory and the third **y** time series corresponds to a different hidden state trajectory. Here $n = 3$ and $m = 2$:

$$\mathbf{Z} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

The **Z** model can be specified using either numeric or character factor levels. `c(1,1,2)` is the same as `c("north","north","south")`

`Z="identity"`: This is the default behavior. This means **Z** is a $n \times n$ identity matrix and $m = n$. If $n = 3$, it is the same as `Z=factor(c(1,2,3))`.

`Z=matrix(c(), n, m)`, where the elements in `c()` are all strings: Passing in a $n \times m$ character matrix, means that each character string is a value to be estimated. Be careful that you are specifying an identifiable model when using this option.

`Z=matrix(c(), n, m)`, where the elements in `c()` are all numeric: Passing in a $n \times m$ numeric matrix means that **Z** is fixed to the values in the matrix. The matrix must be numeric but it does not need to be a design matrix.

`Z=matrix(list(), n, m)`: Passing in a $n \times m$ list matrix allows you to combine fixed and estimated values in the **Z** matrix. Be careful that you are specifying an identifiable model.

3.5 Default model structures

The defaults for the model arguments in `form="marxss"` are

`Z="identity"` each y in **y** corresponds to one x in **x**
`B="identity"` no interactions among the x's in **x**
`U="unequal"` the u's in **u** are all different

`Q="diagonal and unequal"` process errors are independent but have different variances
`R="diagonal and equal"` the observations are i.i.d.
`A="scaling"` **a** is a set of scaling factors
`C="zero"` and `D="zero"` no inputs.
`c="zero"` and `d="zero"` no inputs.
`pi="unequal"` all initial states are different
`V0="zero"` the initial condition on the states ($\mathbf{x}_0$ or $\mathbf{x}_1$) is fixed but unknown

4

Algorithms used in the MARSS package

4.1 Kalman filter and smoother

The MARSS model (Equation 1.1) is a linear dynamical system in discrete time. In 1960, Rudolf Kalman published the Kalman filter (Kalman, 1960), a recursive algorithm that solves for the expected value of the hidden state(s) at time t conditioned on the data up to time t: $\mathrm{E}(\mathbf{X}_t|\mathbf{y}_1^t)$. The Kalman filter gives the optimal (lowest mean square error) estimate of the unobserved $\mathbf{x}_t$ based on the observed data up to time t for this class of linear dynamical system. The Kalman smoother (Rauch et al., 1965) solves for the expected value of the hidden state(s) conditioned on all the data: $\mathrm{E}(\mathbf{X}_t|\mathbf{y}_1^T)$. If the errors in the stochastic process are Gaussian, then the estimators from the Kalman filter and smoother are also the maximum-likelihood estimates.

However, even if the the errors are not Gaussian, the estimators are optimal in the sense that they are estimators with the least variability possible. This robustness is one reason the Kalman filter is so powerful—it provides well-behaving estimates of the hidden states for all kinds of multivariate autoregressive processes, not just Gaussian processes. The Kalman filter and smoother are widely used in time-series analysis, and there are many textbooks covering it and its applications. In the interest of giving the reader a single point of reference, we use Shumway and Stoffer (2006) as our primary reference.

The `MARSSkf` function provides the Kalman filter and smoother output using one of two algorithms (specified by verb@fun.kf@). The algorithm in `MARSSkfss` is that shown in Shumway and Stoffer (2006). This algorithm is not computationally efficient; see Koopman et al. (1999, sec. 4.3) for a more efficient Kalman filter implementation. The Koopman et al. implementation is provided in the functions `MARSSkfas` using the KFAS R package. `MARSSkfss` (and `MARSSkfas` with a few exceptions) has the following outputs:

`xtt1` The expected value of $\mathbf{X}_t$ conditioned on the data up to time $t-1$.
`xtt` The expected value of $\mathbf{X}_t$ conditioned on the data up to time t.

`xtT` The expected value of $\mathbf{X}_t$ conditioned on all the data from time 1 to T. This the smoothed state estimate.

`Vtt1` The variance of $\mathbf{X}_t$ conditioned on the data up to time $t-1$. Denoted P_t^{t-1} in section 6.2 in Shumway and Stoffer (2006).

`Vtt` The variance of $\mathbf{X}_t$ conditioned on the data up to time t. Denoted P_t^t in section 6.2 in Shumway and Stoffer (2006).

`VtT` The variance of $\mathbf{X}_t$ conditioned on all the data from time 1 to T.

`Vtt1T` The covariance of $\mathbf{X}_t$ and $\mathbf{X}_{t-1}$ conditioned on all the data, 1 to T.

`Kt` The Kalman gain. This is part of the update equations and relates to the amount `xtt1` is updated by the data at time t to produce `xtt`. Not output by `MARSSkfas`.

`J` This is similar to the Kalman gain but is part of the Kalman smoother. See Equation 6.49 in Shumway and Stoffer (2006). Not output by `MARSSkfas`.

`Innov` This has the innovations at time t, defined as $\varepsilon_t \equiv \mathbf{y}_t\text{-}\mathrm{E}(\mathbf{Y}_t)$. These are the residuals, the difference between the data and their predicted values. See Equation 6.24 in Shumway and Stoffer (2006). Not output by `MARSSkfas`.

`Sigma` This has the Σ_t, the variance-covariance matrices for the innovations at time t. This is used for the calculation of confidence intervals, the s.e. on the state estimates and the likelihood. See Equation 6.25 in Shumway and Stoffer (2006) for the Σ_t calculation. Not output by `MARSSkfas`.

`logLik` The log-likelihood of the data conditioned on the model parameters.

4.2 The exact likelihood

The likelihood of the data given a set of MARSS parameters is part of the output of the `MARSSkfss` and `MARSSkfas` functions. The likelihood computation is based on the innovations form of the likelihood (Schweppe, 1965) and uses the output from the Kalman filter:

$$\log \mathrm{L}(\Theta|data) = -\frac{N}{2\log(2\pi)} - \frac{1}{2}\left(\sum_{t=1}^{T}\log|\Sigma_t| + \sum_{t=1}^{T}(\varepsilon_t)^\top\Sigma_t^{-1}\varepsilon_t\right) \tag{4.1}$$

where N is the total number of data points, ε_t is the innovations at time t and $|\Sigma_t|$ is the determinant of the innovations variance-covariance matrix at time t. Reference Equation 6.62 in Shumway and Stoffer (2006). However there are a few differences between the log-likelihood output by `MARSSkf` and `MARSSkfas` and that described in Shumway and Stoffer (2006).

The standard likelihood calculation (Equation 6.62 in Shumway and Stoffer (2006)) is biased when there are missing values in the data, and the missing data modifications discussed in Section 6.4 in Shumway and Stoffer (2006) do not correct for this bias. Harvey (1989), Section 3.4.7, discusses at length that the standard missing values correction leads to an inexact likelihood when there are missing values. The bias is minor if there are few missing values, but

it becomes severe as the number of missing values increases. Many ecological datasets may have over 25% missing values and this level of missing values leads to a very biased likelihood if one uses the inexact formula. Harvey (1989) provides some non-trivial ways to compute the exact likelihood.

We use instead the exact likelihood correction for missing values that is presented in Section 12.3 in Brockwell and Davis (1991). This solution is straight-forward to implement. The correction involves the following changes to ε_t and Σ_t in the Equation 4.1. Suppose the value $y_{i,t}$ is missing. First, the corresponding i-th value of ε_t is set to 0. Second, the i-th diagonal value of Σ_t is set to 1 and the off-diagonal elements on the i-th column and i-th row are set to 0.

4.3 Maximum-likelihood parameter estimation

4.3.1 EM algorithm

Function `MARSSkem` in the MARSS package provides a maximum-likelihood algorithm which uses an Expectation-Maximization (EM) algorithm with output from the Kalman smoother (Holmes, 2012). EM algorithms are widely used algorithms that extend maximum-likelihood estimation to cases where there are hidden random variables in a model (Dempster et al., 1977; Harvey, 1989; Harvey and Shephard, 1993; McLachlan and Krishnan, 2008).

The EM algorithm finds the maximum-likelihood estimates of the parameters in a MARSS model using an iterative process. Starting with an initial set of parameters[1], which we will denote $\hat{\Theta}_1$, an updated parameter set $\hat{\Theta}_2$ is obtaining by finding the $\hat{\Theta}_2$ that maximizes the expected value of the likelihood over the distribution of the states ($\mathbf{X}$) conditioned on $\hat{\Theta}_1$:

$$\hat{\Theta}_2 = \arg\max_{\Theta} \quad \mathrm{E}_{\mathbf{X}|\hat{\Theta}_1}[\log L(\Theta|\mathbf{Y} = \mathbf{y}_1^T, \mathbf{X})] \tag{4.2}$$

Then using $\hat{\Theta}_2$ in place of $\hat{\Theta}_1$ in Equation (4.2), an updated parameter set $\hat{\Theta}_3$ is calculated. This is repeated until the expected log-likelihood stops increasing (or increases less than some set tolerance level).

Implementing this algorithm is straight-forward, hence its popularity.

1. Set an initial set of parameters, $\hat{\Theta}_1$
2. E step: using the model for the hidden states ($\mathbf{X}$) and $\hat{\Theta}_1$, calculate the expected values of $\mathbf{X}$ conditioned on all the data $\mathbf{y}_1^T$; this is `xtT` output by `MARSSkf`. Also calculate expected values of any functions of $\mathbf{X}$, $g(\mathbf{X})$, that appear in your expected log-likelihood function.

[1] You can choose these however you wish, however choosing something not too far off from the correct values will make the algorithm go faster.

3. M step: put those $\mathrm{E}(\mathbf{X}|\mathbf{Y}=\mathbf{y}_1^T,\hat{\Theta}_1)$ and $\mathrm{E}(g(\mathbf{X})|\mathbf{Y}=\mathbf{y}_1^T,\hat{\Theta}_1)$ into your expected log-likelihood function in place of $\mathbf{X}$ (and $g(\mathbf{X})$) and maximize with respect to Θ. This gives you $\hat{\Theta}_2$.
4. Repeat the E and M steps until the log likelihood stops increasing.

The EM equations used in the MARSS package (function `MARSSkem`) are described in Holmes (2012) and are extensions of those in Shumway and Stoffer (1982) and Ghahramani and Hinton (1996). Our EM algorithm is an extended version because our algorithm is for cases where there are constraints within the parameter matrices (shared values, diagonal structure, block-diagonal structure, ...), where there are fixed values within the parameter matrices, or where there may be 0s on the diagonal of $\mathbf{Q}$, $\mathbf{R}$ and Λ.

The EM algorithm is a hill-climbing algorithm and like all hill-climbing algorithms can get stuck on local maxima. The MARSS package includes a Monte-Carlo initial conditions searcher (function `MARSSmcinit`) based on Biernacki et al. (2003) to minimize this problem. EM algorithms are also known to get close to the maximum very quickly but then creep toward the absolute maximum. Once in the vicinity of the maximum, quasi-Newton methods find the absolute maximum much faster, but they can be sensitive to initial conditions and in practice, we have found the EM algorithm to be much faster for large problems.

4.4 Parametric and innovations bootstrapping

Bootstrapping can be used to construct frequentist confidence intervals on the parameter estimates (Stoffer and Wall, 1991) and to compute the small-sample AIC corrector for MARSS models (Cavanaugh and Shumway, 1997); the functions `MARSSparamCIs` and `MARSSaic` do these computations.

The `MARSSboot` function provides both parametric and innovations bootstrapping of MARSS models. The innovations bootstrap algorithm by Stoffer and Wall (1991) bootstraps the model residuals (the innovations). This is a semi-parametric bootstrap since is uses, partially, the maximum-likelihood parameter estimates. This algorithm cannot be used if there are missing values in the data. Also for short time series, it gives biased bootstraps because one cannot resample the first few innovations.

`MARSSboot` also provides a fully parametric bootstrap. This uses the maximum-likelihood MARSS parameters to simulate data from which bootstrap parameter estimates are obtained. Our research (Holmes and Ward, 2010) indicates that this provides unbiased bootstrap parameter estimates, and it works with datasets with missing values. Lastly, `MARSSboot` can also output parameters sampled from a numerically estimated Hessian matrix.

4.5 Simulation and forecasting

The `MARSSsimulate` function simulates from a fitted marssMLE object (e.g. output from a `MARSS()` call). It use the `mvrnorm` function to produce draws of the process and observation errors from multivariate normal distributions for each time step.

4.6 Model selection

The package provides the `MARSSaic` function for computing AIC, AICc and AICb. The latter is a small-sample corrector for autoregressive state-space models. The bias problem with AIC and AICc for short time-series data has been shown in Cavanaugh and Shumway (1997) and Holmes and Ward (2010). AIC and AICc tend to select overly complex MARSS models when the time-series data are short. AICb corrects this bias. The algorithm for a non-parametric AICb is given in Cavanaugh and Shumway (1997). Their algorithm uses the innovations bootstrap (Stoffer and Wall, 1991), which means it cannot be used when there are missing data. We added a parametric AICb (Holmes and Ward, 2010), which uses a parametric bootstrap. This algorithm allows one to compute AICb when there are missing data and it provides unbiased AIC even for short time series. See Holmes and Ward (2010) for discussion and testing of parametric AICb for MARSS models.

AICb is comprised of the familiar AIC fit term, $-2\log L$, plus a penalty term that is the mean difference between the log likelihood the data under the bootstrapped maximum-likelihood parameter estimates and the log likelihood of the data under the original maximum-likelihood parameter estimate:

$$AICb = -2\log \mathrm{L}(\hat{\Theta}|\mathbf{y}) + 2\left(\frac{1}{N_b}\sum_{i=1}^{N_b} -\log\frac{\mathrm{L}(\hat{\Theta}^*(i)|\mathbf{y})}{\mathrm{L}(\hat{\Theta}|\mathbf{y})}\right) \tag{4.3}$$

where $\hat{\Theta}$ is the maximum-likelihood parameter set under the original data $\mathbf{y}$, $\hat{\Theta}^*(i)$ is a maximum-likelihood parameter set estimated from the i-th bootstrapped data set $\mathbf{y}^*(i)$, and N_b is the number of bootstrap data sets. It is important to notice that the likelihood in the AICb equation is $\mathrm{L}(\hat{\Theta}^*|\mathbf{y})$ not $\mathrm{L}(\hat{\Theta}^*|\mathbf{y}^*)$. In other words, we are taking the average of the likelihood of the original data given the bootstrapped parameter sets.

5

Examples

In this chapter, we work through a series of short examples using the MARSS package functions. This chapter is oriented towards those who are already somewhat familiar with MARSS models and want to get started quickly. We provide little explanatory text. Those unfamiliar with MARSS models might want to start with the case studies.

In these examples, we will use the default `form="marxss"` argument for a `MARSS()` call. This specifies a MARSS model of the form:

$$\mathbf{x}_t = \mathbf{B}_t\mathbf{x}_{t-1} + \mathbf{u}_t + \mathbf{C}_t\mathbf{c}_t + \mathbf{w}_t, \text{ where } \mathbf{w}_t \sim \text{MVN}(0, \mathbf{Q}_t) \tag{5.1a}$$

$$\mathbf{y}_t = \mathbf{Z}_t\mathbf{x}_t + \mathbf{a}_t + \mathbf{D}_t\mathbf{d}_t + \mathbf{v}_t, \text{ where } \mathbf{v}_t \sim \text{MVN}(0, \mathbf{R}_t) \tag{5.1b}$$

$$\mathbf{x}_1 \sim \text{MVN}(\pi, \Lambda) \text{ or } \mathbf{x}_0 \sim \text{MVN}(\pi, \Lambda) \tag{5.1c}$$

For this chapter, we do not fit models with inputs (or covariates) thus **C**, **c**, **D** and **d** are set to zero. This is the default so we just leave them off the model specification. See chapter 6 for examples of including inputs in a MARSS model.

5.1 Fixed and estimated elements in parameter matrices

Suppose one has a MARSS model (Equation 5.1) with the following model parameter matrices:

$$\begin{bmatrix} x_{1,t} \\ x_{2,t} \end{bmatrix} = \begin{bmatrix} b_1 & 0.1 \\ b_2 & 2 \end{bmatrix} \begin{bmatrix} x_{1,t-1} \\ x_{2,t-1} \end{bmatrix} + \begin{bmatrix} u \\ u \end{bmatrix} + \begin{bmatrix} w_{1,t} \\ w_{2,t} \end{bmatrix}, \ \mathbf{w}_t \sim \text{MVN}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} q_1 & q_3 \\ q_3 & q_2 \end{bmatrix} \right)$$

$$\begin{bmatrix} y_{1,t} \\ y_{2,t} \\ y_{3,t} \end{bmatrix} = \begin{bmatrix} z_1 & 0 \\ z_2 & z_2 \\ 0 & 3 \end{bmatrix} \begin{bmatrix} x_{1,t} \\ x_{2,t} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} v_{1,t} \\ v_{2,t} \\ v_{3,t} \end{bmatrix}, \ \mathbf{v}_t \sim \text{MVN}\left(\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} r & 0 & 0 \\ 0 & r & 0 \\ 0 & 0 & 1 \end{bmatrix} \right)$$

$$\mathbf{x}_0 \sim \text{MVN}\left(\begin{bmatrix} \pi_1 \\ \pi_2 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right)$$

Notice how this model mixes fixed values, estimated values and shared values.

The MARSS model specification is a list with the names, `Z`, `A`, `R`, `B`, `U`, `Q`, `x0` and `V0`. Each element is matrix (class matrix) with the same dimensions as your model on paper. MARSS distinguishes between the estimated and fixed values by using list matrices in which you can have numeric and character elements. Numeric elements are fixed; character elements are names of things to be estimated. The model above would be specified as:

```
Z=matrix(list("z1","z2",0,0,"z2",3),3,2)
A=matrix(0,3,1)
R=matrix(list(0),3,3); diag(R)=c("r","r",1)
B=matrix(list("b1",0.1,"b2",2),2,2)
U=matrix(c("u","u"),2,1)
Q=matrix(c("q1","q3","q3","q2"),2,2)
x0=matrix(c("pi1","pi2"),2,1)
V0=diag(1,2)
model.gen=list(Z=Z,A=A,R=R,B=B,U=U,Q=Q,x0=x0,V0=V0)
```

Notice that there is a one-to-one correspondence between the model list in R and the model on paper. Fitting the model is then just a matter of passing the data and model list to the MARSS function:

```
kemfit = MARSS(dat, model=model.gen)
```

If you work often with MARSS models then you will probably know whether prior sensitivity is a problem for your types of MARSS applications. If so, note that the MARSS package is unusual in that it allows you to set $\Lambda = 0$ and treat $\mathbf{x}_0$ as an unknown estimated parameter. This eliminates the prior and thus the prior sensitivity problems—at the cost of adding m parameters. Depending on your application, you may need to set the initial conditions at $t = 1$ instead of the default of $t = 0$. If you are unsure, look in the index and read all the sections that talk about troubleshooting priors.

5.2 Different numbers of state processes

Here we show a series of short examples using a dataset on Washington harbor seals (`?harborSealWA`), which has five observation time series. The dataset is a little unusual in that it has four missing years from years 2 to 5. This causes some interesting issues with prior specification. Before starting the harbor seal examples, we set up the data, making time go across the columns and removing the year column:

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

5.2.1 One hidden state process for each observation time series

This is the default model for the `MARSS()` function. In this case, $n = m$, the observation errors are i.i.d. and the process errors are independent and have different variances. The elements in **u** are all different (meaning, they are not forced to be the same). Mathematically, the MARSS model being fit is:

$$\begin{bmatrix} x_{1,t} \\ x_{2,t} \\ x_{3,t} \\ x_{4,t} \\ x_{5,t} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{1,t-1} \\ x_{2,t-1} \\ x_{3,t-1} \\ x_{4,t-1} \\ x_{5,t-1} \end{bmatrix} + \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \end{bmatrix} + \begin{bmatrix} w_{1,t} \\ w_{2,t} \\ w_{3,t} \\ w_{4,t} \\ w_{5,t} \end{bmatrix}, \ \mathbf{w}_t \sim \text{MVN}\left(\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} q_1 & 0 & 0 & 0 & 0 \\ 0 & q_2 & 0 & 0 & 0 \\ 0 & 0 & q_3 & 0 & 0 \\ 0 & 0 & 0 & q_4 & 0 \\ 0 & 0 & 0 & 0 & q_5 \end{bmatrix} \right)$$

$$\begin{bmatrix} y_{1,t} \\ y_{2,t} \\ y_{3,t} \\ y_{4,t} \\ y_{5,t} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{1,t} \\ x_{2,t} \\ x_{3,t} \\ x_{4,t} \\ x_{5,t} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} v_{1,t} \\ v_{2,t} \\ v_{3,t} \\ v_{4,t} \\ v_{5,t} \end{bmatrix}, \ \mathbf{v}_t \sim \text{MVN}\left(\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} r & 0 & 0 & 0 & 0 \\ 0 & r & 0 & 0 & 0 \\ 0 & 0 & r & 0 & 0 \\ 0 & 0 & 0 & r & 0 \\ 0 & 0 & 0 & 0 & r \end{bmatrix} \right)$$

This is the default model, so you can fit it by simply passing `dat` to `MARSS()`.

```
kemfit = MARSS(dat)
```

```
Success! abstol and log-log tests passed at 38 iterations.
Alert: conv.test.slope.tol is 0.5.
Test with smaller values (<0.1) to ensure convergence.

MARSS fit is
Estimation method: kem
Convergence test: conv.test.slope.tol = 0.5, abstol = 0.001
Estimation converged in 38 iterations.
Log-likelihood: 19.13428
AIC: -6.268557   AICc: 3.805517

                Estimate
R.diag           0.00895
U.SJF            0.06839
U.SJI            0.07163
U.EBays          0.04179
U.PSnd           0.05226
U.HC            -0.00279
Q.(SJF,SJF)      0.03205
Q.(SJI,SJI)      0.01098
Q.(EBays,EBays)  0.00706
Q.(PSnd,PSnd)    0.00414
```

```
Q.(HC,HC)        0.05450
x0.SJF           5.98647
x0.SJI           6.72487
x0.EBays         6.66212
x0.PSnd          5.83969
x0.HC            6.60482

Standard errors have not been calculated.
Use MARSSparamCIs to compute CIs and bias estimates.
```

The output warns you that the convergence tolerance is high. You can set it lower by passing in `control=list(conv.test.slope.tol=0.1)`. `MARSS()` is automatically creating parameter names since you did not tell it the names. To see exactly where each parameter element appears in its parameter matrix, type `summary(kemfit$model)`.

Though it is not necessary to specify the model for this example since it is the default, here is how you could do so using matrices:

```
B=Z=diag(1,5)
U=matrix(c("u1","u2","u3","u4","u5"),5,1)
x0=A=matrix(0,5,1)
R=Q=matrix(list(0),5,5)
diag(R)="r"
diag(Q)=c("q1","q2","q3","q4","q5")
```

Notice that there is a one-to-one relationship between the model on paper and the model specification for MARSS. Notice also that when a matrix has both fixed and estimated elements (like **R** and **Q**), a list matrix is used to allow you to specify the fixed elements as numeric and to give the estimated elements character names.

The default MLE method is the EM algorithm (`method="kem"`). You can also use a quasi-Newton method (BFGS) by setting `method="BFGS"`.

```
kemfit.bfgs = MARSS(dat, method="BFGS")
```

```
Success! Converged in 99 iterations.
Function MARSSkfas used for likelihood calculation.

MARSS fit is
Estimation method: BFGS
Estimation converged in 99 iterations.
Log-likelihood: 19.13936
AIC: -6.278712   AICc: 3.795362

                Estimate
R.diag           0.00849
U.SJF            0.06838
```

```
U.SJI            0.07152
U.EBays          0.04188
U.PSnd           0.05233
U.HC            -0.00271
Q.(SJF,SJF)      0.03368
Q.(SJI,SJI)      0.01124
Q.(EBays,EBays)  0.00722
Q.(PSnd,PSnd)    0.00437
Q.(HC,HC)        0.05600
x0.SJF           5.98437
x0.SJI           6.72169
x0.EBays         6.65689
x0.PSnd          5.83527
x0.HC            6.60425

Standard errors have not been calculated.
Use MARSSparamCIs to compute CIs and bias estimates.
```

Using the default EM convergence criteria, the EM algorithm stops at a log-likelihood a little lower than the BFGS algorithm does, but the EM algorithm was faster, 10 times faster, in this case. If you wanted to use the EM fit as the initial conditions, pass in the `inits` argument using the `$par` element (or `coef(fit,form="marss")`) of the EM fit.

```
kemfit.bfgs2 = MARSS(dat, method="BFGS", inits=kemfit$par)
```

The BFGS algorithm now converges in 103 iterations. Output not shown.

We mentioned that the missing years from year 2 to 4 creates an interesting issue with the prior specification. The default behavior of MARSS is to treat the initial state as at $t = 0$ instead of $t = 1$. Usually this doesn't make a difference, but for this dataset, if we set the prior at $t = 1$, the MLE estimate of $\mathbf{R}$ becomes 0. If we estimate $\mathbf{x}_1$ as a parameter and let $\mathbf{R}$ go to 0, the likelihood will go to infinity (slowly but surely). This is neither an error nor a pathology, but is probably not what you would like to have happen. Note that the "BFGS" algorithm will not find the maximum in this case; it will stop before $\mathbf{R}$ gets small and the likelihood gets very large. However, the EM algorithm will climb up the peak. You can try it by running the following code. It will report warnings which you can read about in Appendix B.

```
kemfit.strange = MARSS(dat, model=list(tinitx=1))
```

5.2.2 Five correlated hidden state processes

This is the same model except that the hidden states have temporally correlated process errors. Mathematically, this is the model:

$$\begin{bmatrix} x_{1,t} \\ x_{2,t} \\ x_{3,t} \\ x_{4,t} \\ x_{5,t} \end{bmatrix} = \begin{bmatrix} x_{1,t-1} \\ x_{2,t-1} \\ x_{3,t-1} \\ x_{4,t-1} \\ x_{5,t-1} \end{bmatrix} + \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \end{bmatrix} + \begin{bmatrix} w_{1,t} \\ w_{2,t} \\ w_{3,t} \\ w_{4,t} \\ w_{5,t} \end{bmatrix}, \ \mathbf{w}_t \sim \text{MVN}\left(0, \begin{bmatrix} q_1 & c_{1,2} & c_{1,3} & c_{1,4} & c_{1,5} \\ c_{1,2} & q_2 & c_{2,3} & c_{2,4} & c_{2,5} \\ c_{1,3} & c_{2,3} & q_3 & c_{3,4} & c_{3,5} \\ c_{1,4} & c_{2,4} & c_{3,4} & q_4 & c_{4,5} \\ c_{1,5} & c_{2,5} & c_{3,5} & c_{4,5} & q_5 \end{bmatrix}\right)$$

$$\begin{bmatrix} y_{1,t} \\ y_{2,t} \\ y_{3,t} \\ y_{4,t} \\ y_{5,t} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{1,t} \\ x_{2,t} \\ x_{3,t} \\ x_{4,t} \\ x_{5,t} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} v_{1,t} \\ v_{2,t} \\ v_{3,t} \\ v_{4,t} \\ v_{5,t} \end{bmatrix}, \ \mathbf{v}_t \sim \text{MVN}\left(0, \begin{bmatrix} r & 0 & 0 & 0 & 0 \\ 0 & r & 0 & 0 & 0 \\ 0 & 0 & r & 0 & 0 \\ 0 & 0 & 0 & r & 0 \\ 0 & 0 & 0 & 0 & r \end{bmatrix}\right)$$

B is not shown in the top equation; it is a $m \times m$ identity matrix. To fit, use `MARSS()` with the `model` argument set. The output is not shown but it will appear if you type this on the R command line.

```
kemfit = MARSS(dat, model=list(Q="unconstrained"))
```

This shows one of the text shortcuts, `"unconstrained"`, which means estimate all elements in the matrix. This shortcut can be used for all parameter matrices.

5.2.3 Five equally correlated hidden state processes

This is the same model except that now there is only one process error variance and one process error covariance. Mathematically, the model is:

$$\begin{bmatrix} x_{1,t} \\ x_{2,t} \\ x_{3,t} \\ x_{4,t} \\ x_{5,t} \end{bmatrix} = \begin{bmatrix} x_{1,t-1} \\ x_{2,t-1} \\ x_{3,t-1} \\ x_{4,t-1} \\ x_{5,t-1} \end{bmatrix} + \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \end{bmatrix} + \begin{bmatrix} w_{1,t} \\ w_{2,t} \\ w_{3,t} \\ w_{4,t} \\ w_{5,t} \end{bmatrix}, \ \mathbf{w}_t \sim \text{MVN}\left(0, \begin{bmatrix} q & c & c & c & c \\ c & q & c & c & c \\ c & c & q & c & c \\ c & c & c & q & c \\ c & c & c & c & q \end{bmatrix}\right)$$

$$\begin{bmatrix} y_{1,t} \\ y_{2,t} \\ y_{3,t} \\ y_{4,t} \\ y_{5,t} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{1,t} \\ x_{2,t} \\ x_{3,t} \\ x_{4,t} \\ x_{5,t} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} v_{1,t} \\ v_{2,t} \\ v_{3,t} \\ v_{4,t} \\ v_{5,t} \end{bmatrix}, \ \mathbf{v}_t \sim \text{MVN}\left(0, \begin{bmatrix} r & 0 & 0 & 0 & 0 \\ 0 & r & 0 & 0 & 0 \\ 0 & 0 & r & 0 & 0 \\ 0 & 0 & 0 & r & 0 \\ 0 & 0 & 0 & 0 & r \end{bmatrix}\right)$$

Again **B** is not shown in the top equation; it is a $m \times m$ identity matrix. To fit, use the following code (output not shown):

```
kemfit = MARSS(dat, model=list(Q="equalvarcov"))
```

The shortcut '`"equalvarcov"` means one value on the diagonal and one on the off-diagonal. It can be used for all square matrices (**B**,**Q**,**R**, and Λ).

5.2.4 Five hidden state processes with a "north" and a "south" u and Q elements

Here we fit a model with five independent hidden states where each observation time series is an independent observation of a different hidden trajectory but the hidden trajectories 1-3 share their **u** and **Q** elements, while hidden trajectories 4-5 share theirs. This is the model:

$$\begin{bmatrix} x_{1,t} \\ x_{2,t} \\ x_{3,t} \\ x_{4,t} \\ x_{5,t} \end{bmatrix} = \begin{bmatrix} x_{1,t-1} \\ x_{2,t-1} \\ x_{3,t-1} \\ x_{4,t-1} \\ x_{5,t-1} \end{bmatrix} + \begin{bmatrix} u_n \\ u_n \\ u_n \\ u_s \\ u_s \end{bmatrix} + \begin{bmatrix} w_{1,t} \\ w_{2,t} \\ w_{3,t} \\ w_{4,t} \\ w_{5,t} \end{bmatrix}, \ \mathbf{w}_t \sim \text{MVN}\left(0, \begin{bmatrix} q_n & 0 & 0 & 0 & 0 \\ 0 & q_n & 0 & 0 & 0 \\ 0 & 0 & q_n & 0 & 0 \\ 0 & 0 & 0 & q_s & 0 \\ 0 & 0 & 0 & 0 & q_s \end{bmatrix}\right)$$

$$\begin{bmatrix} y_{1,t} \\ y_{2,t} \\ y_{3,t} \\ y_{4,t} \\ y_{5,t} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{1,t} \\ x_{2,t} \\ x_{3,t} \\ x_{4,t} \\ x_{5,t} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} v_{1,t} \\ v_{2,t} \\ v_{3,t} \\ v_{4,t} \\ v_{5,t} \end{bmatrix}, \ \mathbf{v}_t \sim \text{MVN}\left(0, \begin{bmatrix} r & 0 & 0 & 0 & 0 \\ 0 & r & 0 & 0 & 0 \\ 0 & 0 & r & 0 & 0 \\ 0 & 0 & 0 & r & 0 \\ 0 & 0 & 0 & 0 & r \end{bmatrix}\right)$$

To fit use the following code, we specify the `model` argument for **u** and **Q** using list matrices. List matrices allow us to combine numeric and character values in a matrix. MARSS will interpret the numeric values as fixed, and the character values as parameters to be estimated. Parameters with the same name are constrained to be identical.

```
regions=list("N","N","N","S","S")
U=matrix(regions,5,1)
Q=matrix(list(0),5,5); diag(Q)=regions
kemfit = MARSS(dat, model=list(U=U, Q=Q))
```

Only **u** and **Q** need to be specified since the other parameters are at their default values.

5.2.5 Fixed observation error variance

Here we fit the same model but with a known observation error variance. This is the model:

$$\begin{bmatrix} x_{1,t} \\ x_{2,t} \\ x_{3,t} \\ x_{4,t} \\ x_{5,t} \end{bmatrix} = \begin{bmatrix} x_{1,t-1} \\ x_{2,t-1} \\ x_{3,t-1} \\ x_{4,t-1} \\ x_{5,t-1} \end{bmatrix} + \begin{bmatrix} u_n \\ u_n \\ u_n \\ u_s \\ u_s \end{bmatrix} + \begin{bmatrix} w_{1,t} \\ w_{2,t} \\ w_{3,t} \\ w_{4,t} \\ w_{5,t} \end{bmatrix}, \ \mathbf{w}_t \sim \text{MVN}\left(0, \begin{bmatrix} q_n & 0 & 0 & 0 & 0 \\ 0 & q_n & 0 & 0 & 0 \\ 0 & 0 & q_n & 0 & 0 \\ 0 & 0 & 0 & q_s & 0 \\ 0 & 0 & 0 & 0 & q_s \end{bmatrix}\right)$$

$$\begin{bmatrix} y_{1,t} \\ y_{2,t} \\ y_{3,t} \\ y_{4,t} \\ y_{5,t} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{1,t} \\ x_{2,t} \\ x_{3,t} \\ x_{4,t} \\ x_{5,t} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} v_{1,t} \\ v_{2,t} \\ v_{3,t} \\ v_{4,t} \\ v_{5,t} \end{bmatrix},$$

$$\mathbf{v}_t \sim \text{MVN}\left(0, \begin{bmatrix} 0.01 & 0 & 0 & 0 & 0 \\ 0 & 0.01 & 0 & 0 & 0 \\ 0 & 0 & 0.01 & 0 & 0 \\ 0 & 0 & 0 & 0.01 & 0 \\ 0 & 0 & 0 & 0 & 0.01 \end{bmatrix}\right)$$

To fit this model, use the following code (output not shown):

```
regions=list("N","N","N","S","S")
U=matrix(regions,5,1)
Q=matrix(list(0),5,5); diag(Q)=regions
R=diag(0.01,5)
kemfit = MARSS(dat, model=list(U=U, Q=Q, R=R))
```

5.2.6 One hidden state and five i.i.d. observation time series

Instead of five hidden state trajectories, we specify that there is only one and all the observations are of that one trajectory. Mathematically, the model is:

$$x_t = x_{t-1} + u + w_t, \ w_t \sim \text{N}(0, q)$$

$$\begin{bmatrix} y_{1,t} \\ y_{2,t} \\ y_{3,t} \\ y_{4,t} \\ y_{5,t} \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} x_t + \begin{bmatrix} 0 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix} + \begin{bmatrix} v_{1,t} \\ v_{2,t} \\ v_{3,t} \\ v_{4,t} \\ v_{5,t} \end{bmatrix}, \ \mathbf{v}_t \sim \text{MVN}\left(0, \begin{bmatrix} r & 0 & 0 & 0 & 0 \\ 0 & r & 0 & 0 & 0 \\ 0 & 0 & r & 0 & 0 \\ 0 & 0 & 0 & r & 0 \\ 0 & 0 & 0 & 0 & r \end{bmatrix}\right)$$

Note the default model for **R** is `"diagonal and equal"`' so we can leave this off when specifying the `model` argument. To fit, use this code (output not shown):

```
Z=factor(c(1,1,1,1,1))
kemfit = MARSS(dat, model=list(Z=Z))
```

```
Success! abstol and log-log tests passed at 28 iterations.
Alert: conv.test.slope.tol is 0.5.
Test with smaller values (<0.1) to ensure convergence.

MARSS fit is
Estimation method: kem
Convergence test: conv.test.slope.tol = 0.5, abstol = 0.001
Estimation converged in 28 iterations.
Log-likelihood: 3.593276
AIC: 8.813447   AICc: 11.13603

        Estimate
A.SJI    0.80153
A.EBays  0.28245
A.PSnd  -0.54802
A.HC    -0.62665
R.diag   0.04523
U.U      0.04759
Q.Q      0.00429
x0.x0    6.39199

Standard errors have not been calculated.
Use MARSSparamCIs to compute CIs and bias estimates.
```

You can also pass in **Z** exactly as it is in the equation: `Z=matrix(1,5,1)`, but the factor shorthand is handy if you need to assign different observed time series to different underlying state time series (see next examples). The default **a** form is `"scaling"`, which means that the first **y** row associated with a given x has $a = 0$ and the rest are estimated.

5.2.7 One hidden state and five independent observation time series with different variances

Mathematically, this model is:

$$x_t = x_{t-1} + u + w_t,\ w_t \sim \mathrm{N}(0, q)$$

$$\begin{bmatrix} y_{1,t} \\ y_{2,t} \\ y_{3,t} \\ y_{4,t} \\ y_{5,t} \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} x_t + \begin{bmatrix} 0 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix} + \begin{bmatrix} v_{1,t} \\ v_{2,t} \\ v_{3,t} \\ v_{4,t} \\ v_{5,t} \end{bmatrix},\ \mathbf{v}_t \sim \mathrm{MVN}\left(0, \begin{bmatrix} r_1 & 0 & 0 & 0 & 0 \\ 0 & r_2 & 0 & 0 & 0 \\ 0 & 0 & r_3 & 0 & 0 \\ 0 & 0 & 0 & r_4 & 0 \\ 0 & 0 & 0 & 0 & r_5 \end{bmatrix}\right)$$

To fit this model:

```
Z=factor(c(1,1,1,1,1))
R="diagonal and unequal"
kemfit = MARSS(dat, model=list(Z=Z, R=R))
```

```
Success! abstol and log-log tests passed at 24 iterations.
Alert: conv.test.slope.tol is 0.5.
Test with smaller values (<0.1) to ensure convergence.

MARSS fit is
Estimation method: kem
Convergence test: conv.test.slope.tol = 0.5, abstol = 0.001
Estimation converged in 24 iterations.
Log-likelihood: 16.66199
AIC: -9.323982   AICc: -3.944671

                Estimate
A.SJI            0.79555
A.EBays          0.27540
A.PSnd          -0.53694
A.HC            -0.60874
R.(SJF,SJF)      0.03229
R.(SJI,SJI)      0.03528
R.(EBays,EBays)  0.01352
R.(PSnd,PSnd)    0.01082
R.(HC,HC)        0.19609
U.U              0.05270
Q.Q              0.00604
x0.x0            6.26676

Standard errors have not been calculated.
Use MARSSparamCIs to compute CIs and bias estimates.
```

5.2.8 Two hidden state processes

Here we fit a model with two hidden states (north and south) where observation time series 1-3 are for the north and 4-5 are for the south. We make the hidden state processes independent (meaning a diagonal **Q** matrix) but with the same process variance. We make the observation errors i.i.d. (the default) and the **u** elements equal. Mathematically, this is the model:

$$\begin{bmatrix} x_{n,t} \\ x_{s,t} \end{bmatrix} = \begin{bmatrix} x_{n,t-1} \\ x_{s,t-1} \end{bmatrix} + \begin{bmatrix} u \\ u \end{bmatrix} + \begin{bmatrix} w_{n,t} \\ w_{s,t} \end{bmatrix}, \ \mathbf{w}_t \sim \text{MVN}\left(0, \begin{bmatrix} q & 0 \\ 0 & q \end{bmatrix}\right)$$

$$\begin{bmatrix} y_{1,t} \\ y_{2,t} \\ y_{3,t} \\ y_{4,t} \\ y_{5,t} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_{n,t} \\ x_{s,t} \end{bmatrix} + \begin{bmatrix} 0 \\ a_2 \\ a_3 \\ 0 \\ a_5 \end{bmatrix} + \begin{bmatrix} v_{1,t} \\ v_{2,t} \\ v_{3,t} \\ v_{4,t} \\ v_{5,t} \end{bmatrix}, \ \mathbf{v}_t \sim \text{MVN}\left(0, \begin{bmatrix} r & 0 & 0 & 0 & 0 \\ 0 & r & 0 & 0 & 0 \\ 0 & 0 & r & 0 & 0 \\ 0 & 0 & 0 & r & 0 \\ 0 & 0 & 0 & 0 & r \end{bmatrix}\right)$$

To fit the model, use the following code (output not shown):

```
Z=factor(c("N","N","N","S","S"))
Q="diagonal and equal"
U="equal"
kemfit = MARSS(dat, model=list(Z=Z,Q=Q,U=U))
```

You can also pass in **Z** exactly as it is in the equation as a numeric matrix (`matrix(1,5,1)`; the `factor` notation is a shortcut for making a design matrix (as **Z** is in these examples). `"equal"` is a shortcut meaning all elements in a matrix are constrained to be equal. It can be used for all column matrices (**a**, **u** and π). `"diagonal and equal"` can be used as a shortcut for all square matrices (**B**,**Q**,**R**, and Λ).

5.3 Time-varying parameters

Time-varying parameters are specified by passing in an array of matrices (list, numeric or character) where the 3rd dimension of the array is time and must be the same value as the 2nd (time) dimension of the data matrix. No text shortcuts are allowed for time-varying parameters; you need to use the matrix form.

For example, let's say we wanted a different **u** for the first half versus second half of the harbor seal time series. We would pass in an array for **u** as follows:

```
U1=matrix("t1",5,1); U2=matrix("t2",5,1)
Ut=array(U2,dim=c(dim(U1),dim(dat)[2]))
TT=dim(dat)[2]
Ut[,,1:floor(TT/2)]=U1
kemfit.tv=MARSS(dat,model=list(U=Ut,Q="diagonal and equal"))
```

You can have some elements in a parameter matrix be time-constant and some be time-varying:

```
U1=matrix(c(rep("t1",4),"hc"),5,1); U2=matrix(c(rep("t2",4),"hc"),5,1)
Ut=array(U2,dim=c(dim(U1),dim(dat)[2]))
```

```
Ut[,,1:floor(TT/2)]=U1
kemfit.tv=MARSS(dat,model=list(U=Ut,Q="diagonal and equal"))
```

Note that how the time-varying model is specified for MARSS is the same as you would write the time-varying model on paper in matrix math form.

5.4 Printing and summarizing models and model fits

The package includes print functions for marssMODEL objects (model specification) and marssMLE objects (fitted models).

```
print(kemfit)
print(kemfit$model)
```

This will print the basic information on model structure and model fit that you have seen in the previous examples.

The package also includes a summary function for models.

```
summary(kemfit$model)
```

Output is not shown because it is verbose, but it prints each matrix with the fixed elements denoted with their values and the free elements denoted by their names. This is very helpful for confirming exactly what model structure you are fitting to the data.

The print function will also print various other things like a vector of the estimated parameters, the estimated states, the state standard errors, etc., using the `what` argument in the print call:

```
print(kemfit, what="par")
```

```
List of the estimated values in each parameter matrix
$Z
     [,1]

$A
              [,1]
SJI    0.79786453
EBays  0.27743474
HC    -0.07035021

$R
            [,1]
diag 0.03406192

$B
     [,1]
```

```
$U
        [,1]
1 0.04317641

$Q
            [,1]
diag 0.007669608

$x0
      [,1]
N 6.172048
S 6.206155

$V0
     [,1]

$C
     [,1]

$D
     [,1]

$c
     [,1]

$d
     [,1]
```

```
print(kemfit, what="Q")
```

```
Parameter matrix Q
            [,1]        [,2]
[1,] 0.007669608 0.000000000
[2,] 0.000000000 0.007669608
```

Type `?print.MARSS` to see a list of the types of output that can be printed with a `print` call. If you want to use the output from print instead of printing, then assign the print call to a value:

```
x=print(kemfit, what="states")
```

```
Smoothed state estimates
      [,1]     [,2]     [,3]     [,4]     [,5]     [,6]
N 6.215483 6.329702 6.443921 6.558140 6.672359 6.786578
S 6.249445 6.295591 6.341736 6.387881 6.434027 6.480172
      [,7]     [,8]     [,9]    [,10]    [,11]    [,12]
N 6.904124 6.944425 6.976697 7.050053 7.156567 7.198947
```

```
S 6.526317 6.572463 6.613358 6.654252 6.695147 6.736042
     [,13]    [,14]    [,15]    [,16]    [,17]    [,18]
N 7.228397 7.293141 7.380439 7.467975 7.488458 7.541996
S 6.776937 6.817832 6.786202 6.764235 6.786233 6.816405
     [,19]    [,20]    [,21]    [,22]
N 7.561182 7.524175 7.475514 7.459263
S 6.846578 6.813743 6.791537 6.819195
```

```
x
```

```
      [,1]     [,2]     [,3]     [,4]     [,5]     [,6]
N 6.215483 6.329702 6.443921 6.558140 6.672359 6.786578
S 6.249445 6.295591 6.341736 6.387881 6.434027 6.480172
      [,7]     [,8]     [,9]    [,10]    [,11]    [,12]
N 6.904124 6.944425 6.976697 7.050053 7.156567 7.198947
S 6.526317 6.572463 6.613358 6.654252 6.695147 6.736042
     [,13]    [,14]    [,15]    [,16]    [,17]    [,18]
N 7.228397 7.293141 7.380439 7.467975 7.488458 7.541996
S 6.776937 6.817832 6.786202 6.764235 6.786233 6.816405
     [,19]    [,20]    [,21]    [,22]
N 7.561182 7.524175 7.475514 7.459263
S 6.846578 6.813743 6.791537 6.819195
```

5.5 Confidence intervals on a fitted model

The function `MARSSparamCIs()` is used to compute confidence intervals with a default alpha level of 0.05. The function can compute approximate confidence intervals using a numerically estimated Hessian matrix (`method="hessian"`) or via parametric (`method="parametric"`) or non-parametric (`method="innovations"`) bootstrapping.

5.5.1 Approximate confidence intervals from a Hessian matrix

The default method for `MARSSparamCIs` is to use a numerically estimated Hessian matrix:

```
kem.with.hess.CIs = MARSSparamCIs(kemfit)
```

Use `print` or just type the marssMLE object name to see the confidence intervals:

```
print(kem.with.hess.CIs)
```

```
MARSS fit is
Estimation method: kem
Convergence test: conv.test.slope.tol = 0.5, abstol = 0.001
```

```
Estimation converged in 22 iterations.
Log-likelihood: 7.949236
AIC: 0.1015284   AICc: 2.424109

          ML.Est Std.Err   low.CI  up.CI
A.SJI    0.79786  0.0615  0.67729 0.9184
A.EBays  0.27743  0.0625  0.15487 0.4000
A.HC    -0.07035  0.0888 -0.24447 0.1038
R.diag   0.03406  0.0175  0.02256 0.0479
U.1      0.04318  0.0144  0.01500 0.0714
Q.diag   0.00767  0.0235  0.00173 0.0179
x0.N     6.17205  0.1455  5.88696 6.4571
x0.S     6.20615  0.1571  5.89827 6.5140

CIs calculated at alpha = 0.05 via method=hessian
```

The Hessian matrix is an estimate of the variance-covariance matrix of the parameter estimates. For the variances, **Q** and **R**, the normality assumption is not so good because they are constrained to be positive. Thus you may see lower confidence intervals on variances that are negative using the Hessian approximation.

5.5.2 Confidence intervals from a parametric bootstrap

Use `method="parametric"` to use a parametric bootstrap to compute confidence intervals and bias using a parametric bootstrap.

```
kem.w.boot.CIs=MARSSparamCIs(kemfit,method="parametric",nboot=10)
#nboot should be more like 1000, but set low for example's sake
print(kem.w.boot.CIs)
```

```
MARSS fit is
Estimation method: kem
Convergence test: conv.test.slope.tol = 0.5, abstol = 0.001
Estimation converged in 22 iterations.
Log-likelihood: 7.949236
AIC: 0.1015284   AICc: 2.424109

          ML.Est Std.Err   low.CI   up.CI Est.Bias
A.SJI    0.79786 0.05693  0.72022 0.88044  0.00794
A.EBays  0.27743 0.06916  0.19855 0.39620  0.00505
A.HC    -0.07035 0.13502 -0.30625 0.09153 -0.00113
R.diag   0.03406 0.00521  0.02738 0.04259  0.00105
U.1      0.04318 0.01651  0.00502 0.05559  0.00963
Q.diag   0.00767 0.00413  0.00000 0.00986  0.00465
x0.N     6.17205 0.17218  5.87776 6.36076  0.01429
```

```
x0.S     6.20615 0.27183  6.08383 6.89110 -0.17488
        Unbias.Est
A.SJI       0.8058
A.EBays     0.2825
A.HC       -0.0715
R.diag      0.0351
U.1         0.0528
Q.diag      0.0123
x0.N        6.1863
x0.S        6.0313

CIs calculated at alpha = 0.05 via method=parametric
Bias calculated via parametric bootstrapping with 10 bootstraps.
```

5.6 Vectors of just the estimated parameters

Often it is useful to have a vector of the estimated parameters. For example, if you are writing a call to `optim`, you will need a vector of just the estimated parameters. You can use the function `coef`:

```
parvec=coef(kemfit, type="vector")
parvec
```

```
      A.SJI      A.EBays         A.HC       R.diag
0.797864531  0.277434738 -0.070350207  0.034061922
        U.1       Q.diag         x0.N         x0.S
0.043176408  0.007669608  6.172047633  6.206154697
```

5.7 Kalman filter and smoother output

All the standard Kalman filter and smoother output (along with the lag-one covariance smoother output) is available using the `MARSSkf` function. Read the help file (`?MARSSkf`) for details and meanings of the names in the output list.

```
kf=MARSSkf(kemfit)
names(kf)
```

```
 [1] "xtT"        "VtT"        "Vtt1T"      "x0T"
 [5] "V0T"        "x10T"       "V10T"       "x00T"
 [9] "V00T"       "Vtt"        "Vtt1"       "J"
[13] "J0"         "Kt"         "xtt1"       "xtt"
[17] "Innov"      "Sigma"      "kfas.model" "logLik"
[21] "ok"         "errors"
```

```
#if you only need the logLik, this is the fast way to get it
MARSSkf(kemfit, only.logLik=TRUE)
```

5.8 Degenerate variance estimates

If your data are short relative to the number of parameters you are estimating, then you are liable to find that some of the variance elements are degenerate (equal to zero). Try the following:

```
dat.short = dat[1:4,1:10]
kem.degen = MARSS(dat.short,control=list(allow.degen=FALSE))
```

```
Warning! Abstol convergence only. Maxit (=500) reached before log-log convergence.

MARSS fit is
Estimation method: kem
Convergence test: conv.test.slope.tol = 0.5, abstol = 0.001
WARNING: abstol convergence only no log-log convergence.
 maxit reached at 500 iter before log-log convergence.
 The likelihood and params might not be at the ML values.
 Try setting control$maxit higher.
Log-likelihood: 11.67854
AIC: 2.642914   AICc: 63.30958

                Estimate
R.diag          1.22e-02
U.SJF           9.79e-02
U.SJI           1.09e-01
U.EBays         9.28e-02
U.PSnd          1.11e-01
Q.(SJF,SJF)     1.89e-02
Q.(SJI,SJI)     1.03e-05
Q.(EBays,EBays) 8.24e-06
Q.(PSnd,PSnd)   3.05e-05
x0.SJF          5.96e+00
x0.SJI          6.73e+00
x0.EBays        6.60e+00
x0.PSnd         5.71e+00

Standard errors have not been calculated.
Use MARSSparamCIs to compute CIs and bias estimates.

Convergence warnings
 Warning: the  Q.(SJI,SJI)  parameter value has not converged.
 Warning: the  Q.(EBays,EBays)  parameter value has not converged.
 Warning: the  Q.(PSnd,PSnd)  parameter value has not converged.
 Type MARSSinfo("convergence") for more info on this warning.
```

This will print a warning that the maximum number of iterations was reached before convergence of some of the **Q** parameters. It might be that if you just

ran a few more iterations the variances will converge. So first try setting `control$maxit` higher.

```
kem.degen2 = MARSS(dat.short, control=list(maxit=1000,
      allow.degen=FALSE), silent=2)
```

Output not shown, but if you run the code, you will see that some of the **Q** terms are still not converging. MARSS can detect if a variance is going to zero and it will try zero to see if that has a higher likelihood. Try removing the `allow.degen=FALSE` which was turning off this feature.

```
kem.short = MARSS(dat.short)
```

```
Warning! Abstol convergence only. Maxit (=500) reached before log-log convergence.

MARSS fit is
Estimation method: kem
Convergence test: conv.test.slope.tol = 0.5, abstol = 0.001
WARNING: abstol convergence only no log-log convergence.
 maxit reached at 500 iter before log-log convergence.
 The likelihood and params might not be at the ML values.
 Try setting control$maxit higher.
Log-likelihood: 11.6907
AIC: 2.6186   AICc: 63.28527

                Estimate
R.diag          1.22e-02
U.SJF           9.79e-02
U.SJI           1.09e-01
U.EBays         9.24e-02
U.PSnd          1.11e-01
Q.(SJF,SJF)     1.89e-02
Q.(SJI,SJI)     1.03e-05
Q.(EBays,EBays) 0.00e+00
Q.(PSnd,PSnd)   3.04e-05
x0.SJF          5.96e+00
x0.SJI          6.73e+00
x0.EBays        6.60e+00
x0.PSnd         5.71e+00

Standard errors have not been calculated.
Use MARSSparamCIs to compute CIs and bias estimates.

Convergence warnings
 Warning: the  Q.(SJI,SJI)  parameter value has not converged.
 Warning: the  Q.(PSnd,PSnd)  parameter value has not converged.
 Type MARSSinfo("convergence") for more info on this warning.
```

So three of the four **Q** elements are going to zero. This often happens when you do not have enough data to estimate both observation and process variance.

Perhaps we are trying to estimate too many variances. We can try using only one variance value in **Q** and one u value in **u**:

```
kem.small=MARSS(dat.short,model=list(Q="diagonal and equal",
   U="equal"))
```

```
Success! abstol and log-log tests passed at 164 iterations.
Alert: conv.test.slope.tol is 0.5.
Test with smaller values (<0.1) to ensure convergence.

MARSS fit is
Estimation method: kem
Convergence test: conv.test.slope.tol = 0.5, abstol = 0.001
Estimation converged in 164 iterations.
Log-likelihood: 11.19
AIC: -8.379994   AICc: 0.9533396

         Estimate
R.diag     0.0191
U.1        0.1027
Q.diag     0.0000
x0.SJF     6.0609
x0.SJI     6.7698
x0.EBays   6.5307
x0.PSnd    5.7451

Standard errors have not been calculated.
Use MARSSparamCIs to compute CIs and bias estimates.
```

No, there are simply not enough data to estimate both process and observation variances.

5.9 Bootstrap parameter estimates

You can easily produce bootstrap parameter estimates from a fitted model using `MARSSboot()`:

```
boot.params = MARSSboot(kemfit,
     nboot=20, output="parameters", sim="parametric")$boot.params
```

```
          |2%      |20%      |40%      |60%      |80%      |100%
Progress: ||||||||||||||||||||||||||||||||||||||||||||||||||||||
```

Use `silent=TRUE` to stop the progress bar from printing. The function will also produce parameter sets generated using a Hessian matrix (`sim="hessian"`) or a non-parametric bootstrap (`sim="innovations"`).

5.10 Random initial conditions

You can use random initial conditions by passing in `MCInit=TRUE`:

```
Z.model = factor(c(1,1,2,2,2))
U.model = "equal"
Q.model =  "diagonal and unequal"
R.model = "diagonal and equal"
model.list=list(Z=Z.model, R=R.model, U=U.model, Q=Q.model)
#Set the numInits very low so the example runs quickly
cntl.list=list(MCInit=TRUE,numInits=10)
kem.mcinit = MARSS(dat, model=model.list, control=cntl.list)
```

```
> Starting Monte Carlo Initializations
          |2%      |20%      |40%      |60%      |80%      |100%
Progress: ||||||||||||||||||||||||||||||||||||||||||||||||||||||
Success! abstol and log-log tests passed at 26 iterations.
Alert: conv.test.slope.tol is 0.5.
Test with smaller values (<0.1) to ensure convergence.

MARSS fit is
Estimation method: kem
Monte Carlo initialization with random starts.
Convergence test: conv.test.slope.tol = 0.5, abstol = 0.001
Estimation converged in 26 iterations.
Log-likelihood: 12.02624
AIC: -6.052479   AICc: -3.10166

        Estimate
A.SJI    0.79876
A.PSnd  -0.78587
A.HC    -0.85458
R.diag   0.02894
U.1      0.04186
Q.(1,1)  0.01163
Q.(2,2)  0.00439
x0.1     6.05133
x0.2     6.89118

Standard errors have not been calculated.
Use MARSSparamCIs to compute CIs and bias estimates.
```

5.11 Data simulation

5.11.1 Simulated data from a fitted MARSS model

Data can be simulated from marssMLE object using `MARSSsimulate()`.

```
sim.data=MARSSsimulate(kemfit, nsim=2, tSteps=100)$sim.data
```

Then you might want to estimate parameters from that simulated data. Above we created two simulated datasets (`nsim=2`). We will fit to the first one. Here the default settings for `MARSS()` are used.

```
kem.sim.1 = MARSS(sim.data[,,1])
```

Then we might like to see the likelihood of the second set of simulated data under the model fit to the first set of data. We do that with the Kalman filter function. This function takes a marssMLE object (as output by say the MARSS function), and we have to replace the data in kem.sim.1 with the second set of simulated data.

```
kem.sim.2 = kem.sim.1
kem.sim.2$model$data = sim.data[,,2]
MARSSkf( kem.sim.2 )$logLik
```

```
[1] 56.35444
```

5.12 Bootstrap AIC

The function `MARSSaic()` computes a bootstrap AIC for model selection purposes. Use `output="AICbp"` to produce a parameter bootstrap. Use `output="AICbb"` to produce a non-parametric bootstrap AIC.

```
kemfit.with.AICb = MARSSaic(kemfit, output = "AICbp",
   Options = list(nboot = 10, silent=TRUE))
#nboot should be more like 1000, but set low here for example sake

print(kemfit.with.AICb)
```

```
MARSS fit is
Estimation method: kem
Convergence test: conv.test.slope.tol = 0.5, abstol = 0.001
Estimation converged in 22 iterations.
Log-likelihood: 7.949236
AIC: 0.1015284   AICc: 2.424109   AICbp(param): 815.4256

        Estimate
A.SJI    0.79786
A.EBays  0.27743
```

```
A.HC    -0.07035
R.diag   0.03406
U.1      0.04318
Q.diag   0.00767
x0.N     6.17205
x0.S     6.20615

Standard errors have not been calculated.
Use MARSSparamCIs to compute CIs and bias estimates.
```

We used only 10 bootstraps so the AICb estimate will be terrible, but this shows you how to compute AICb with the MARSS package.

5.13 Convergence

MARSS uses two convergence tests. The first is

$$\text{logLik}_{i+1} - \text{logLik}_i < \text{tol}$$

This is called `abstol` (meaning absolute tolerance) in the output. The second is called the `conv.test.slope`. This looks at the slope of the log parameter value (or likelihood) versus log iteration number and asks whether that is close to zero (not changing).

If you are having trouble getting the model to converge, then start by addressing the following 1) Are you trying to fit a bad model, e.g. a non-stationary model fit to stationary data or the opposite or a model that specifies independence of errors or states to data that clearly violate that or a model that implies a particular stationary distribution (particular mean and variance) to data that strongly violate that? 2) Do you have confounded parameters, e.g. two parameters that have the same effect (like effectively two intercepts)?, 3) Are you trying to fit a model to 1 data point somewhere, e.g. in a big multivariate dataset with lots of missing values? 4) How many parameters are you trying to estimate per data point? 5) Check your residuals (`residuals(kemfit)$model.residuals`) for normality. 6) Did you do any data transformations that would cause one of the variance to go to zero? Replacing 0s with a constant will do that. Replace them with NAs (missing).

6

Incorporating covariates into MARSS models

6.1 Covariates as inputs

A MARSS model with covariate effects in both the process and observation components is usually written as:

$$\begin{aligned} \mathbf{x}_t &= \mathbf{B}_t\mathbf{x}_{t-1} + \mathbf{u}_t + \mathbf{C}_t\mathbf{c}_t + \mathbf{w}_t, \text{ where } \mathbf{w}_t \sim \text{MVN}(0, \mathbf{Q}_t) \\ \mathbf{y}_t &= \mathbf{Z}_t\mathbf{x}_t + \mathbf{a}_t + \mathbf{D}_t\mathbf{d}_t + \mathbf{v}_t, \text{ where } \mathbf{v}_t \sim \text{MVN}(0, \mathbf{R}_t) \end{aligned} \tag{6.1}$$

where $\mathbf{c}_t$ is the $p \times 1$ vector of covariates (e.g., temperature, rainfall) which affect the states and $\mathbf{d}_t$ is a $q \times 1$ vector of covariates (potentially the same as $\mathbf{c}_t$), which affect the observations[1]. $\mathbf{C}_t$ is an $m \times p$ matrix of coefficients relating the effects of $\mathbf{c}_t$ to the $m \times 1$ state vector $\mathbf{x}_t$, and $\mathbf{D}_t$ is an $n \times q$ matrix of coefficients relating the effects of $\mathbf{d}_t$ to the $n \times 1$ observation vector $\mathbf{y}_t$.

With the MARSS() function, one can fit this model by simply passing in `model$c` and/or `model$d` in the MARSS() call as a $p \times T$ or $q \times T$ matrix, respectively. The form for $\mathbf{C}_t$ and $\mathbf{D}_t$ is similarly specified by passing in `model$C` and/or `model$D`. Because **C** and **D** are matrices, they must be passed in as an 3-dimensional array with the 3rd dimension equal to the number of time steps if they are time-varying. If they are time-constant, then they can be specified as 2-dimensional matrices.

6.2 Examples using plankton data

Here we show some examples[2] using the Lake Washington plankton data set and covariates in that dataset. We use 1965-1975 in order to look at only the years with high green and bluegreen algae. We use the transformed dataset

[1] One must be careful that the model is identifiable.

[2] To open a script file with the R code in this chapter, use `RShowDoc("Covariates.R",package="MARSS")`.

which has 0s replaced with NAs. First, we set up the data and z-score the data. The transformed data are z-scored, but we changed the mean when we subsampled the year so need to z-score again.

```
fulldat=lakeWAplanktonTrans
years=fulldat[,"Year"]>=1965 & fulldat[,"Year"]<1975
dat = t(fulldat[years,c("Greens", "Bluegreens")])
the.mean=apply(dat,1,mean,na.rm=TRUE)
the.sigma=sqrt(apply(dat,1,var,na.rm=TRUE))
dat=(dat-the.mean)*(1/the.sigma)
```

Next we set up the covariate data, temperature and total phosphorous. We z-score the covariates to standardize and remove the mean.

```
covariates = rbind(
   Temp=fulldat[years,"Temp"],
   TP=fulldat[years,"TP"])
#z.score the covariates
the.mean=apply(covariates,1,mean,na.rm=TRUE)
the.sigma=sqrt(apply(covariates,1,var,na.rm=TRUE))
covariates=(covariates-the.mean)*(1/the.sigma)
```

We can estimate the effect of the covariates using a process-error only model, an observation-error only model, or a model with both types of error. An observation-error only model is a multivariate regression, and we will start here so you see the relationship of MARSS model to more familiar linear regression models.

6.2.1 Multivariate linear regression

In a standard multivariate linear regression, the state process does not appear in the model and we just have an observation process with independent errors:

$$\mathbf{y}_t = \mathbf{a} + \mathbf{D}\mathbf{d}_t + \mathbf{v}_t, \text{ where } \mathbf{v}_t \sim \text{MVN}(0, \mathbf{R}) \tag{6.2}$$

The $\mathbf{a}$ are the intercepts and the $\mathbf{D}$ are the effects. We have dropped the t subscript on $\mathbf{a}$ and $\mathbf{D}$ since these will be modeled as time-constant. Writing this out for the two plankton and the two covariates we get:

$$\begin{bmatrix} y_g \\ y_{bg} \end{bmatrix}_t = \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} + \begin{bmatrix} \beta_{g,temp} & \beta_{g,tp} \\ \beta_{bg,temp} & \beta_{bg,tp} \end{bmatrix} \begin{bmatrix} temp \\ tp \end{bmatrix}_{t-1} + \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}_t \tag{6.3}$$

Let's fit this model with MARSS. The $\mathbf{x}$ part of the model is irrelevant so we want to fix the parameters in that part of the model. We won't set $\mathbf{B} = 0$ or $\mathbf{Z} = 0$ since that might cause numerical issues for the Kalman filter. Instead we fix them as identity matrices and fix $\mathbf{x}_0 = 0$ so that $\mathbf{x}_t = 0$.

```
Q=U=x0="zero"; B=Z="identity"
d=covariates
A="zero"
D="unconstrained"
y=dat #to show relationship between dat and the equation
model.list=list(B=B,U=U,Q=Q,Z=Z,A=A,D=D,d=d,x0=x0)
kem = MARSS(y, model=model.list)
```

```
Success! algorithm run for 15 iterations. abstol and log-log tests passed.
Alert: conv.test.slope.tol is 0.5.
Test with smaller values (<0.1) to ensure convergence.

MARSS fit is
Estimation method: kem
Convergence test: conv.test.slope.tol = 0.5, abstol = 0.001
Algorithm ran 15 (=minit) iterations and convergence was reached.
Log-likelihood: -276.4287
AIC: 562.8573   AICc: 563.1351

                    Estimate
R.diag                 0.706
D.(Greens,Temp)        0.367
D.(Bluegreens,Temp)    0.392
D.(Greens,TP)          0.058
D.(Bluegreens,TP)      0.535

Standard errors have not been calculated.
Use MARSSparamCIs to compute CIs and bias estimates.
```

We set `A="zero"` since the data and covariates are demeaned. Of course, one can do multiple regression in R using, say, `lm()`, and that would about 1000x faster. The EM algorithm is over-kill here, but it is shown for comparison.

We can put a twist on this and have autoregressive errors. There is still no state process in our model but instead of having i.i.d. errors in the observation process, we'll have autoregressive errors. Mathematically, this can be modelled as

$$\begin{aligned} \mathbf{x}_t &= \mathbf{B}\mathbf{x}_{t-1} + \mathbf{w}_t, \text{ where } \mathbf{w}_t \sim \text{MVN}(0, \mathbf{Q}) \\ \mathbf{y}_t &= \mathbf{D}_t\mathbf{d}_t + \mathbf{x}_t \end{aligned} \tag{6.4}$$

Our $\mathbf{x}_t$ are the errors for the observation model. We are modeling them as an autoregressive process, the $\mathbf{x}$ equation. We drop the $\mathbf{v}_t$ (set $\mathbf{R} = 0$) since the $\mathbf{x}_t$ are the errors. As usual, we have left the intercepts ($\mathbf{a}$ and $\mathbf{u}$) off since the data and covariates are all demeaned.

Here's how we fit this model in MARSS:

```
Q="unconstrained"
B="diagonal and unequal"
```

```
A=U=x0="zero"
R="diagonal and equal"
d=covariates
D="unconstrained"
y=dat #to show the relation between dat and the model equations
model.list=list(B=B,U=U,Q=Q,Z=Z,A=A,R=R,D=D,d=d,x0=x0)
kem = MARSS(y, model=model.list)
```

```
Success! abstol and log-log tests passed at 79 iterations.
Alert: conv.test.slope.tol is 0.5.
Test with smaller values (<0.1) to ensure convergence.

MARSS fit is
Estimation method: kem
Convergence test: conv.test.slope.tol = 0.5, abstol = 0.001
Estimation converged in 79 iterations.
Log-likelihood: -209.3408
AIC: 438.6816   AICc: 439.7243

                          Estimate
R.diag                      0.0428
B.(Greens,Greens)           0.2479
B.(Bluegreens,Bluegreens)   0.9136
Q.(1,1)                     0.7639
Q.(2,1)                    -0.0285
Q.(2,2)                     0.1265
D.(Greens,Temp)             0.3777
D.(Bluegreens,Temp)         0.2621
D.(Greens,TP)               0.0459
D.(Bluegreens,TP)           0.0675

Standard errors have not been calculated.
Use MARSSparamCIs to compute CIs and bias estimates.
```

You can try setting **B** to identity and MARSS will fit a model with non-mean-reverting autoregressive errors to the data. It is not done here since it turns out that that is not a very good model and it takes a long time to fit. If you try it, you'll see the **Q** gets small meaning that the **x** part is getting removed from the model.

Now let's model the data as an autoregressive process observed with no error and incorporate the covariates in the process model. Now we have left the realm of multiple regression. The **x** part represents our model of the data (in this case plankton species). How is this different from the autoregressive observation errors? Well, we are modeling our data as autoregressive so data at $t-1$ affects the data at t. Population abundances are inherently autoregressive so this model is a bit closer to the underlying mechanism generating the data.

Here is our new process model for plankton abundance.

$$\mathbf{x}_t = \mathbf{x}_{t-1} + \mathbf{C}\mathbf{c}_t + \mathbf{w}_t, \text{ where } \mathbf{w}_t \sim \text{MVN}(0, \mathbf{Q}) \tag{6.5}$$

We can fit this as follows:

```
R=A=U="zero"; B=Z="identity"
Q="equalvarcov"
C="unconstrained"
x=dat  #to show the relation between dat and the equations
model.list=list(B=B,U=U,Q=Q,Z=Z,A=A,R=R,C=C,c=covariates)
kem = MARSS(x, model=model.list)
```

```
Success! abstol and log-log tests passed at 17 iterations.
Alert: conv.test.slope.tol is 0.5.
Test with smaller values (<0.1) to ensure convergence.

MARSS fit is
Estimation method: kem
Convergence test: conv.test.slope.tol = 0.5, abstol = 0.001
Estimation converged in 17 iterations.
Log-likelihood: -285.0732
AIC: 586.1465   AICc: 586.8225

                    Estimate
Q.diag                0.7269
Q.offdiag            -0.0210
x0.Greens            -0.5189
x0.Bluegreens        -0.2431
C.(Greens,Temp)      -0.0434
C.(Bluegreens,Temp)   0.0988
C.(Greens,TP)        -0.0589
C.(Bluegreens,TP)     0.0104

Standard errors have not been calculated.
Use MARSSparamCIs to compute CIs and bias estimates.
```

Now, it looks like temperature has a strong negative effect on algae? Also our log-likelihood dropped a lot. Well, the data do not look at all like a non-mean-reverting random walk (meaning $\mathbf{B} = 1$) model which we can see by plotting the data.

```
matplot(t(dat))
```

Fig. 6.1. Plot of the Green and Bluegreen algae in Lake Washington.

The data are clearly fluctuating about some mean and so a random walk (which is non-stationary, i.e. $\mathbf{B} = 1$) is a bad model for the data. Let's switch to a better autoregressive model, a mean-reverting model. To do this, we will allow the diagonal elements of $\mathbf{B}$ to be something other than 1.

```
model.list$B="diagonal and unequal"
kem = MARSS(dat, model=model.list)
```

```
Success! algorithm run for 15 iterations. abstol and log-log tests passed.
Alert: conv.test.slope.tol is 0.5.
Test with smaller values (<0.1) to ensure convergence.

MARSS fit is
Estimation method: kem
Convergence test: conv.test.slope.tol = 0.5, abstol = 0.001
Algorithm ran 15 (=minit) iterations and convergence was reached.
Log-likelihood: -236.6106
AIC: 493.2211   AICc: 494.2638

                          Estimate
B.(Greens,Greens)           0.1981
B.(Bluegreens,Bluegreens)   0.7672
Q.diag                      0.4899
Q.offdiag                  -0.0221
x0.Greens                  -1.2915
x0.Bluegreens              -0.4179
C.(Greens,Temp)             0.2844
C.(Bluegreens,Temp)         0.1655
C.(Greens,TP)               0.0332
C.(Bluegreens,TP)           0.1340

Standard errors have not been calculated.
Use MARSSparamCIs to compute CIs and bias estimates.
```

Notice that the log-likelihood goes up quite a bit, which means that the mean-reverting model is fitting the data better.

With this model, we are estimating $\mathbf{x}_0$. If we set `model$tinitx=1`, we will get a error message that $\mathbf{R}$ diagonals are equal to 0 and we need to fix `x0`. Since $\mathbf{R} = 0$, if we set the initial states at $t = 1$, they are fully determined by the data.

```
x0=dat[,1,drop=FALSE]
model.list$tinitx=1
model.list$x0=x0
kem = MARSS(dat, model=model.list)
```

```
Success! algorithm run for 15 iterations. abstol and log-log tests passed.
Alert: conv.test.slope.tol is 0.5.
```

```
Test with smaller values (<0.1) to ensure convergence.

MARSS fit is
Estimation method: kem
Convergence test: conv.test.slope.tol = 0.5, abstol = 0.001
Algorithm ran 15 (=minit) iterations and convergence was reached.
Log-likelihood: -237.3205
AIC: 490.6411   AICc: 491.3171

                          Estimate
B.(Greens,Greens)           0.1980
B.(Bluegreens,Bluegreens)   0.7671
Q.diag                      0.4944
Q.offdiag                  -0.0223
C.(Greens,Temp)             0.2844
C.(Bluegreens,Temp)         0.1655
C.(Greens,TP)               0.0332
C.(Bluegreens,TP)           0.1340

Standard errors have not been calculated.
Use MARSSparamCIs to compute CIs and bias estimates.
```

6.3 Including observation error

The MARSS package is really designed for state-space models where you have both process and observation variability. Let's add that to the model:

$$\begin{aligned} \mathbf{x}_t &= \mathbf{B}\mathbf{x}_{t-1} + \mathbf{C}_t\mathbf{c}_t + \mathbf{w}_t, \text{ where } \mathbf{w}_t \sim \text{MVN}(0, \mathbf{Q}) \\ \mathbf{y}_t &= \mathbf{x}_{t-1} + \mathbf{v}_t, \text{ where } \mathbf{v}_t \sim \text{MVN}(0, \mathbf{R}) \end{aligned} \tag{6.6}$$

$\mathbf{y}$ is the observation of the unknown true algae abundances $\mathbf{x}$.

Let's say we knew that the observation variability on the algae measurements was about 0.16 and we wanted to include that variability in the model. Here's how we could do that:

```
model.list$R=diag(0.16,2)
kem = MARSS(dat, model=model.list)
```

```
Success! abstol and log-log tests passed at 26 iterations.
Alert: conv.test.slope.tol is 0.5.
Test with smaller values (<0.1) to ensure convergence.

MARSS fit is
Estimation method: kem
Convergence test: conv.test.slope.tol = 0.5, abstol = 0.001
```

```
Estimation converged in 26 iterations.
Log-likelihood: -240.3718
AIC: 496.7436   AICc: 497.4196

                          Estimate
B.(Greens,Greens)          0.31201
B.(Bluegreens,Bluegreens)  0.76142
Q.diag                     0.33842
Q.offdiag                 -0.00355
C.(Greens,Temp)            0.23569
C.(Bluegreens,Temp)        0.16966
C.(Greens,TP)              0.02449
C.(Bluegreens,TP)          0.14164

Standard errors have not been calculated.
Use MARSSparamCIs to compute CIs and bias estimates.
```

Note, our estimates of the effect of temperature and total phosphorous are not that different than what you get from a simple multiple regression (our first example). This might be because the autoregressive component is small, meaning the estimated diagonals on the **B** matrix are small.

6.4 Covariates with missing values or observation error

The specific formulation of Equation 6.1 creates restrictions on the assumptions regarding the covariate data. You have to assume that your covariate data has no error, which is probably not true. You cannot have missing values in your covariate data, again unlikely. You cannot combine instrument time series; for example, if you have two temperature recorders with different error rates and biases. Also, what if you have one noisy temperature recorder in the first part of your time series and then you switch to a much better recorder in the second half of your time series? All these problems require pre-analysis massaging of the covariate data, leaving out noisy and gappy covariate data, and making what can feel like arbitrary choices about which covariate time series to include.

To circumvent these potential problems and allow more flexibility in how we incorporate covariate data, one can instead treat the covariates as components of an auto-regressive process by including them in both the process and observation models. Beginning with the process equation, we can write

$$\begin{gathered}\begin{bmatrix}\mathbf{x}^{(v)}\\ \mathbf{x}^{(c)}\end{bmatrix}_t = \begin{bmatrix}\mathbf{B}^{(v)} & \mathbf{C}\\ 0 & \mathbf{B}^{(c)}\end{bmatrix}\begin{bmatrix}\mathbf{x}^{(v)}\\ \mathbf{x}^{(c)}\end{bmatrix}_{t-1} + \begin{bmatrix}\mathbf{u}^{(v)}\\ \mathbf{u}^{(c)}\end{bmatrix} + \mathbf{w}_t,\\ \mathbf{w}_t \sim \mathrm{MVN}\left(0, \begin{bmatrix}\mathbf{Q}^{(v)} & 0\\ 0 & \mathbf{Q}^{(c)}\end{bmatrix}\right)\end{gathered} \tag{6.7}$$

The elements with superscript (v) are for the k variate states and those with superscript (c) are for the q covariate states. The dimension of $\mathbf{x}^{(c)}$ is $q \times 1$ and q is not necessarily equal to p, the number of covariate observation time series in your dataset. Imagine, for example, that you have two temperature sensors and you are combining these data. Then you have two covariate observation time series ($p = 2$) but only one underlying covariate state time series ($q = 1$). The matrix $\mathbf{C}$ is dimension $k \times q$, and $\mathbf{B}^{(c)}$ and $\mathbf{Q}^{(c)}$ are dimension $q \times q$. The dimension[3] of $\mathbf{x}^{(v)}$ is $k \times 1$, and $\mathbf{B}^{(v)}$ and $\mathbf{Q}^{(v)}$ are dimension $k \times k$.

Next, we can write the observation equation in an analogous manner, such that

$$\begin{gathered}\begin{bmatrix}\mathbf{y}^{(v)}\\ \mathbf{y}^{(c)}\end{bmatrix}_t = \begin{bmatrix}\mathbf{Z}^{(v)} & \mathbf{D}\\ 0 & \mathbf{Z}^{(c)}\end{bmatrix}\begin{bmatrix}\mathbf{x}^{(v)}\\ \mathbf{x}^{(c)}\end{bmatrix}_t + \begin{bmatrix}\mathbf{a}^{(v)}\\ \mathbf{a}^{(c)}\end{bmatrix} + \mathbf{v}_t,\\ \mathbf{v}_t \sim \mathrm{MVN}\left(0, \begin{bmatrix}\mathbf{R}^{(v)} & 0\\ 0 & \mathbf{R}^{(c)}\end{bmatrix}\right)\end{gathered} \tag{6.8}$$

The dimension of $\mathbf{y}^{(c)}$ is $p \times 1$, where p is the number of covariate observation time series in your dataset. The dimension of $\mathbf{y}^{(v)}$ is $l \times 1$, where l is the number of variate observation time series in your dataset. The total dimension of $\mathbf{y}$ is $l + p$. The matrix $\mathbf{D}$ is dimension $l \times q$, $\mathbf{Z}^{(c)}$ is dimension $p \times q$, and $\mathbf{R}^{(c)}$ are dimension $p \times p$. The dimension of $\mathbf{Z}^{(v)}$ is dimension $l \times k$, and $\mathbf{R}^{(v)}$ are dimension $l \times l$.

The $\mathbf{D}$ matrix would presumably have a number of all zero rows in it, as would the $\mathbf{C}$ matrix. The covariates that affect the states would often be different than the covariates that affect the observations. For example, mean annual temperature would affect population growth rates for many species while having little or no affect on observability, and turbidity might strongly affect observability in many types of aquatic, say, surveys but have little affect on population growth rate.

Our MARSS model with covariates now looks on the surface like a regular MARSS model:

$$\begin{gathered}\mathbf{x}_t = \mathbf{B}\mathbf{x}_{t-1} + \mathbf{u} + \mathbf{w}_t, \text{ where } \mathbf{w}_t \sim \mathrm{MVN}(0, \mathbf{Q})\\ \mathbf{y}_t = \mathbf{Z}\mathbf{x}_t + \mathbf{a} + \mathbf{v}_t, \text{ where } \mathbf{v}_t \sim \mathrm{MVN}(0, \mathbf{R})\end{gathered} \tag{6.9}$$

with the $\mathbf{x}_t$, $\mathbf{y}_t$ and parameter matrices redefined as in Equations 6.7 and 6.8:

$$\begin{aligned}\mathbf{x} &= \begin{bmatrix}\mathbf{x}^{(v)}\\ \mathbf{x}^{(c)}\end{bmatrix} & \mathbf{B} &= \begin{bmatrix}\mathbf{B}^{(v)} & \mathbf{C}\\ 0 & \mathbf{B}^{(c)}\end{bmatrix} & \mathbf{u} &= \begin{bmatrix}\mathbf{u}^{(v)}\\ \mathbf{u}^{(c)}\end{bmatrix} & \mathbf{Q} &= \begin{bmatrix}\mathbf{Q}^{(v)} & 0\\ 0 & \mathbf{Q}^{(c)}\end{bmatrix}\\ \mathbf{y} &= \begin{bmatrix}\mathbf{y}^{(v)}\\ \mathbf{y}^{(c)}\end{bmatrix} & \mathbf{Z} &= \begin{bmatrix}\mathbf{Z}^{(v)} & \mathbf{D}\\ 0 & \mathbf{Z}^{(c)}\end{bmatrix} & \mathbf{a} &= \begin{bmatrix}\mathbf{a}^{(v)}\\ \mathbf{a}^{(c)}\end{bmatrix} & \mathbf{R} &= \begin{bmatrix}\mathbf{R}^{(v)} & 0\\ 0 & \mathbf{R}^{(c)}\end{bmatrix}\end{aligned} \tag{6.10}$$

[3] The dimension of $\mathbf{x}$ is always denoted m. If your process model includes only variates, then $k = m$, but now your process model includes k variates and q covariate states so $m = k + q$.

Note **Q** and **R** are written as block diagonal matrices, but you could allow covariances if that made sense. **u** and **a** are column vectors here. We can fit the model (Equation 6.9) as usual using the `MARSS()` function.

The log-likelihood that is returned by MARSS will include the log-likelihood of the covariates under the covariate state model. If you want only the the log-likelihood of the non-covariate data, you will need to substract off the log-likelihood of the covariate model:

$$\begin{aligned} \mathbf{x}_t^{(c)} &= \mathbf{B}^{(c)}\mathbf{x}_{t-1}^{(c)} + \mathbf{u}^{(c)} + \mathbf{w}_t, \text{ where } \mathbf{w}_t \sim \text{MVN}(0, \mathbf{Q}^{(c)}) \\ \mathbf{y}_t^{(c)} &= \mathbf{Z}^{(c)}\mathbf{x}_t^{(c)} + \mathbf{a}^{(c)} + \mathbf{v}_t, \text{ where } \mathbf{v}_t \sim \text{MVN}(0, \mathbf{R}^{(c)}) \end{aligned} \tag{6.11}$$

An easy way to get this log-likelihood for the covariate data only is use the augmented model (Equation 6.9 with terms defined as in Equation 6.10) but pass in missing values for the non-covariate data. The following code shows how to do this.

```
y.aug=rbind(data,covariates)
fit.aug=MARSS(y.aug, model=model.aug)
```

`fit.aug` is the MLE object that can be passed to MARSSkf(). You need to make a version of this MLE object with the non-covariate data filled with NAs so that you can compute the logLik without the covariates. This needs to be done in the `marss` element since that is what is used by MARSSkf(). Below is code to do this.

```
fit.cov=fit.aug; fit.cov$marss$data[1:dim(data)[1],]=NA
extra.LL=MARSSkf(fit.cov)$logLik
```

Note that when you fit the augmented model, the estimates of **C** and $\mathbf{B}^{(c)}$ are affected by the non-covariate data since the model for both the non-covariate and covariate data are estimated simultaneously and are not independent (since the covariate states affect the non-covariates states). If you want the covariate model to be unaffected by the non-covariate data, you can fit the covariate model separately and use the estimates for $\mathbf{B}^{(c)}$ and $\mathbf{Q}^{(c)}$ as fixed values in your augmented model.

6.5 Modeling seasonal effects

We return to our basic model to show a few approaches for including seasonal effects in a MARSS model. We will use the five phytoplankton species for this example. We will use the transformed dataset which has 0s replaced with NAs and is z-scored.

```
years=fulldat[,"Year"]>=1965 & fulldat[,"Year"]<1975
phytos = c("Diatoms","Greens","Bluegreens","Unicells","Other.algae")
dat = t(fulldat[years,phytos])
```

```
#z.score the data because we changed the mean when we subsampled
the.mean=apply(dat,1,mean,na.rm=TRUE)
the.sigma=sqrt(apply(dat,1,var,na.rm=TRUE))
dat=(dat-the.mean)*(1/the.sigma)
TT=dim(dat)[2] #number of columns
```

6.5.1 Treat month as a factor

We can treat month as a factor and estimate a month effect for each month. To take this approach, we create a covariate with a row for each month (Jan, Feb, ...) and a column for each time point. We put a 1 in the January row for each January time point, repeat for the other months. The following code will create such a **c** matrix for the Lake Washington data which were collected monthly.

```
period=12
c.in = diag(period)
for(i in 2:(ceiling(TT / period))) { c.in <- cbind(c.in, diag(period)) }
# better row names
the.months <- c("Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sep","Oct","Nov","Dec")
rownames(c.in) <- the.months
```

Next we need to set up the form of the **C** matrix which defines any constraints we want to set on the month effects. If we want each month to have a separate effect for each taxon ($5 \times 12 = 60$ effects), we set

```
C = "unconstrained"
```

Then we set up the form for the rest of the model parameters. We make the following assumptions:

```
#taxon dynamics are independent with unique density-dependence
#and independent process errors (a bad assumption probably)
B = "diagonal and unequal"
Q = "diagonal and unequal"
#we have demeaned the data and are fitting a mean-reverting model
# by estimating the B diagonal, thus
U = "zero"
#Each time series is associated with only one taxon
Z = "identity"
#We have demeaned the data and it fluatuates around a mean
A = "zero"
#We assume that any observation error is independent
#but has similar variance due to the similar collection methods
R = "diagonal and equal"
#We are not including covariate effects in the observation equation
D = "zero"
d = "zero"
```

Now we can set up the model list for MARSS and fit. Results not shown since they are verbose (60 effects).

```
model.list = list(B=B,U=U,Q=Q,Z=Z,A=A,R=R,C=C,c=c.in,D=D,d=d)
seasonal.model.1 = MARSS(dat, model=model.list, control=list(maxit=1500))
# Get the estimated seasonal effects
# rows are taxa, cols are seasonal effects
seas.1 <- coef(seasonal.model.1,type="matrix")$C
rownames(seas.1) <- phytos
colnames(seas.1) <- the.months
```

Figure 6.2, row 1, shows the estimated seasonal effects for this model. Note that if we had set U="unequal", we would need to set one of the columns of **C** to zero because the model would be under-determined (infinite number of solutions). If we substracted the mean January abundance off each time series, we could set the January column in **C** to 0 and get rid of 5 estimated effects.

6.5.2 Model the month effect as a polynomial

The factor approach required estimating 60 effects. Another approach is to model the month effect as a 3rd order (or higher) polynomial: $a+b\times m+c\times m^2+d\times m^3$ where m is month number. This approach has less flexibility but requires only 20 estimated parameters (4 **C** elements times 5 taxa). For this approach, we make a month covariate matrix, the **c**, with 1, m, m^2, and m^3 in different rows. m is the month number, i.e. January=1, February=2, etc. Below shows how we can set up this matrix.

```
poly.order = 3
month.cov = matrix(1,1,period)
for(i in 1:poly.order) { month.cov = rbind(month.cov,(1:12)^i)  }
# our c matrix is month.cov replicated once for each year
c.m.poly <- matrix(month.cov, poly.order+1, TT, byrow=FALSE)
# Everything else remains the same as in the previous example
model.list <- list(B=B,U=U,Q=Q,Z=Z,A=A,R=R,C=C,c=c.m.poly,D=D,d=d)
seasonal.model.2 <- MARSS(dat, model=model.list, control=list(maxit=1500))
```

The effect of month m for taxon i is $a_i+b_i\times m+c_i\times m^2+d_i\times m^3$, where a, b, c and d are in the i-th row of **C**. We can then calculate the seasonal matrix as follows, where each row is a taxon and each column is a month:

```
C.2 <- coef(seasonal.model.2,type="matrix")$C
seas.2 <- C.2 %*% month.cov
rownames(seas.2) <- phytos
colnames(seas.2) <- the.months
```

6.5.3 Model the seasonal effect using a Fourier transform

The factor approach required estimating 60 effects, and the 3rd order polynomial model was an improvement at only 20 parameters. A third option is to use a combination of sine and cosine waves, which would require only 10 parameters. We begin by defining the seasonal covariate matrix, **c**, as

```
cos.t <- cos(2 * pi * seq(TT) / period)
sin.t <- sin(2 * pi * seq(TT) / period)
c.Four <- rbind(cos.t,sin.t)
```

The net seasonal effect for taxon i is then $a_i \times cos.t + b_i \times sin.t$.

Everything else remains the same and we can fit this model as follows:

```
model.list <- list(B=B,U=U,Q=Q,Z=Z,A=A,R=R,C=C,c=c.Four,D=D,d=d)
seasonal.model.3 <- MARSS(dat, model=model.list, control=list(maxit=1500))
```

We make our seasonal effect matrix as follows:

```
C.c <- coef(seasonal.model.3, type="matrix")$C
seas.3 <- C.c %*% c.Four[,1:period]
rownames(seas.3) <- phytos
colnames(seas.3) <- the.months
```

6.5.4 Diagnostics

We will do some basic diagnostics for these three approaches and look at a plot of the model residuals and ACF's for all five taxa using the following code:

```
for(i in 1:5) {
  plot.ts(residuals(seasonal.model)$model.residuals[i,],
      ylab="Residual", main=phytos[i])
  abline(h=0, lty="dashed")
  acf(residuals(seasonal.model.1)$model.residuals[i,])
  }
```

Figure 6.3 shows these diagnostics for the first model. The model residuals for all taxa appear to show significant negative autocorrelation at lag=1, suggesting that the model is inadequate to capture all of the systematic variation in phytoplankton abundance. The model with the month effect as a polynomial is better behaved (Figure 6.4) and the model with the month effect as a Fourier series is best of all three (Figure 6.5).

The model selection results for the different seasonal models also suggests that the model with a seasonal effect via Fourier sequence would be the most parsimonious.

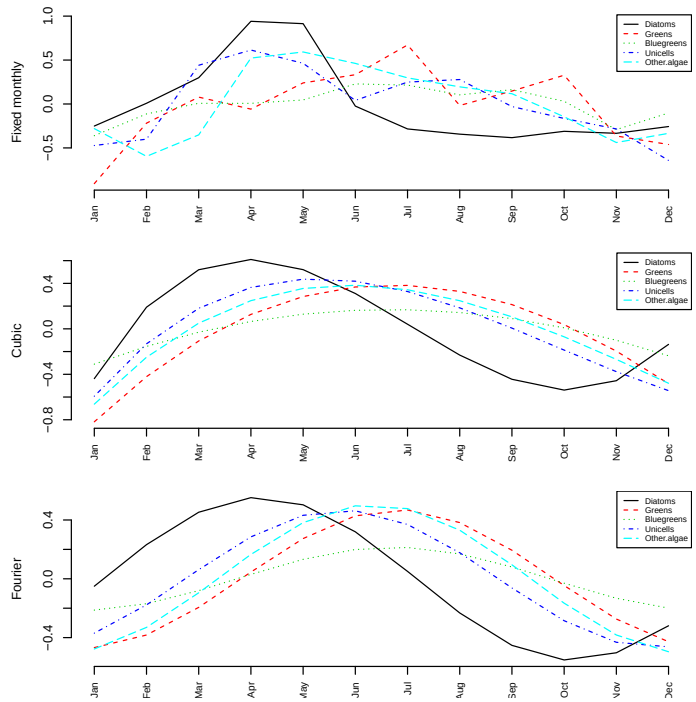

Fig. 6.2. Estimated month effects for the three approaches: row 1) each month modelled as a separate effect for each taxon (60 parameters), row 2) month effect modelled as a 3rd order polynomial (20 parameters), row 3) month effect modelled as a combination of sine and cosine (10 parameters).

```
data.frame(Model=c("Fixed", "Cubic", "Fourier"),
                    AICc=round(c(seasonal.model.1$AICc,
                     seasonal.model.2$AICc,
                     seasonal.model.3$AICc),1))
```

```
    Model   AICc
1   Fixed 1188.4
2   Cubic 1144.9
3 Fourier 1127.4
```

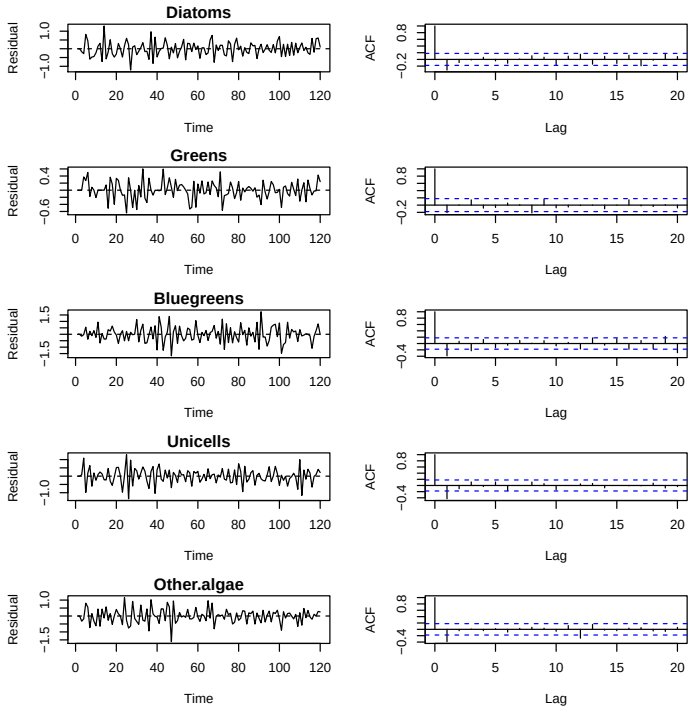

Fig. 6.3. The model residuals and ACF for the model with each month having an estimated fixed effect.

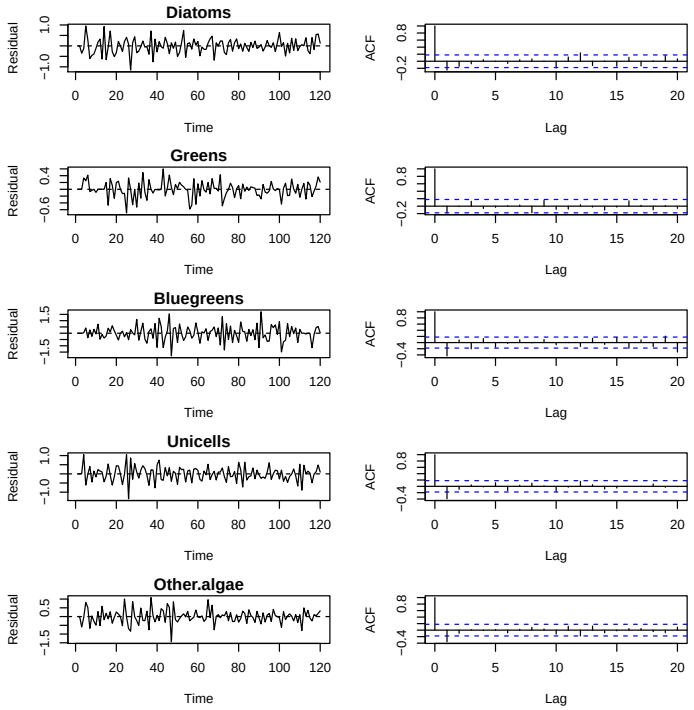

Fig. 6.4. The model residuals and ACF for the model with month effect modelled as a 3rd order polynomial.

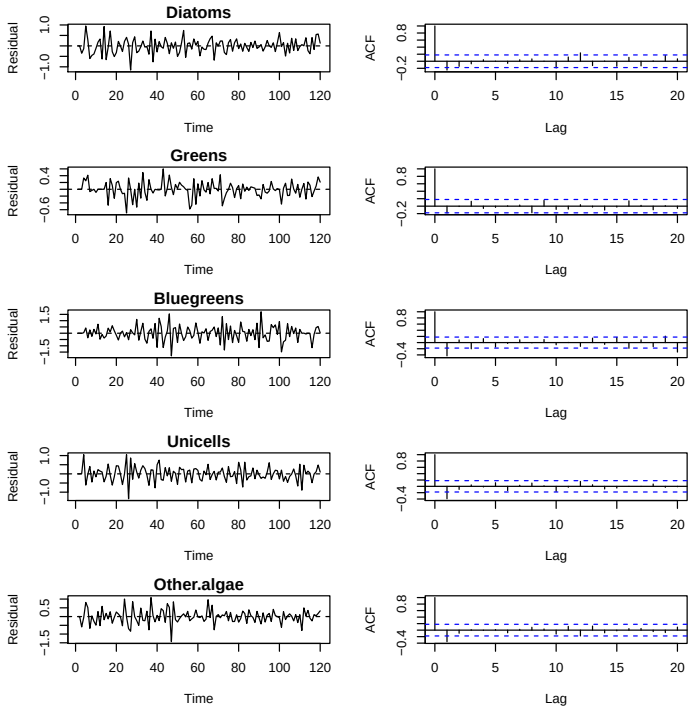

Fig. 6.5. The model residuals and ACF for the model with month effect modelled as a sum of sines and cosines.

7

Lag-p models with MARSS

7.1 Background

Many types of time-series models, for example AR(p), ARMA(p,q), and ARMAX(p,q), can be written in MARSS(1) form (see section 11.3.2 in Tsay (2010)). Writing these models in state-space form allows one to take advantage of the fitting algorithms for MARSS(1) models. There are many R packages for fitting AR(p) and ARMA(p,q) models. If you are only interested in univariate models then I suggest looking into the `arima()` function included in base R and into R packages that specialize in fitting ARMA models to univariate data. The forecast package in R is a good place to start but others can be found on the CRAN task view: Time Series Analysis. This chapter is focused on fitting ARMA models by converting to a MARSS. When written in state-space form, it does not necessarily mean that the process is observed with error, e.g. the ARMA(p,q) is not overlaid with an observation model. Rather the original model, with no observation process, is rewritten in the state-space form to facilitate statistical analysis.

7.2 MAR(2) models

The MARSS model (Equation 1.1) is written such that $\mathbf{x}_t$ is only affected by $\mathbf{x}_{t-1}$. This is called a lag-1 model. A model in which lags up to time p are included allows that $\mathbf{x}_t$ may be affected by $\mathbf{x}_{t-p}$. These are called lag-p models. A MARSS model with p lags in the process model can be equivalently rewritten as a MARSS(1) model by redefining the $\mathbf{x}$ state vector.

Here is a MAR(2) model, a lag-2 MAR model:

$$\mathbf{x}_t' = \mathbf{B}_1\mathbf{x}_{t-1}' + \mathbf{B}_2\mathbf{x}_{t-2}' + \mathbf{u} + \mathbf{w}_t, \text{ where } \mathbf{w}_t \sim \text{MVN}(0, \mathbf{Q}) \tag{7.1}$$

We rewrite this as lag-1 by defining $\mathbf{x}_t = \begin{bmatrix} \mathbf{x}_t' \\ \mathbf{x}_{t-1}' \end{bmatrix}$:

$$\begin{bmatrix} \mathbf{x}_t' \\ \mathbf{x}_{t-1}' \end{bmatrix} = \begin{bmatrix} \mathbf{B}_1 & \mathbf{B}_2 \\ \mathbf{I}_m & 0 \end{bmatrix} \begin{bmatrix} \mathbf{x}_{t-1}' \\ \mathbf{x}_{t-2}' \end{bmatrix}_{t-1} + \begin{bmatrix} \mathbf{u} \\ 0 \end{bmatrix} + \begin{bmatrix} w_t \\ 0 \end{bmatrix}, \quad \begin{bmatrix} w_t \\ 0 \end{bmatrix} \sim \text{MVN}\left(0, \begin{bmatrix} \mathbf{Q} & 0 \\ 0 & 0 \end{bmatrix}\right)$$
$$\begin{bmatrix} \mathbf{x}_0' \\ \mathbf{x}_{-1}' \end{bmatrix} \sim \text{MVN}(\pi, \Lambda) \tag{7.2}$$

7.2.1 Example of AR(2)

Here is an example of fitting a univariate AR(2) model to AR(2) data. First, let's generate some simulated AR(2) data:

```
TT=50
true.2=c(r=0,b1=-1.5,b2=-0.75,q=1)
temp=arima.sim(n=TT,list(ar=true.2[2:3]),sd=sqrt(true.2[4]))
sim.ar2=matrix(temp,nrow=1)
```

Next, we write the AR(2) model (Equation 7.2) as a MARSS model in R :

```
Z=matrix(c(1,0),1,2)
B=matrix(list("b1",1,"b2",0),2,2)
U=matrix(0,2,1)
Q=matrix(list("q",0,0,0),2,2)
A=matrix(0,1,1)
R=matrix(0,1,1)
pi=matrix(sim.ar2[2:1],2,1)
V=matrix(0,2,2)
model.list.2=list(Z=Z,B=B,U=U,Q=Q,A=A,R=R,x0=pi,V0=V,tinitx=1)
```

Notice that we do not need to estimate π for x_1 since we are setting $\pi = \begin{bmatrix} x_2 \\ x_1 \end{bmatrix}$, using only data starting at x_2, and setting $\mathbf{R} = 0$. Thus $\mathbf{x}_1$, and by extension the first element of π, is known the data.

Then we can fit the AR(2) model to these simulated data. We pass in good initial conditions so it does not take quite so long to run.

```
init.list=list(Q=matrix(1,1,1),B=matrix(1,2,1))
ar2=MARSS(sim.ar2[2:TT],model=model.list.2,inits=init.list)
```

```
Success! algorithm run for 15 iterations. abstol and log-log tests passed.
Alert: conv.test.slope.tol is 0.5.
Test with smaller values (<0.1) to ensure convergence.

MARSS fit is
Estimation method: kem
Convergence test: conv.test.slope.tol = 0.5, abstol = 0.001
Algorithm ran 15 (=minit) iterations and convergence was reached.
Log-likelihood: -65.9146
AIC: 137.8292   AICc: 138.3625
```

```
     Estimate
B.b1   -1.405
B.b2   -0.710
Q.q     0.878

Standard errors have not been calculated.
Use MARSSparamCIs to compute CIs and bias estimates.
```

```
print(cbind(true=true.2[2:4],estimated=coef(ar2,type="vector")))
```

```
    true  estimated
b1 -1.50 -1.4053160
b2 -0.75 -0.7098832
q   1.00  0.8783375
```

Missing values in the data are fine. Let's make half the data missing:

```
TT=50
gappy.data=sim.ar2[2:TT]
gappy.data[floor(runif(TT/2,1,TT))]=NA
ar2.gappy=MARSS(gappy.data,model=model.list.2,inits=init.list)
```

```
Success! abstol and log-log tests passed at 21 iterations.
Alert: conv.test.slope.tol is 0.5.
Test with smaller values (<0.1) to ensure convergence.

MARSS fit is
Estimation method: kem
Convergence test: conv.test.slope.tol = 0.5, abstol = 0.001
Estimation converged in 21 iterations.
Log-likelihood: -41.7885
AIC: 89.57701   AICc: 90.62049

     Estimate
B.b1   -1.473
B.b2   -0.781
Q.q     0.642

Standard errors have not been calculated.
Use MARSSparamCIs to compute CIs and bias estimates.
```

By the way, there are much easier and faster functions in R for fitting univariate AR models (no observation error). For example, here is how you would fit the AR(2) model using the base `arima()` function:

```
arima(gappy.data,order=c(2,0,0),include.mean=FALSE)
```

```
Call:
arima(x = gappy.data, order = c(2, 0, 0), include.mean = FALSE)

Coefficients:
          ar1      ar2
      -1.4648  -0.7782
s.e.   0.1048   0.1049

sigma^2 estimated as 0.7443:  log likelihood = -45.13,  aic = 96.26
```

The advantage of MARSS really only comes in when you are fitting multivariate models.

7.2.2 Example of MAR(2)

Here we show an example of fitting a multivariate AR(2) model, MAR(2). Let's make some simulated data of two realizations of the same AR(2) process:

```
TT=50
true.2=c(r=0,b1=-1.5,b2=-0.75,q=1)
temp1=arima.sim(n=TT,list(ar=true.2[2:3]),sd=sqrt(true.2[4]))
temp2=arima.sim(n=TT,list(ar=true.2[2:3]),sd=sqrt(true.2[4]))
sim.mar2=rbind(temp1,temp2)
```

We want to fit with a MAR(2) model to allow us to use both datasets together to estimate the AR(2) parameters. We need to set up the multivariate model (Equation 7.2):

```
Z=matrix(c(1,0,0,1,0,0,0,0),2,4)
B1=matrix(list(0),2,2); diag(B1)="b1"
B2=matrix(list(0),2,2); diag(B2)="b2"
B=matrix(list(0),4,4)
B[1:2,1:2]=B1; B[1:2,3:4]=B2; B[3:4,1:2]=diag(1,2)
U=matrix(0,4,1)
Q=matrix(list(0),4,4)
Q[1,1]="q"; Q[2,2]="q"
A=matrix(0,2,1)
R=matrix(0,2,2)
pi=matrix(c(sim.mar2[,2],sim.mar2[,1]),4,1)
V=matrix(0,4,4)
model.list.2m=list(Z=Z,B=B,U=U,Q=Q,A=A,R=R,x0=pi,V0=V,tinitx=1)
```

We then fit the model as usual:

```
init.list=list(Q=matrix(1,1,1),B=matrix(1,2,1))
mar2=MARSS(sim.mar2[,2:TT],model=model.list.2m,inits=init.list)
```

Then we can compare how using two time series improves the fit versus using only one alone:

```
model.list.2$x0=matrix(sim.mar2[1,2:1],2,1)
mar2a=MARSS(sim.mar2[1,2:TT],model=model.list.2,inits=init.list)
model.list.2$x0=matrix(sim.mar2[2,2:1],2,1)
mar2b=MARSS(sim.mar2[2,2:TT],model=model.list.2,inits=init.list)
```

```
   true   est.mar2  est.mar2a  est.mar2b
b1 -1.50 -1.5395800 -1.5559545 -1.4991659
b2 -0.75 -0.8435272 -0.8766708 -0.7672839
q   1.00  1.3724611  1.6557291  1.0588134
```

7.3 MAR(p) models

A MAR(p) model, a MAR model with lag-p, would be:

$$\mathbf{x}_t' = \mathbf{B}_1\mathbf{x}_{t-1}' + \mathbf{B}_2\mathbf{x}_{t-2}' + \cdots + \mathbf{B}_p\mathbf{x}_{t-p}' + \mathbf{u}' + \mathbf{w}_t', \text{ where } \mathbf{w}_t' \sim \text{MVN}(0, \mathbf{Q}')$$

where

$$\mathbf{x}_t = \begin{bmatrix} \mathbf{x}_t' \\ \mathbf{x}_{t-1}' \\ \vdots \\ \mathbf{x}_{t-p}' \end{bmatrix}, \mathbf{B} = \begin{bmatrix} \mathbf{B}_1 & \mathbf{B}_2 & \dots & \mathbf{B}_p \\ \mathbf{I}_m & 0 & \dots & 0 \\ 0 & \mathbf{I}_m & \dots & 0 \\ 0 & 0 & \ddots & \vdots \\ 0 & 0 & \dots & 0 \end{bmatrix}, \mathbf{u} = \begin{bmatrix} \mathbf{u}' \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \mathbf{Q} = \begin{bmatrix} \mathbf{Q}' & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ 0 & 0 & \ddots & \vdots \\ 0 & 0 & \dots & 0 \end{bmatrix} \tag{7.3}$$

Here's an example of fitting a univariate AR(3) with MARSS. We need more data to estimate an AR(3), so we up the length of data.

```
TT=100
temp=arima.sim(n=TT,list(ar=c(-1.5,-.75, .05)),sd=1)
sim.ar3=matrix(temp,nrow=1)
```

We write the AR lag-3 model as a MARSS model in R :

```
Z=matrix(c(1,0,0),1,3)
B=matrix(list("b.1",1,0,"b.2",0,1,"b.3",0,0),3,3)
U=matrix(0,3,1)
Q=matrix(list(0),3,3); Q[1,1]="q.1"
A=matrix(0,1,1)
R=matrix(0,1,1)
pi=matrix(sim.ar3[3:1],3,1)
V=matrix(0,3,3)
model.list=list(Z=Z,B=B,U=U,Q=Q,A=A,R=R,x0=pi,V0=V,tinitx=1)

init.list=list(Q=matrix(1,1,1),B=matrix(1,3,1))
ar3=MARSS(sim.ar3[3:TT],model=model.list,inits=init.list)
```

```
Success! algorithm run for 15 iterations. abstol and log-log tests passed.
Alert: conv.test.slope.tol is 0.5.
Test with smaller values (<0.1) to ensure convergence.

MARSS fit is
Estimation method: kem
Convergence test: conv.test.slope.tol = 0.5, abstol = 0.001
Algorithm ran 15 (=minit) iterations and convergence was reached.
Log-likelihood: -142.1761
AIC: 292.3523   AICc: 292.7824

      Estimate
B.b.1  -1.6083
B.b.2  -0.8900
B.b.3  -0.0235
Q.q.1   1.0775

Standard errors have not been calculated.
Use MARSSparamCIs to compute CIs and bias estimates.
```

7.4 MARSS(p): models with observation error

We can estimate MARSS(p) models, but the difficulty is specifying the initial state condition. Our $\mathbf{x}_{t_0}$ involves $\mathbf{x}_t$, $\mathbf{x}_{t-1}$, ... However, we do not know the variance covariance structure for them. Specifying $V_{t_0} = 0$ and estimating $\mathbf{x}_{t_0}$ does not seem to be very robust—meaning the EM algorithm runs into numerical problems. But if we have lots of data and fix $\mathbf{x}_{t_0}$, we can still recover $\mathbf{B}$ and $\mathbf{Q}$, at least for the univariate case.

Here is an example where we set `x0` to the mean of the data. This is an approximation, but if the data are long enough, the estimates are insensitive to `x0`. Why not set Λ equal to a diagonal matrix with large values on the diagonal to approximate a vague prior? The temporally consecutive initial states are definitely not independent. A diagonal matrix would imply independence which will conflict with the process model and lead to an inconsistent model.

```
TT=1000 #set long
true.2ss=c(r=.5,b1=-1.5,b2=-0.75,q=.1)
temp=arima.sim(n=TT,list(ar=true.2ss[2:3]),sd=sqrt(true.2ss[4]))
sim.ar=matrix(temp,nrow=1)
noise=rnorm(TT-1,0,sqrt(true.2ss[1]))
noisy.data=sim.ar[2:TT]+noise
#set up a lag-2 model as MARSS lag-1
Z=matrix(c(1,0),1,2)
B=matrix(list("b1",1,"b2",0),2,2)
```

```
U=matrix(0,2,1)
Q=matrix(list("q",0,0,0),2,2)
A=matrix(0,1,1)
R=matrix("r")
V=matrix(0,2,2)
pi=matrix(mean(noisy.data),2,1)
model.list.2ss=list(Z=Z,B=B,U=U,Q=Q,A=A,R=R,x0=pi,V0=V,tinitx=0)
#give it some reasonable inits to run faster
init.list=list(Q=matrix(.01,1,1),B=matrix(1,2,1))
ar2ss=MARSS(noisy.data,model=model.list.2ss,inits=init.list)
```

```
Success! abstol and log-log tests passed at 68 iterations.
Alert: conv.test.slope.tol is 0.5.
Test with smaller values (<0.1) to ensure convergence.

MARSS fit is
Estimation method: kem
Convergence test: conv.test.slope.tol = 0.5, abstol = 0.001
Estimation converged in 68 iterations.
Log-likelihood: -1351.566
AIC: 2711.132   AICc: 2711.172

     Estimate
R.r     0.506
B.b1   -1.408
B.b2   -0.667
Q.q     0.111

Standard errors have not been calculated.
Use MARSSparamCIs to compute CIs and bias estimates.
```

We can compare the results to modeling the data as if there is no observation error, and we see that that assumption leads to poor **B** estimates:

```
model.list.2ss$R=matrix(0)
ar2ss2=MARSS(noisy.data,model=model.list.2ss,inits=init.list)
```

```
Success! algorithm run for 15 iterations. abstol and log-log tests passed.
Alert: conv.test.slope.tol is 0.5.
Test with smaller values (<0.1) to ensure convergence.

MARSS fit is
Estimation method: kem
Convergence test: conv.test.slope.tol = 0.5, abstol = 0.001
Algorithm ran 15 (=minit) iterations and convergence was reached.
Log-likelihood: -1366.807
AIC: 2739.614   AICc: 2739.638
```

```
     Estimate
B.b1  -0.4694
B.b2   0.0553
Q.q    0.9035

Standard errors have not been calculated.
Use MARSSparamCIs to compute CIs and bias estimates.
```

```
 print(cbind(true=true.2ss,
   est.no.noise=c(NA,coef(ar2ss2,type="vector")),
   est.noisy=coef(ar2ss,type="vector")))
```

```
    true est.no.noise  est.noisy
r   0.50           NA  0.5059845
b1 -1.50   -0.4693653 -1.4083813
b2 -0.75    0.0553177 -0.6668164
q   0.10    0.9034567  0.1107144
```

The middle column are the estimates assuming that the data have no observation error and the right column are our estimates with the observation error estimated. Clearly, assuming no observation error when it is present has negative consequences for the **B** and **Q** estimates.

By the way, there is a straight-forward way to deal with the measurement error if you are working with univariate ARMA models and you are only interested in the AR parameters (the b's). Inclusion of measurement error leads to additional MA components up to lag p (Staudenmayer and Buonaccorsi, 2005). This means that if you are fitting a AR(p) model with measurement error, you can fit a ARMA(p,p) and the measurement error will be absorbed in the p MA components. For the example above, we could estimate the AR parameters for our AR(2) data with measurement error by fitting a ARMA(p,p) model. Here's how we could do that using R 's `arima()` function:

```
 arima(noisy.data,order=c(2,0,2),include.mean=FALSE)
```

```
Call:
arima(x = noisy.data, order = c(2, 0, 2), include.mean = FALSE)

Coefficients:
          ar1      ar2     ma1     ma2
      -1.5828  -0.6661  1.1308  0.2103
s.e.   0.0439   0.0423  0.0546  0.0576

sigma^2 estimated as 0.8722:  log likelihood = -1349.5,  aic = 2709
```

As you can see accounting for the measurement error definitely improves the estimates for the AR component. Although both MARSS(1) and

ARMA(p,p) approaches can be used to deal with AR(p) processes observed with error, our simulations suggest that the MARSS(1) approach is less biased and more precise (Figure 7.1) and that the EM algorithm is working better for this problem. The performance of different approaches depends greatly on the underlying model and if we used, for example, $b_1 = 0.8$ and $b_2 = -0.2$, the ARMA(2,2) gives b_1 estimates close to 0 while the MARSS(1) EM approach relatively unbiased (but variable) estimates. One would want to also check REML approaches for fitting the ARMA(p,p) models since REML has been found to be less biased than ML estimation for this class (Cheang and Reinsel, 2000; Ives et al., 2010) (Ives et al. 2010 has R code for REML estimation of ARMA(p,q) models in their appendix).

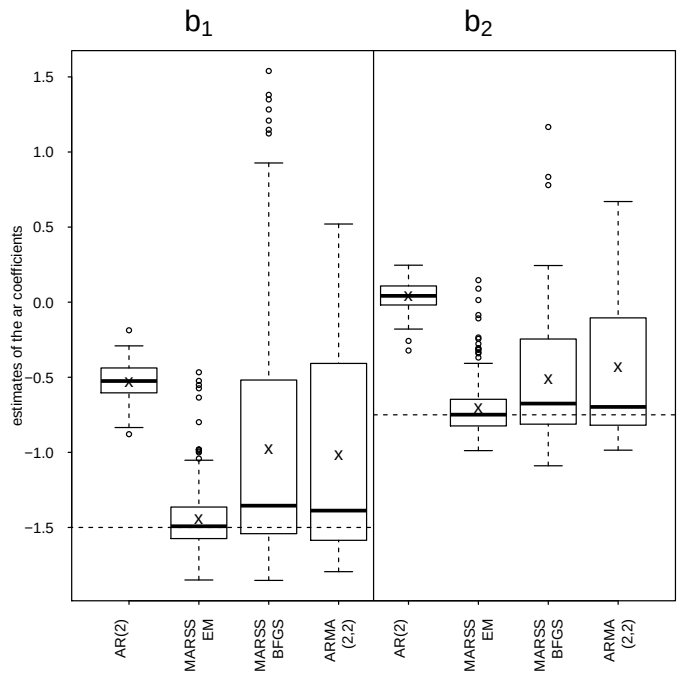

Fig. 7.1. Comparison of the AR parameter estimates using different approaches to model ARSS(2) data (univariate AR(2) data observed with error). Results are from 200 simulations of AR(2) data with 100 time steps. Results are shown for the b_1 and b_2 parameters of the AR process fit with a 1) AR(2) model with no correction for measurement error, 2) MARSS(1) model fit via the EM optimization, 3) MARSS(1) model fit via BFGS optimization, 4) ARMA(2,2) model fit with the R `arima` function, and 5) AR(2) model fit 2nd differencing with the R `arima` function. The "x" shows the mean of the simulations and the bar in the boxplot is the median. The true values are shown with the dashed horizontal line. The σ^2 for the AR process was 0.1 and the σ^2 for the measurement error was 0.5. The b_1 parameters was -1.5,and b_2 was -0.75.

8

Case study instructions

The case studies walk you through some analyses of multivariate population count data using MARSS models and the `MARSS()` function. This will take you through both the conceptual steps (with pencil and paper) and a R step which translates the conceptual model into code.

Set-up

- If you haven't already, install the MARSS package. See directions on the CRAN webpage (http://cran.r-project.org/) for instructions on installing packages. You will need write permissions for your R program directories to install packages. See the help pages on CRAN for workarounds if you don't have write permission.
- Type in `library(MARSS)` at the R command line to load the package after you install it.
- To open up a copy of the case study script with the code you need to do the exercises, type `RShowDoc("Case_study_X.R",package="MARSS")` (with X replaced by the case study number).

Tips

- `summary(foo$model)`, where `foo` is a fitted model object, will print detailed information on the structure of the MARSS model that was fit in the call `foo = MARSS(logdata)`. This allows you to double check the model you fit. `print(foo)` will print a 'English' version of the model structure along with the parameter estimates.
- When you run `MARSS()`, it will output the number of iterations used. If you reached the maximum, re-run with `control=list(maxit=...)` set higher than the default.

- If you mis-specify the model, `MARSS()` will post an error that should give you an idea of the problem (make sure `silent=FALSE` to see full error reports). Remember, the number of rows in your data is n, time is across the columns, and the length of the vector or factors passed in for `model$Z` must be m, the number of x hidden state trajectories in your model.
- The missing value indicator is NA.
- Running `MARSS(data)`, with no arguments except your data, will fit a MARSS model with $m = n$, a diagonal $\mathbf{Q}$ matrix with m variances, and i.i.d. observation errors.

9

Case Study 1: Count-based population viability analysis (PVA) using corrupted data

9.1 Background

Estimates of extinction and quasi-extinction risk are an important risk metric used in the management and conservation of endangered and threatened species. By necessity, these estimates are based on data that contain both variability due to real year-to-year changes in the population growth rate (process errors) and variability in the relationship between the true population size and the actual count (observation errors). Classic approaches to extinction risk assume the data have only process error, i.e. no observation error. In reality, observation error is ubiquitous both because of the sampling variability and also because of year-to-year (and day-to-day) variability in sightability.

In this case study, we will fit a univariate (meaning one time series) state-space model to population count data with observation error. We will compute the extinction risk metrics given in Dennis et al. (1991), however instead of using a process-error only model (as is done in the original paper), we use a model with both process and observation error. The risk metrics and their interpretations are the same as in Dennis et al. (1991). The only real difference is how we compute $\boldsymbol{\sigma}^2$, the process error variance. However this difference has a large effect on our risk estimates, as you will see.

We use here a density-independent model, a stochastic exponential growth model in log space. This equivalent to a MARSS model with $\mathbf{B} = 1$. Density-independence is often a reasonable assumption when doing a population viability analysis because we do such calculations for at-risk populations that are either declining or that are well below historical levels (and presumably carrying capacity). In an actual population viability analysis, it is necessary to justify this assumption and if there is reason to doubt the assumption, one tests for density-dependence (Taper and Dennis, 1994) and does sensitivity analyses using state-space models with density-dependence (Dennis et al., 2006).

The univariate model is written:

$$x_t = x_{t-1} + u + w_t \qquad \text{where } w_t \sim \mathrm{N}(0, \sigma^2) \tag{9.1}$$

$$y_t = x_t + v_t \qquad \text{where } v_t \sim \mathrm{N}(0, \eta^2) \tag{9.2}$$

where y_t is the logarithm of the observed population size at time t, x_t is the unobserved state at time t, u is the growth rate, and σ^2 and η^2 are the process and observation error variances, respectively. In the R code to follow, σ^2 is denoted `Q` and η^2 is denoted `R` because the functions we are using are also for multivariate state-space models and those models use **Q** and **R** for the respective variance-covariance matrices.

9.2 Simulated data with process and observation error

We will start by using simulated data to see the difference between data and estimates from a model with process error only versus a model that also includes observation error. For our simulated data, we used a decline of 5% per year, process variability of 0.02 (typical for small to medium-sized vertebrates), and a observation variability of 0.05 (which is a bit on the high end). We'll randomly set 10% of the values as missing. Here is the code:

First, set things up:

```
sim.u = -0.05          # growth rate
sim.Q = 0.02           # process error variance
sim.R = 0.05           # non-process error variance
nYr= 50                # number of years of data to generate
fracmissing = 0.1      # fraction of years that are missing
init = 7               # log of initial pop abundance
years = seq(1:nYr)     # sequence 1 to nYr
x = rep(NA,nYr)        # replicate NA nYr times
y = rep(NA,nYr)
```

Then generate the population sizes using Equation 9.1:

```
x[1]=init
for(t in 2:nYr){
   x[t] = x[t-1]+ sim.u + rnorm(1,mean=0,sd=sqrt(sim.Q)) }
```

Lastly, add observation error using Equation 9.2 and then add missing values:

```
for(t in 1:nYr){
  y[t]= x[t] + rnorm(1,mean=0,sd=sqrt(sim.R))
}
missYears = sample(years[2:(nYr-1)],floor(fracmissing*nYr),
   replace = FALSE)
y[missYears]=NA
```

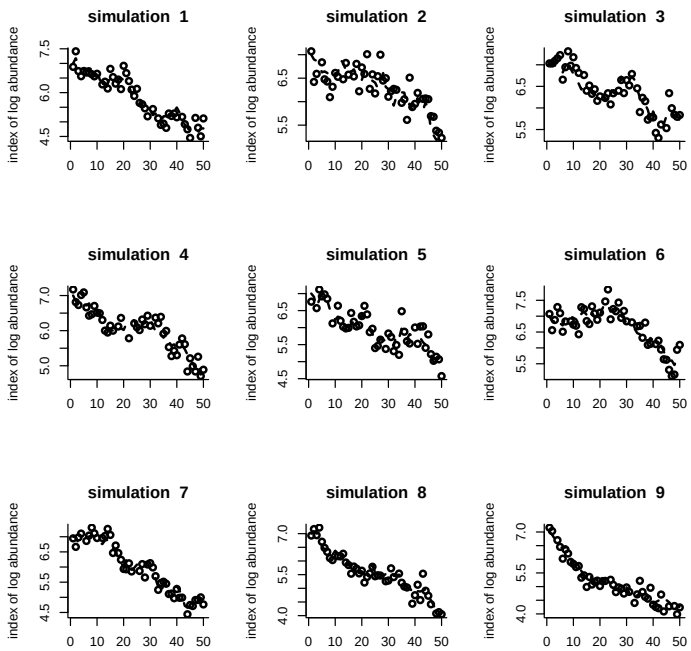

Fig. 9.1. Plot of nine simulated population time series with process and observation error. Circles are observation and the dashed line is the true population size.

Stochastic population trajectories show much variation, so it is best to look at a few simulated data sets at once. In Figure 9.1, nine simulations from the identical parameters are shown.

Example 9.1 (The effect of parameter values on parameter estimates)

A good way to get a feel for reasonable σ^2 *values is to generate simulated data and look at the time series. As a biologist, you probably have a pretty good idea of what kind of year-to-year population changes are reasonable for your species. For example for many large mammalian species, the maximum population yearly increase would be around 50% (the population could go from 1000 to 1500 in one year), but some of fish species could easily double or even triple in a really good year. Your observed data may bounce around a lot for many different reasons having to do with sightability, sampling error, age-structure, etc., but the underlying population trajectory is constrained by the*

kinds of year-to-year changes in population size that are biologically possible for your species. σ^2 *describes those true population changes.*

Run the exercise code several times using different parameter values to get a feel for how different the time series can look based on identical parameter values. You can cut and paste from the pdf into the R command line. Typical vertebrate σ^2 *values are 0.002 to 0.02, and typical* η^2 *values are 0.005 to 0.1. A* u *of -0.01 translates to an average 1% per year decline and a* u *of -0.1 translates to an average 10% per year decline (approximately).*

Example 9.1 code

Type `RShowDoc("Case_study_1.R",package="MARSS")` *to open a file with all the example code.*

```
par(mfrow=c(3,3))
sim.u = -0.05
sim.Q = 0.02
sim.R = 0.05
nYr= 50
fracmiss = 0.1
init = 7
years = seq(1:nYr)
for(i in 1:9){
  x = rep(NA,nYr) # vector for ts w/o measurement error
  y = rep(NA,nYr) # vector for ts w/ measurement error
  x[1]=init
  for(t in 2:nYr){
    x[t] = x[t-1]+ sim.u + rnorm(1, mean=0, sd=sqrt(sim.Q)) }
  for(t in 1:nYr){
    y[t]= x[t] + rnorm(1,mean=0,sd=sqrt(sim.R)) }
  missYears =
    sample(years[2:(nYr-1)],floor(fracmiss*nYr),replace = FALSE)
  y[missYears]=NA
  plot(years, y,
    xlab="",ylab="log abundance",lwd=2,bty="l")
  lines(years,x,type="l",lwd=2,lty=2)
  title(paste("simulation ",i) )
}
legend("topright", c("Observed","True"),
  lty = c(-1, 2), pch = c(1, -1))
```

9.3 Maximum-likelihood parameter estimation

9.3.1 Model with process and observation error

Using the simulated data, we estimate the parameters, u, σ^2, and η^2, and the hidden population sizes. These are the estimates using a model with process and observation variability. The function call is `kem = MARSS(data)`, where `data` is a vector of logged (base e) counts with missing values denoted by NA. After this call, the maximum-likelihood parameter estimates are shown with `coef(kem)`. There are numerous other outputs from the `MARSS()` function. To get a list of the standard model output available type in `?print.MARSS`. Note that `kem` is just a name; the output could have been called `foo`. Here's code to fit to the simulated time series:

```
kem = MARSS(y)
```

Let's look at the parameter estimates for the nine simulated time series in Figure 9.1 to get a feel for the variation. The `MARSS()` function was used on each time series to produce parameter estimate for each simulation. The estimates are followed by the mean (over the nine simulations) and the true values:

```
               kem.U       kem.Q      kem.R
sim 1    -0.04460270 0.012935709 0.04625213
sim 2    -0.02668878 0.006056401 0.05441229
sim 3    -0.02542762 0.017356063 0.02729418
sim 4    -0.04464406 0.015976031 0.03200482
sim 5    -0.04019964 0.018796592 0.07106619
sim 6    -0.02369627 0.023012045 0.06096778
sim 7    -0.04293795 0.020232637 0.02036582
sim 8    -0.06064397 0.016360664 0.03766525
sim 9    -0.05965552 0.011851461 0.03120967
mean sim -0.04094406 0.015841956 0.04235979
true     -0.05000000 0.020000000 0.05000000
```

As expected, the estimated parameters do not exactly match the true parameters, but the average should be fairly close (although nine simulations is a small sample size). Also note that although we do not get u quite right, our estimates are usually negative. Thus our estimates usually indicate declining dynamics. Some of the `kem.Q` estimates may be 0. This means that the maximum-likelihood estimate that the data are generated by is a process with no environment variation and only observation error.

The MARSS model fit also gives an estimate of the true population size with observation error removed. This is in `kem$states`. Figure 9.2 shows the estimated true states of the population over time as a solid line. Note that the solid line is considerably closer to the actual true states (dashed line) than the observations. On the other hand with certain datasets, the estimates can be quite wrong as well!

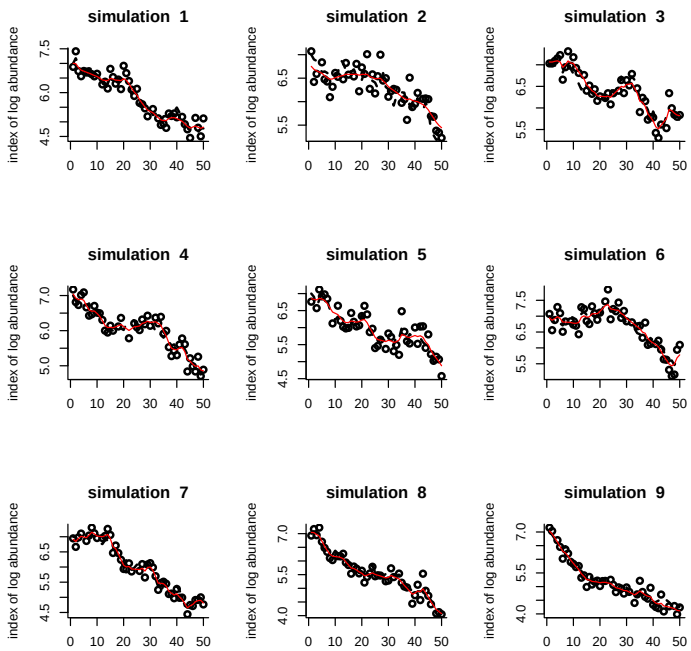

Fig. 9.2. The circles are the observed population sizes with error. The dashed lines are the true population sizes. The solid thin lines are the estimates of the true population size from the MARSS model. When the process error variance is 0, these lines are straight.

9.3.2 Model with no observation error

We used the MARSS model to estimate the mean population rate u and process variability σ^2 under the assumption that the count data have observation error. However, the classic approach to this problem, referred to as the "Dennis model" (Dennis et al., 1991), uses a model that assumes the data have no observation error (a MAR model); all the variability in the data is assumed to result from process error. This approach works well if the observation error in the data is low, but not so well if the observation error is high. We will next fit the data using the classic approach so that we can compare and contrast parameter estimates from the different methods.

Using the estimation method in Dennis et al. (1991), our data need to be re-specified as the observed population changes (`delta.pop`) between censuses along with the time between censuses (`tau`). We re-specify the data as follows:

```
den.years = years[!is.na(y)] # the non missing years
den.y = y[!is.na(y)] # the non missing counts
```

```
den.n.y = length(den.years)
delta.pop = rep(NA, den.n.y-1 ) # population transitions
tau = rep(NA, den.n.y-1 ) # step sizes
for (i in 2:den.n.y ){
  delta.pop[i-1] = den.y[i] - den.y[i-1]
  tau[i-1] =  den.years[i] - den.years[i-1]
} # end i loop
```

Next, we regress the changes in population size between censuses (`delta.pop`) on the time between censuses (`tau`) while setting the regression intercept to 0. The slope of the resulting regression line is an estimate of u, while the variance of the residuals around the line is an estimate of σ^2. The regression is shown in Figure 9.3. Here is the code to do that regression:

```
den91 <- lm(delta.pop ~ -1 + tau)
# note: the "-1" specifies no intercept
den91.u = den91$coefficients
den91.Q = var(resid(den91))
#type ?lm to learn about the linear regression function in R
#form is lm(dependent.var ~ response.var1 + response.var2 + ...)
#type summary(den91) to see other info about our regression fit
```

Here are the parameter values for the data in Figure 9.2 using the process-error only model:

```
             den91.U    den91.Q
sim 1    -0.02537667 0.10978359
sim 2    -0.03551612 0.12294854
sim 3    -0.02780081 0.07580098
sim 4    -0.04377727 0.08457376
sim 5    -0.04040716 0.16378747
sim 6    -0.01328259 0.13919017
sim 7    -0.04819441 0.06301253
sim 8    -0.06347384 0.09538016
sim 9    -0.05797770 0.07972705
mean sim -0.03953406 0.10380047
true     -0.05000000 0.02000000
```

Notice that the u estimates are similar to those from MARSS model, but the σ^2 estimate (`Q`) is much larger. That is because this approach treats all the variance as process variance, so any observation variance in the data is lumped into process variance (in fact it appears as an additional variance of twice the observation variance).

Example 9.2 (The variability in parameter estimates)

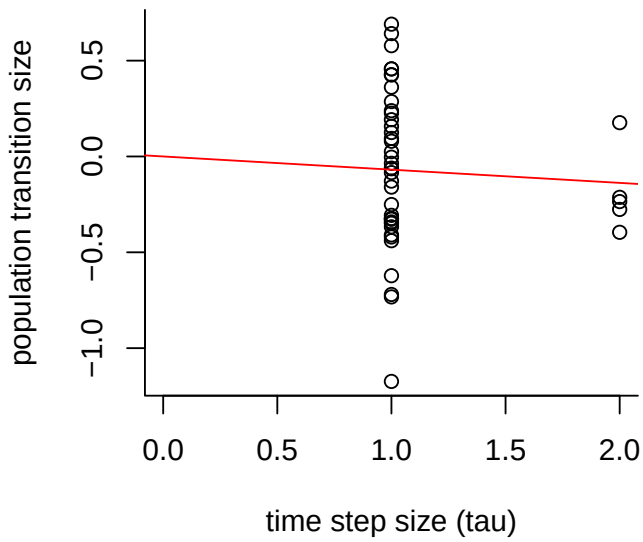

Fig. 9.3. The regression of $log(N_{t+\tau}) - log(N_t)$ against τ. The slope is the estimate of u and the variance of the residuals is the estimate of σ^2.

In this example, you will look at how variable the parameter estimates are by generating multiple simulated data sets and then estimating parameter values for each. You'll compare the MARSS estimates to the estimates using a process error only model (i.e. ignoring the observation error).

Example 9.2 code

Type `RShowDoc("Case_study_1.R",package="MARSS")` *to open a file with all the example code. You will not be able to edit this file. To edit, copy and paste into a new script file.*

```
sim.u = -0.05   # growth rate
sim.Q = 0.02    # process error variance
sim.R = 0.05    # non-process error variance
nYr= 50         # number of years of data to generate
fracmiss = 0.1  # fraction of years that are missing
init = 7        # log of initial pop abundance (~1100 individuals)
nsim = 9
years = seq(1:nYr)  # col of years
params = matrix(NA, nrow=(nsim+2), ncol=5,
  dimnames=list(c(paste("sim",1:nsim),"mean sim","true"),
c("kem.U","den91.U","kem.Q","kem.R", "den91.Q")))
x.ts = matrix(NA,nrow=nsim,ncol=nYr)  # ts w/o measurement error
y.ts = matrix(NA,nrow=nsim,ncol=nYr)  # ts w/ measurement error
for(i in 1:nsim){
  x.ts[i,1]=init
  for(t in 2:nYr){
    x.ts[i,t] = x.ts[i,t-1]+sim.u+rnorm(1,mean=0,sd=sqrt(sim.Q))}
  for(t in 1:nYr){
    y.ts[i,t] = x.ts[i,t]+rnorm(1,mean=0,sd=sqrt(sim.R))}
  missYears = sample(years[2:(nYr-1)], floor(fracmiss*nYr),
    replace = FALSE)
  y.ts[i,missYears]=NA

  #MARSS estimates
  kem = MARSS(y.ts[i,], silent=TRUE)
  #type=vector outputs the estimates as a vector instead of a list
  params[i,c(1,3,4)] = coef(kem,type="vector")[c(2,3,1)]

  #Dennis et al 1991 estimates
  den.years = years[!is.na(y.ts[i,])]  # the non missing years
  den.yts = y.ts[i,!is.na(y.ts[i,])]   # the non missing counts
  den.n.yts = length(den.years)
  delta.pop = rep(NA, den.n.yts-1 ) # transitions
  tau = rep(NA, den.n.yts-1 )       # time step lengths
  for (t in 2:den.n.yts ){
    delta.pop[t-1] = den.yts[t] - den.yts[t-1] # transitions
    tau[t-1] =  den.years[t]-den.years[t-1]    # time step length
  } # end i loop
  den91 <- lm(delta.pop ~ -1 + tau) # -1 specifies no intercept
  params[i,c(2,5)] = c(den91$coefficients, var(resid(den91)))
}
params[nsim+1,]=apply(params[1:nsim,],2,mean)
params[nsim+2,]=c(sim.u,sim.u,sim.Q,sim.R,sim.Q)
```

Here is an example of the output from the code:

```
print(params,digits=3)
```

```
            kem.U   den91.U   kem.Q  kem.R den91.Q
sim 1    -0.03967 -0.015257 0.00907 0.0585  0.1358
sim 2    -0.00123 -0.000224 0.01551 0.0589  0.1688
sim 3    -0.05314 -0.027060 0.03940 0.0278  0.1035
sim 4    -0.01803 -0.016884 0.01264 0.0233  0.0620
sim 5    -0.04067 -0.052789 0.01490 0.0620  0.1693
sim 6    -0.03327 -0.024551 0.02080 0.0441  0.1097
sim 7    -0.04581 -0.051013 0.03977 0.0338  0.1213
sim 8    -0.07351 -0.066388 0.00000 0.0534  0.1085
sim 9    -0.03767 -0.037336 0.00000 0.0531  0.0964
mean sim -0.03811 -0.032389 0.01690 0.0461  0.1195
true     -0.05000 -0.050000 0.02000 0.0500  0.0200
```

1. *Re-run the code a few times to see the performance of the estimates using a state-space model (*`kem`*) versus the model with no observation error (*`den91`*). You can copy and paste the code from the pdf file into R .*

2. *Alter the observation variance,* `sim.R`*, in the data generation step in order to get a feel for performance as observations are further corrupted. What happens as observation error is increased?*

3. *Decrease the number of years of data,* `nYr`*, and re-run the parameter estimation. What changes?*

If you find that the exercise code takes too long to run, reduce the number of simulations (by reducing `nsim` *in the code).*

9.4 Probability of hitting a threshold $\Pi(x_d, t_e)$

A common extinction risk metric is 'the probability that a population will hit a certain threshold x_d within a certain time frame t_e – *if the observed trends continue*'. In practice, the threshold used is not $N_e = 1$, which would be true extinction. Often a 'functional' extinction threshold will be used ($N_e >> 1$). Other times a threshold representing some fraction of current levels is used. The latter is used because we often have imprecise information about the relationship between the true population size and what we measure in the field; that is, many population counts are index counts. In these cases, one

must use 'fractional declines' as the threshold. Also, extinction estimates that use an absolute threshold (like 100 individuals) are quite sensitive to error in the estimate of true population size. Here, we are going to use fractional declines as the threshold, specifically $p_d = 0.1$ which means a 90% decline.

The probability of hitting a threshold, denoted $\Pi(x_d,t_e)$, is typically presented as a curve showing the probabilities of hitting the threshold (y-axis) over different time horizons (t_e) on the x-axis. Extinction probabilities can be computed through Monte Carlo simulations or analytically using Equation 16 in Dennis et al. (1991) (note there is a typo in Equation 16; the last + is supposed to be a −). We will use the latter method:

$$\Pi(x_d,t_e) = \pi(u) \times \Phi\left(\frac{-x_d + |u|t_e}{\sqrt{\sigma^2 t_e}}\right) + \exp(2x_d|u|/\sigma^2)\Phi\left(\frac{-x_d - |u|t_e}{\sqrt{\sigma^2 t_e}}\right) \quad (9.3)$$

where x_e is the threshold and is defined as $x_e = \log(N_0/N_e)$. N_0 is the current population estimate and N_e is the threshold. If we are using fractional declines then $x_e = \log(N_0/(p_d \times N_0)) = -\log(p_d)$. $\pi(u)$ is the probability that the threshold is eventually hit (by $t_e = \infty$). $\pi(u) = 1$ if $u <= 0$ and $\pi(u) = \exp(-2ux_d/\sigma^2)$ if $u > 0$. $\Phi()$ is the cumulative probability distribution of the standard normal (mean = 0, sd = 1).

Here is the R code for that computation:

```
pd = 0.1 #means a 90 percent decline
tyrs = 1:100
xd = -log(pd)
p.ever = ifelse(u<=0,1,exp(-2*u*xd/Q)) #Q=sigma2
for (i in 1:100){
 Pi[i] = p.ever * pnorm((-xd+abs(u)*tyrs[i])/sqrt(Q*tyrs[i]))+
   exp(2*xd*abs(u)/Q)*pnorm((-xd-abs(u)*tyrs[i])/sqrt(Q*tyrs[i]))
}
```

Figure 9.4 shows the estimated probabilities of hitting the 90% decline for the nine 30-year times series simulated with $u = -0.05$, $\sigma^2 = 0.01$ and $\eta^2 = 0.05$. The dashed line shows the estimates using the MARSS parameter estimates and the solid line shows the estimates using a process-error only model (the `den91` estimates). The circles are the true probabilities. The difference between the estimates and the true probalities is due to errors in $\hat{u}$. Those errors are due largely to process error—not observation error. As we saw earlier, by chance population trajectories with a $u < 0$ will increase, even over a 50-year period. In this case, $\hat{u}$ will be positive when in fact $u < 0$.

Looking at the figure, it is obvious that the probability estimates are highly variable. However, look at the first panel. This is the average estimate (over nine simulations). Note that on average (over nine simulations), the estimates are good. If we had averaged over 1000 simulations instead of nine, you would see that the MARSS line falls on the true line. It is an unbiased predictor. While that may seem a small consolation if estimates for individual simulations

are all over the map, it is important for correctly specifying our uncertainty about our estimates. Second, rather than focusing on how the estimates and true lines match up, see if there are any types of forecasts that seem better than others. For example, are 20-year predictions better than 50-year and are 100-year forecasts better or worse. In Exercise 3, you will remake this figure with different u. You'll discover from that forecasts are more certain for populations that are declining faster.

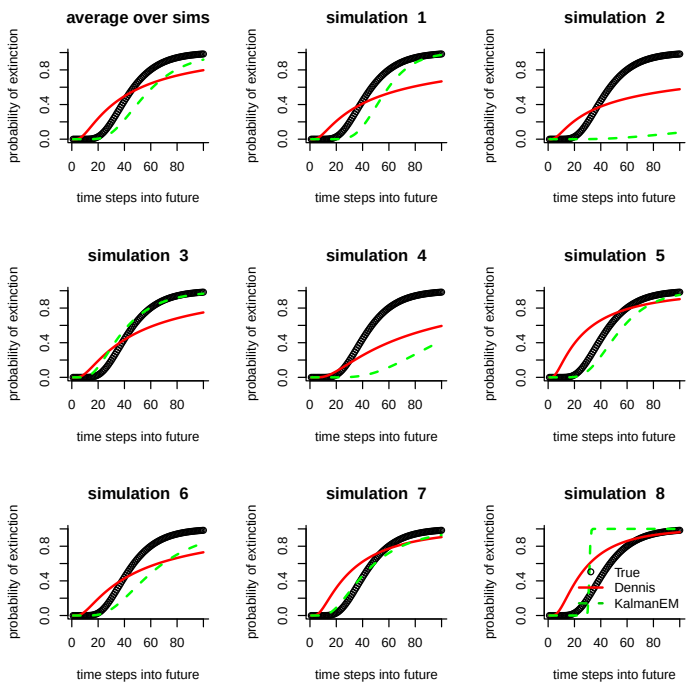

Fig. 9.4. Plot of the true and estimated probability of declining 90% in different time horizons for nine simulated population time series with observation error. The plot may look like a step-function if the σ^2 estimate is very small (<1e-4 or so).

Example 9.3 (The effect of parameter values on risk estimates)

In this example, you will recreate Figure 9.4 using different parameter values. This will give you a feel for how variability in the data and population process affect the risk estimates. You'll need to run the Example 9.2 code before running the Example 9.3 code.

Example 9.3 code

Type `RShowDoc("Case_study_1.R",package="MARSS")` *to open a file with all the example code.*

```
#Needs Exercise 2 to be run first
par(mfrow=c(3,3))
pd = 0.1; xd = -log(pd)   # decline threshold
te = 100; tyrs = 1:te   # extinction time horizon
for(j in c(10,1:8)){
  real.ex = denn.ex = kal.ex = matrix(nrow=te)

  #MARSS parameter estimates
  u=params[j,1];   Q=params[j,3]
  if(Q==0) Q=1e-4  #just so the extinction calc doesn't choke
  p.ever = ifelse(u<=0,1,exp(-2*u*xd/Q))
  for (i in 1:100){
    if(is.finite(exp(2*xd*abs(u)/Q))){
          sec.part = exp(2*xd*abs(u)/Q)*pnorm((-xd-abs(u)* tyrs[i])/sqrt(Q*tyrs[i]))
    }else sec.part=0
    kal.ex[i]=p.ever*pnorm((-xd+abs(u)*tyrs[i])/sqrt(Q*tyrs[i]))+sec.part
  } # end i loop

  #Dennis et al 1991 parameter estimates
  u=params[j,2];   Q=params[j,5]
  p.ever = ifelse(u<=0,1,exp(-2*u*xd/Q))
  for (i in 1:100){
    denn.ex[i]=p.ever*pnorm((-xd+abs(u)*tyrs[i])/sqrt(Q*tyrs[i]))+
      exp(2*xd*abs(u)/Q)*pnorm((-xd-abs(u)*tyrs[i])/sqrt(Q*tyrs[i]))
  } # end i loop

  #True parameter values
  u=sim.u;   Q=sim.Q
  p.ever = ifelse(u<=0,1,exp(-2*u*xd/Q))
  for (i in 1:100){
    real.ex[i]=p.ever*pnorm((-xd+abs(u)*tyrs[i])/sqrt(Q*tyrs[i]))+
      exp(2*xd*abs(u)/Q)*pnorm((-xd-abs(u)*tyrs[i])/sqrt(Q*tyrs[i]))
  } # end i loop

  #plot it
  plot(tyrs, real.ex, xlab="time steps into future",
    ylab="probability of extinction", ylim=c(0,1), bty="l")
  if(j<=8) title(paste("simulation ",j) )
  if(j==10) title("average over sims")
  lines(tyrs,denn.ex,type="l",col="red",lwd=2,lty=1)
  lines(tyrs,kal.ex,type="l",col="green",lwd=2,lty=2)
}
legend("bottomright",c("True","Dennis","KalmanEM"),pch=c(1,-1,-1),
  col=c(1,2,3),lty=c(-1,1,2),lwd=c(-1,2,2),bty="n")
```

1. *Change* `sim.R` *and rerun the Example 9.2 code. Then run the Example 9.3 code. When are the estimates using the process-error only model (*`den91`*) worse and in what way are they worse?*

2. *You might imagine that you should always use a model that includes observation error, since in practice observations are never perfect. However, there is a cost to estimating that extra variance parameter and the cost is a more variable* σ^2 *(*`Q`*) estimate. Play with shortening the time series and decreasing the* `sim.R` *values. Are there situations when the 'cost' of the extra parameter is greater than the 'cost' of ignoring observation error?*

3. *How does changing the extinction threshold (*`pd`*) change the extinction probability curves? (Do not remake the data, i.e. don't rerun the Example 9.2 code.)*

4. *How does changing the rate of decline (*`sim.u`*) change the estimates of risk? Rerun the Example 9.2 code using a lower* `u`*; this will create a new matrix of parameter estimates. Then run the Example 9.3 code. Do the estimates seem better of worse for rapidly declining populations?*

5. *Rerun the Example 9.2 code using fewer number of years (*`nYr` *smaller) and increase* `fracmiss`*. Then run the Example 9.3 code. The graphs will start to look peculiar. Why do you think it is doing that? Hint: look at the estimated parameters.*

9.5 Certain and uncertain regions

From Example 9.3, you have observed one of the problems with estimates of the probability of hitting thresholds. Looking over the nine simulations, your risk estimates will be on the true line sometimes and other times they are way off. So your estimates are variable and one should not present only the point estimates of the probability of 90% decline. At the minimum, confidence intervals need to be added (next section), but even with confidence intervals, the probability of hitting declines often does not capture our certainty and uncertainty about extinction risk estimates.

From Example 9.3, you might have also noticed that there are some time horizons (10, 20 years) for which the estimate are highly certain (the threshold is never hit), while for other time horizons (30, 50 years) the estimates are all over the map. Put another way, you may be able to say with high confidence that a 90% decline will not occur between years 1 to 20 and that by year 100 it most surely will have occurred. However, between the years 20 and 100, you are very uncertain about the risk. The point is that you can be certain about some forecasts while at the same time being uncertain about other forecasts.

One way to show this is to plot the uncertainty as a function of the forecast, where the forecast is defined in terms of the forecast length (number of years) and forecasted decline (percentage). Uncertainty is defined as how much of the 0-1 range your 95% confidence interval covers. Ellner and Holmes (2008) show such a figure (their Figure 1). Figure 9.5 shows a version of this figure that you can produce with the function `CSEGtmufigure(u= val, N= val, s2p= val)`. For the figure, the values $u = -0.05$ which is a 5% per year decline, $N = 25$ so 25 years between the first and last census, and $s_p^2 = 0.01$ are used. The process variability for big mammals is typically in the range of 0.002 to 0.02.

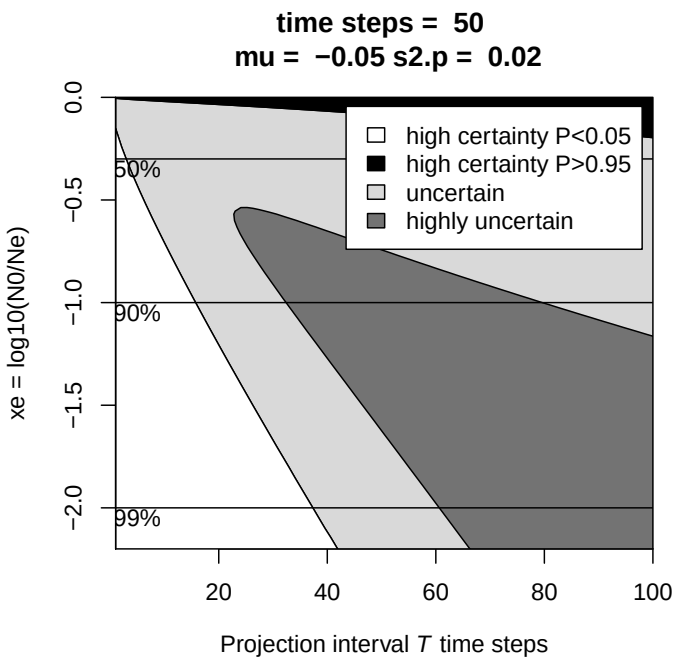

Fig. 9.5. This figure shows your region of high uncertainty (dark gray). In this region, the minimum 95% confidence intervals (meaning if you had no observation error) span 80% of the 0 to 1 probability. That is, you are uncertain if the probability of a specified decline is close to 0 or close to 1. The white area shows where your upper 95% CIs does not exceed P=0.05. So you are quite sure the probability of a specified decline is less than 0.05. The black area shows where your lower 95% confidence interval is above P=.95. So you are quite sure the probability is greater than P=0.95. The light gray is between these two certain/uncertain extremes.

Example 9.4 (Uncertain and certain regions)

Use the Example 9.4 code to re-create Figure 9.5 and get a feel for when risk estimates are more certain and when they are less certain. `N` *are the number of years of data,* `u` *is the mean population growth rate, and* `s2p` *is the process variance.*

Exercise 9.4 code

Type `RShowDoc("Case_study_1.R",package="MARSS")` *to open a file with all the example code.*

```
par(mfrow = c(1, 1))
CSEGtmufigure(N = 50, u = -0.05, s2p = 0.02)
```

9.6 More risk metrics and some real data

The previous sections have focused on the probability of hitting thresholds because this is an important and common risk metric used in population viability analysis and it appears in IUCN Red List criteria. However, as you have seen, there is high uncertainty associated with such estimates. Part of the problem is that probability is constrained to be 0 to 1, and it is easy to get estimates with confidence intervals that span 0 to 1. Other metrics of risk, $\hat{u}$ and the distribution of the time to hit a threshold (Dennis et al., 1991), do not have this problem and may be more informative. Figure 9.6 shows different risk metrics from Dennis et al. (1991) on a single plot. This figure is generated by a call to the function `CSEGriskfigure()`:

```
dat=read.table(datafile, skip=1)
dat=as.matrix(dat)
CSEGriskfigure(dat)
```

The `datafile` is the name of the data file, with years in column 1 and population count (logged) in column 2. `CSEGriskfigure()` has a number of arguments that can be passed in to change the default behavior. The variable `te` is the forecast length (default is 100 years), `threshold` is the extinction threshold either as an absolute number, if `absolutethresh=TRUE`, or as a fraction of current population count, if `absolutethresh=FALSE`. The default is `absolutethresh=FALSE` and `threshold=0.1`. `datalogged=TRUE` means the data are already logged; this is the default.

Example 9.5 (Risk figures for different species)

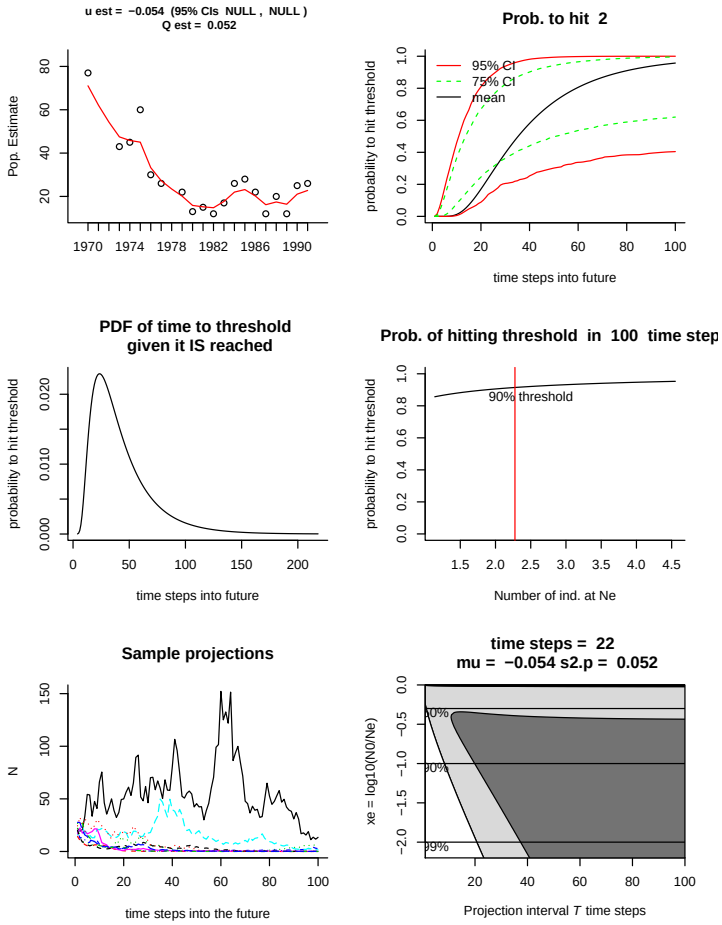

Fig. 9.6. Risk figure using data for the critically endangered African Wild Dog (data from Ginsberg et al. 1995). This population went extinct after 1992.

Use the Example 9.5 code to re-create Figure 9.6. The package includes other data for you to run: `prairiechicken` *from the endangered Attwater Prairie Chicken,* `graywhales` *from Gerber et al. (1999), and* `grouse` *from the Sharp-tailed Grouse (a species of U.S. federal concern) in Washington State. Note for some of these other datasets, the Hessian matrix cannot be inverted and you will need to use* `CI.method="parametric"`*. If you have other text files of data, you can run those too. The commented lines show how to read in data from a tab-delimited text file with a header line.*

Exercise 5 code
Type `RShowDoc("Case_study_1.R",package="MARSS")` *to open a file with the example code.*

```
#If you have your data in a tab delimited file with a header
#This is how you would read it in using file.choose()
#to call up a directory browser.
#However, the package has the datasets for the exercises
#dat=read.table(file.choose(), skip=1)
#dat=as.matrix(dat)
dat = wilddogs
CSEGriskfigure(dat, CI.method="hessian", silent=TRUE)
```

9.7 Confidence intervals

The figures produced by `CSEGriskfigure()` have confidence intervals (95% and 75%) on the probabilities in the top right panel. A standard way to produce these intervals is via parametric bootstrapping. Here are the steps in a parametric bootstrap:

- You estimate u, σ^2 and η^2
- Then you simulate time series using those estimates and Equations 9.1 and 9.2
- Then you re-estimate your parameters from the simulated data (using say `MARSS(simdata)`
- Repeat for 1000s of time series simulated using your estimated parameters. This gives you a large set of bootstrapped parameter estimates
- For each bootstrapped parameter set, compute a set of extinction estimates (you use Equation 9.3 and code from Example 9.3)
- The α% ranges on those bootstrapped extinction estimates gives you your α confidence intervals on your probabilities of hitting thresholds

The MARSS package provides the function `MARSSparamCIs()` to add bootstrapped confidence intervals to fitted models (type `?MARSSparamCIs` to learn about the function).

In the function `CSEGriskfigure()`, you can set `CI.method = c("hessian", "parametric", "innovations", "none")` to tell it how to compute the confidence intervals. The methods 'parametric' and 'innovations' specify parametric and non-parametric bootstrapping respectively. Producing parameter estimates by bootstrapping is quite slow. Approximate confidence intervals on the parameters can be generated rapidly using the inverse of a numerically estimated Hessian matrix (method 'hessian'). This uses an estimate of the

variance-covariance matrix of the parameters (the inverse of the Hessian matrix). Using an estimated Hessian matrix to compute confidence intervals is a handy trick that can be used for all sorts of maximum-likelihood parameter estimates.

9.8 Comments

Data with cycles, from age-structure or predator-prey interactions, are difficult to analyze and the EM algorithm used in the MARSS package will give poor estimates for this type of data. The slope method (Holmes, 2001) is more robust to those problems. Holmes et al. (2007) used the slope method in a large study of data from endangered and threatened species, and Ellner and Holmes (2008) showed that the slope estimates are close to the theoretical minimum uncertainty. Especially, when doing a population viability analysis using a time series with fewer than 25 years of data, the slope method is often less biased and (much) less variable because that method is less data-hungry (Holmes, 2004). However the slope method is not a true maximum-likelihood method and thus constrains the types of further analyses you can do (such as model selection).

10

Case study 2: Combining multi-site data to estimate regional population trends

10.1 Harbor seals in the Puget Sound, WA.

In this case study, we will use multivariate state-space models to combine surveys from multiple regions (or sites) into one estimate of the average long-term population growth rate and the year-to-year variability in that growth rate. Note this is not quite the same as estimating the 'trend'; 'trend' often means what population change happened, whereas the long-term population growth rate refers to the underlying population dynamics. We will use as our example a dataset from harbor seals in Puget Sound, Washington, USA.

We have five regions (or sites) where harbor seals were censused from 1978-1999 while hauled out of land[1]. During the period of this dataset, harbor seals were recovering steadily after having been reduced to low levels by hunting prior to protection. The methodologies were consistent throughout the 20 years of the data but we do not know what fraction of the population that each region represents nor do we know the observation-error variance for each region. Given differences between behaviors of animals in different regions and the numbers of haul-outs in each region, the observation errors may be quite different. The regions have had different levels of sampling; the best sampled region has only 4 years missing while the worst has over half the years missing (Figure 10.1).

For this case study, we will assume that the underlying population process is a stochastic exponential growth process with rates of increase that were not changing through 1978-1999. However, we are not sure if all five regions sample a single "total Puget Sound" population or if there are independent subpopulations. We will estimate the long-term population growth rate using different assumptions about the population structures (one big population versus multiple smaller ones) and observation error structures to see how different assumptions change the trend estimates.

[1] Jeffries et al. 2003. Trends and status of harbor seals in Washington State: 1978-1999. Journal of Wildlife Management 67(1):208–219

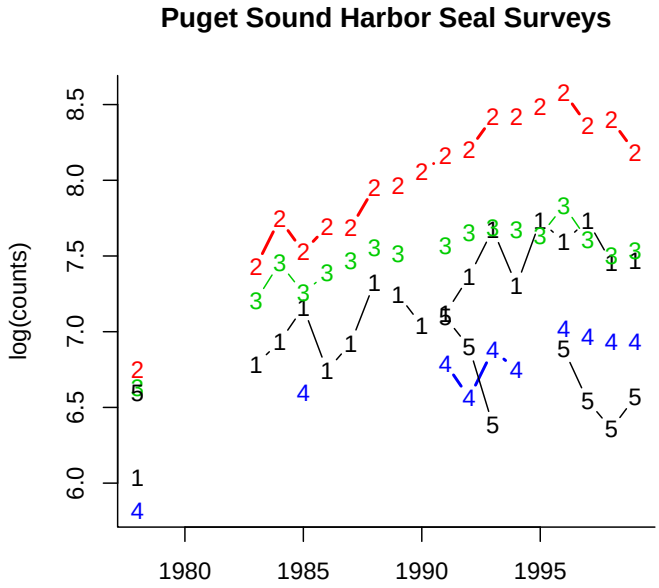

Fig. 10.1. Plot of the of the count data from the five harbor seal regions (Jeffries et al. 2003). The numbers on each line denote the different regions: 1) Strait of Juan de Fuca (SJF), 2) San Juan Islands (SJI), 2) Eastern Bays (EBays), 4) Puget Sound (PSnd), and 5) Hood Canal (HC). Each region is an index of the total harbor seal population, but the bias (the difference between the index and the true population size) for each region is unknown.

The data for this case study are in the MARSS package. The data have time running down the rows and years in the first column. We need time across the columns for the `MARSS()` function, so we will transpose the data:

```
dat=t(harborSealWA) #Transpose
years = dat[1,] #[1,] means row 1
n = nrow(dat)-1
dat = dat[2:nrow(dat),] #no years
```

If you needed to read data in from a comma-delimited or tab-delimited file, these are the commands to do that:

```
dat = read.csv("datafile.csv",header=TRUE)
dat = read.table("datafile.csv",header=TRUE)
```

The years are in column 1 of `dat` and the logged data are in the rest of the columns. The number of observation time series (`n`) is the number of rows in `dat` minus 1 (for years row). Let's look at the first few years of data:

```
print(harborSealWA[1:8,], digits=3)
```

```
     Year  SJF  SJI EBays PSnd  HC
[1,] 1978 6.03 6.75  6.63 5.82 6.6
[2,] 1979   NA   NA    NA   NA  NA
[3,] 1980   NA   NA    NA   NA  NA
[4,] 1981   NA   NA    NA   NA  NA
[5,] 1982   NA   NA    NA   NA  NA
[6,] 1983 6.78 7.43  7.21   NA  NA
[7,] 1984 6.93 7.74  7.45   NA  NA
[8,] 1985 7.16 7.53  7.26 6.60  NA
```

The `NA`'s in the data are missing values.

10.2 A single well-mixed Puget Sound population

The first step is to mathematically specify the population structure and how the regions relate to that structure. The general state-space model is

$$\begin{aligned}\mathbf{x}_t &= \mathbf{B}\mathbf{x}_{t-1} + \mathbf{u} + \mathbf{w}_t, \text{ where } \mathbf{w}_t \sim \text{MVN}(0, \mathbf{Q}) \\ \mathbf{y}_t &= \mathbf{Z}\mathbf{x}_t + \mathbf{a} + \mathbf{v}_t, \text{ where } \mathbf{v}_t \sim \text{MVN}(0, \mathbf{R})\end{aligned}$$

where all the bolded symbols are matrices. To specify the structure of the population and observations, we will specify what those matrices look like.

10.2.1 The population process, x

When we are looking at data over a large geographic region, we might make the assumption that the different census regions are measuring a single population if we think animals are moving sufficiently such that the whole area (multiple regions together) is "well-mixed". We write a model of the total population abundance as:

$$n_t = \exp(u + w_t) n_{t-1}, \tag{10.1}$$

where n_t is the total count in year t, u is the mean population growth rate, and w_t is the deviation from that average in year t. We then take the log of both sides and write the model in log space:

$$x_t = x_{t-1} + u + w_t, \text{ where } w_t \sim \text{N}(0, q) \tag{10.2}$$

$x_t = \log n_t$. When there is one effective population, there is one x, therefore $\mathbf{x}_t$ is a 1×1 matrix. There is one population growth rate (u) and there is one process variance (q). Thus $\mathbf{u}$ and $\mathbf{Q}$ are 1×1 matrices.

10.2.2 The observation process, y

For this first analysis, we assume that all five regional time series are observing this one population trajectory but they are scaled up or down relative to that trajectory. In effect, we think that animals are moving around a lot and our regional samples are some fraction of the population. There is year-to-year variation in the fraction in each region, just by chance. Notice that under this analysis, we do not think the regions represent independent subpopulations but rather independent observations of one population. Our model for the data, $\mathbf{y}_t = \mathbf{Z}\mathbf{x}_t + \mathbf{a} + \mathbf{v}_t$, is written as:

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix}_t = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} x_t + \begin{bmatrix} 0 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix} + \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{bmatrix}_t \tag{10.3}$$

Each y_i is the time series for a different region. The a's are the bias between the regional sample and the total population. The a's are scaling (or intercept-like) parameters[2]. We allow that each region could have a unique observation variance and that the observation errors are independent between regions. Lastly, we assume that the observations errors on log(counts) are normal and thus the errors on (counts) are log-normal.[3]

We specify independent observation errors with unique variances by specifying that the v's come from a multivariate normal distribution with variance-covariance matrix $\mathbf{R}$ ($\mathbf{v} \sim \mathrm{MVN}(0, \mathbf{R})$), where

$$\mathbf{R} = \begin{bmatrix} r_1 & 0 & 0 & 0 & 0 \\ 0 & r_2 & 0 & 0 & 0 \\ 0 & 0 & r_3 & 0 & 0 \\ 0 & 0 & 0 & r_4 & 0 \\ 0 & 0 & 0 & 0 & r_5 \end{bmatrix} \tag{10.4}$$

$\mathbf{Z}$ specifies which observation time series, $y_{i,1:T}$, is associated with which population trajectory, $x_{j,1:T}$. $\mathbf{Z}$ is like a look up table with 1 row for each of the n observation time series and 1 column for each of the m population

[2] To get rid of the a's, we scale multiple observation time series against each other; thus one a will be fixed at 0. Estimating the bias between regional indices and the total population is important for getting an estimate of the total population size. The type of time-series analysis that we are doing here (trend analysis) is not useful for estimating a's. Instead to get a's one would need some type of mark-recapture data. However, for trend estimation, the a's are not important. The regional observation variance captures increased variance due to a regional estimate being a smaller sample of the total population.

[3] The assumption of normality is not unreasonable since these regional counts are the sum of counts across multiple haul-outs.

trajectories. A 1 in row i column j means that observation time series i is measuring state process j. Otherwise the value in $\mathbf{Z}_{ij} = 0$. Since we have only 1 population trajectory, all the regions must be measuring that one population trajectory. Thus $\mathbf{Z}$ is $n \times 1$:

$$\mathbf{Z} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \tag{10.5}$$

10.2.3 Setting the model structure for `MARSS`

Now that we have specified our state-space model, we set the arguments that will tell the function `MARSS()` the structure of our model. We do this by passing in the argument `model` to `MARSS()`. The argument `model` is a list which specifies the model structure for $\mathbf{Z}$, $\mathbf{u}$, $\mathbf{Q}$, etc. The function call will now look like:

```
kem1 = MARSS(dat, model=list(Z=Z.model, U=U.model,
      Q=Q.model, R=R.model) )
```

First we set the $\mathbf{Z}$ model. We need to tell the `MARSS` function that $\mathbf{Z}$ is a 5×1 matrix of 1s (as in Equation 10.3). We can do this two ways. We can pass in `Z.model` as a matrix of ones, `matrix(1,5,1)`, just like in Equation (10.3) or we can pass in a vector of five factors, `factor(c(1,1,1,1,1))`. The i-th factor specifies which population trajectory the i-th observation time series belongs to. Since there is only one population trajectory in this first analysis, we will have a vector of five 1's: every observation time series is measuring the first, and only, population trajectory.

```
Z.model = factor(c(1,1,1,1,1))
```

Note, the vector (the `c()` bit) must be wrapped in `factor()` so that `MARSS` recognizes what it is. You can use either numeric or character vectors: `c(1,1,1,1,1)` is the same as `c("PS","PS","PS","PS","PS")`.

Next we specify that the $\mathbf{R}$ variance-covariance matrix only has terms on the diagonal (the variances) with the off-diagonal terms (the covariances) equal to zero:

```
R.model = "diagonal and unequal"
```

The 'and unequal' part specifies that the variances are allowed to be unique on the diagonal. If we wanted to force the observation variances to be equal at all regions, we would use `"diagonal and equal"`.

For the first analysis, we only need to set the model structure for $\mathbf{Z}$ and $\mathbf{R}$. Since there is only one population, there is only one $\mathbf{u}$ and $\mathbf{Q}$ (they are scalars), so there are no constraints to set on them.

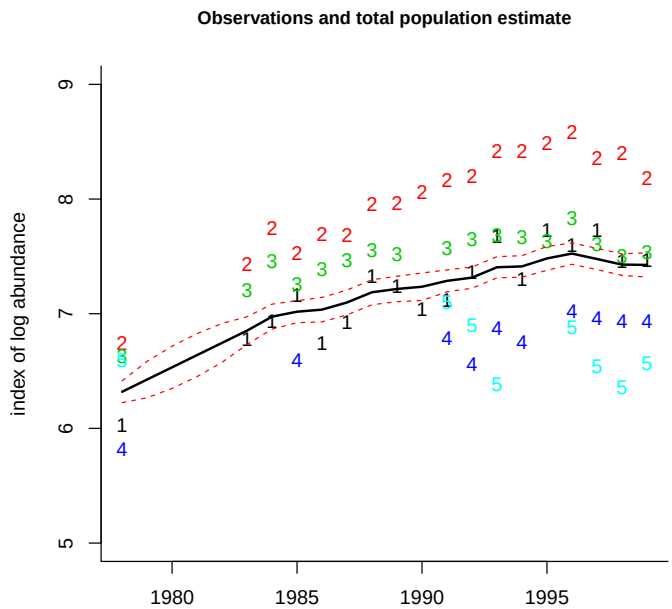

Fig. 10.2. Plot of the estimate of "log total harbor seals in Puget Sound" (minus the unknown bias for time series 1 against the data. The estimate of the total seal count has been scaled relative to the first time series. The 95% confidence intervals on the population estimates are the dashed lines. These are not the confidence intervals on the observations, and the observations (the numbers) will not fall between the confidence interval lines.

10.2.4 The `MARSS()` output

The output from `MARSS()`, here assigned the name `kem`, is a list of objects. To see all the objects in it, type:

```
names(kem1)
```

The maximum-likelihood estimates of "total harbor seal population" scaled to the first observation data series (Figure 10.2) are in `kem1$states`, and `kem1$states.se` are the standard errors on those estimates. To get 95% confidence intervals, use `kem1$states +/- 1.96*kem1$states.se`. Figure 10.2 shows a plot of `kem1$states` with its 95% confidence intervals over the data. Because `kem1$states` has been scaled relative to the first time series, it is on top of that time series. One of the biases, the as, cannot be estimated and arbitrarily our algorithm choses $a_1 = 0$, so the population estimate is scaled to the first observation time series.

The estimated parameters are output with the function `coef`: `coef(kem1)`. To get the estimate just for `U`, which is the estimated long-term population growth rate, use `coef(kem1)$U`. Multiply by 100 to get the percent increase per year. The estimated process variance is given by `coef(kem2)$Q`.

The log-likelihood of the fitted model is in `kem1$logLik`. We estimated one initial x ($t = 1$), one process variance, one u, four a's, and five observation variances's. So $K = 12$ parameters. The AIC of this model is $-2 \times$ log-like $+ 2K$, which we can show by typing `kem1$AIC`.

Example 10.1 (Fit the single population model)

Analyze the harbor seal data using the single population model (Equations 10.2 and 10.3). The code for Example 10.1 shows you how to input data and send it to the function `MARSS()`*. As you run the examples, add the estimates to the table at the end of the chapter so you can compare estimates across the examples.*

Example 10.1 code

Type `RShowDoc("Case_study_2.R",package="MARSS")` *to open a file with all the example code.*

```
#Read in data
dat=t(harborSealWA) #Transpose since MARSS needs time ACROSS columns
years = dat[1,]
n = nrow(dat)-1
dat = dat[2:nrow(dat),]
legendnames = (unlist(dimnames(dat)[1]))
#estimate parameters
Z.model = factor(c(1,1,1,1,1))
R.model = "diagonal and unequal"
kem1 = MARSS(dat, model=list(Z=Z.model, R=R.model))
#make figure
matplot(years, t(dat),xlab="",ylab="index of log abundance",
  pch=c("1","2","3","4","5"),ylim=c(5,9),bty="L")
lines(years,kem1$states-1.96*kem1$states.se,type="l",
  lwd=1,lty=2,col="red")
lines(years,kem1$states+1.96*kem1$states.se,type="l",
  lwd=1,lty=2,col="red")
lines(years,kem1$states,type="l",lwd=2)
title("Observations and total population estimate",cex.main=.9)
coef(kem1, type="vector")  #show the estimated parameter elements as a vector
#show estimated elements for each parameter matrix as a list
coef(kem1)
kem1$logLik   #show the log-likelihood
kem1$AIC  #show the AIC
```

10.3 Different observation error structures

The output from `MARSS()` shows the estimated elements in the **R** matrix. If we want to see the full **R** matrix, we use `coef(..., type="matrix")$R`—`coef` is the generic R function for extracting estimated parameters from model objects. Here is the estimated **R** matrix for our first model:

```
coef(kem1,type="matrix")$R
```

```
           [,1]       [,2]       [,3]       [,4]      [,5]
[1,] 0.03229417 0.00000000 0.00000000 0.00000000 0.0000000
[2,] 0.00000000 0.03527748 0.00000000 0.00000000 0.0000000
```

```
[3,] 0.00000000 0.00000000 0.01352073 0.00000000 0.0000000
[4,] 0.00000000 0.00000000 0.00000000 0.01082157 0.0000000
[5,] 0.00000000 0.00000000 0.00000000 0.00000000 0.1960897
```

Notice that the variances along the diagonal are all different—we estimated five unique observation variances. We might be able to improve the fit (relative to the number of estimated parameters) by assuming that the observation variance is equal across regions but the errors are independent. This means we estimate one observation variance instead of five. This is a fairly standard assumption for data that come from the uniform survey methodology[4].

To impose this model, we set the **R** model to

```
R.model="diagonal and equal"
```

This tells `MARSS` that all the r's along the diagonal in **R** are the same. To fit this model to the data, call `MARSS()` as:

```
Z.model = factor(c(1,1,1,1,1))
R.model = "diagonal and equal"
kem2 = MARSS(dat, model=list(Z=Z.model, R=R.model))
```

We estimated one initial x, one process variance, one u, four a's, and one observation variance. So $K = 8$ parameters. The AIC for this new model compared to the old model with five observation variances is:

```
c(kem1$AIC,kem2$AIC)
```

```
[1] -9.323982  8.813447
```

A smaller AIC means a better model. The difference between the one observation variance versus the unique observation variances is >10, suggesting that the unique observation variances model is better.

One of the key diagnostics when you are comparing fits from multiple models is whether the model is flexible enough to fit the data. This can be checked by looking for temporal trends in the the residuals between the estimated population states (e.g. `kem2$states`) and the data. In Figure 10.3, the residuals for the second analysis are shown. Ideally, these residuals should not have a temporal trend. They should look cloud-like. The fact that the residuals have a strong temporal trend is an indication that our one population model is too restrictive for the data[5].

[4] By the way, this is not a good assumption for these data since the number haul-outs in each region varies and the regional counts are the sums across all haul-outs in a region. We will see that this is a poor assumption when we look at the AIC values.

[5] When comparing models via AIC, it is important that you only compare models that are flexible enough to fit the data. Fortunately if you neglect to do this, the inadequate models will usually have very high AICs and fall out of the mix anyhow.

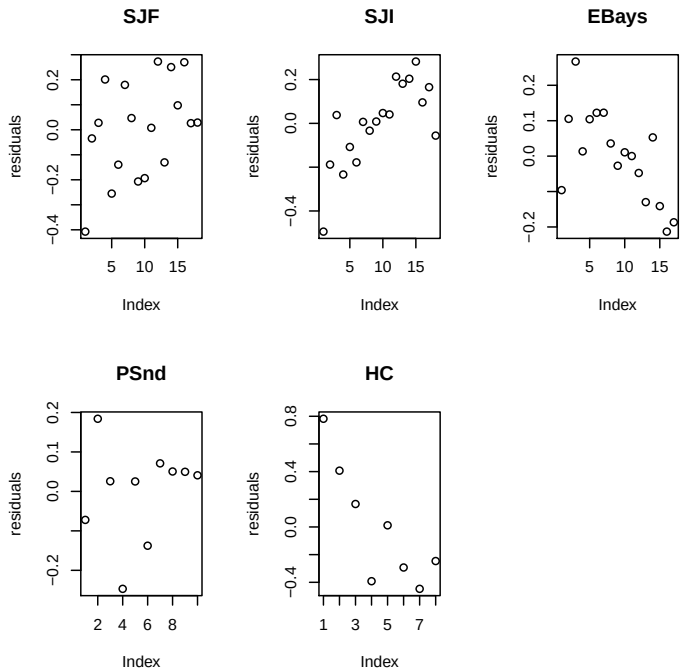

Fig. 10.3. Residuals for the model with a single population. The plots of the residuals should not have trends with time, but they do... This is an indication that the single population model is inconsistent with the data. The code to make this plot is given in the script file for this case study.

Example 10.2 (Fit a model with shared observation variances)

Analyze the data using the same population model as in Example 10.1, but constrain the **R** *matrix so that all five census regions have the same observation variance. The Example 10.2 code shows you how to do this. It also shows you how to make the diagnostics figure (Figure 10.3).*

Example 10.2 code

Type `RShowDoc("Case_study_2.R",package="MARSS")` *to open a file with all the example code.*

```
#fit model
Z.model = factor(c(1,1,1,1,1))
R.model = "diagonal and equal"
kem2 = MARSS(dat, model=list(Z=Z.model, R=R.model))
coef(kem2) #the estimated parameter elements
kem2$logLik #log likelihood
c(kem1$AIC,kem2$AIC)  #AICs
#plot residuals
plotdat = t(dat)
matrix.of.biases = matrix(coef(kem2, type="matrix")$A,
  nrow=nrow(plotdat),ncol=ncol(plotdat),byrow=T)
xs = matrix(kem2$states,
  nrow=dim(plotdat)[1],ncol=dim(plotdat)[2],byrow=F)
resids = plotdat-matrix.of.biases-xs
par(mfrow=c(2,3))
for(i in 1:n){
  plot(resids[!is.na(resids[,i]),i],ylab="residuals")
  title(legendnames[i])
}
par(mfrow=c(1,1))
```

10.4 Two subpopulations, north and south

For the third analysis, we will change our assumption about the structure of the population. We will assume that there are two subpopulations, north and south, and that regions 1 and 2 (Strait of Juan de Fuca and San Juan Islands) fall in the north subpopulation and regions 3, 4 and 5 fall in the south subpopulation. For this analysis, we will assume that these two subpopulations share their growth parameter, u, and process variance, q, since they share a similar environment and prey base. However we postulate that because of fidelity to natal rookeries for breeding, animals do not move much year-to-year between the north and south and the two subpopulations are independent.

We need to write down the state-space model to reflect this population structure. There are two subpopulations, x_n and x_s, and they have the same growth rate u:

$$\begin{bmatrix} x_n \\ x_s \end{bmatrix}_t = \begin{bmatrix} x_n \\ x_s \end{bmatrix}_{t-1} + \begin{bmatrix} u \\ u \end{bmatrix} + \begin{bmatrix} w_n \\ w_s \end{bmatrix}_t \tag{10.6}$$

We specify that they are independent by specifying that their year-to-year population fluctuations (their process errors) come from a multivariate normal with no covariance:

$$\begin{bmatrix} w_n \\ w_s \end{bmatrix}_t \sim MVN\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} q & 0 \\ 0 & q \end{bmatrix}\right) \tag{10.7}$$

For the observation process, we use the **Z** matrix to associate the regions with their respective x_n and x_s values:

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix}_t = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_n \\ x_s \end{bmatrix}_t + \begin{bmatrix} 0 \\ a_2 \\ 0 \\ a_4 \\ a_5 \end{bmatrix} + \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{bmatrix}_t \tag{10.8}$$

10.4.1 Specifying the `MARSS()` arguments

We need to change the **Z** model to specify that there are two subpopulations (north and south), and that regions 1 and 2 are in the north subpopulation and regions 3,4 and 5 are in the south subpopulation. There are a few ways, we can specify this **Z** matrix for `MARSS()`:

```
Z.model = matrix(c(1,1,0,0,0,0,0,1,1,1),5,2)
Z.model = factor(c(1,1,2,2,2))
Z.model = factor(c("N","N","S","S","S"))
```

Which you choose is a matter of preference as they all specify the same form for **Z**.

We also want to specify that the u's are the same for each subpopulation and that **Q** is diagonal with equal q's. To do this, we set

```
U.model = "equal"
Q.model = "diagonal and equal"
```

This says that there is one u and one q parameter and both subpopulations share it (if we wanted the u's to be different, we would use `U.model="unequal"` or leave off the **u** model since the default behavior is `U.model="unequal"`).

Now we specify the new model structures and fit this model to the data:

```
Z.model = factor(c(1,1,2,2,2))
U.model = "equal"
Q.model = "diagonal and equal"
R.model = "diagonal and equal"
kem3 = MARSS(dat, model=list(Z=Z.model,
   R=R.model, U=U.model, Q=Q.model))
```

```
Success! abstol and log-log tests passed at 32 iterations.
Alert: conv.test.slope.tol is 0.5.
Test with smaller values (<0.1) to ensure convergence.

MARSS fit is
Estimation method: kem
Convergence test: conv.test.slope.tol = 0.5, abstol = 0.001
Estimation converged in 32 iterations.
Log-likelihood: 11.65243
AIC: -7.304859   AICc: -4.982278

       Estimate
A.SJI   0.79883
A.PSnd -0.77932
A.HC   -0.84626
R.diag  0.02929
U.1     0.05029
Q.diag  0.00762
x0.1    6.06937
x0.2    6.85729

Standard errors have not been calculated.
Use MARSSparamCIs to compute CIs and bias estimates.
```

Figure 10.4 shows the residuals for the two subpopulations case. The residuals look better (more cloud-like) but the Hood Canal residuals are still temporally correlated.

Example 10.3 (Fit a model with north and south subpopulations)

Analyze the data using a model with two subpopulations, northern and southern. Assume that the subpopulation are independent (diagonal $\mathbf{Q}$*), however let each subpopulation share the same population parameters,* $\boldsymbol{u}$ *and* q*. The Example 10.3 code shows how to set the* `MARSS()` *arguments for this case.*

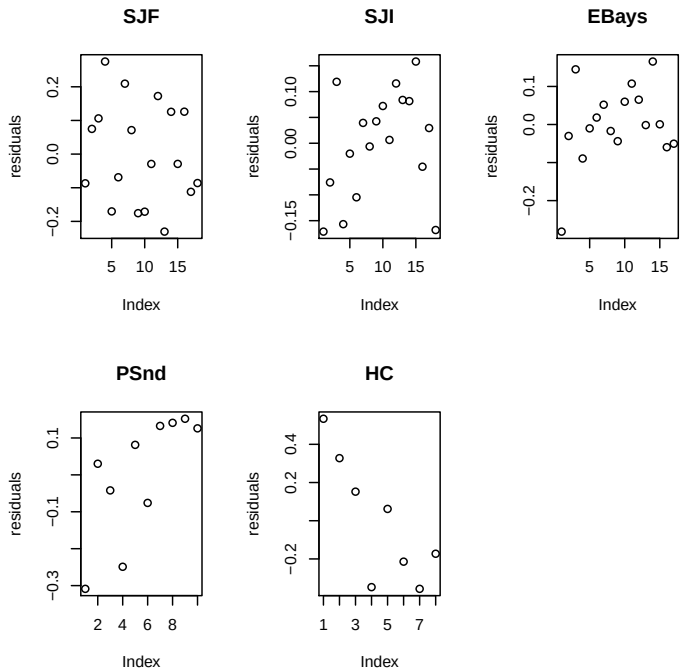

Fig. 10.4. The residuals for the analysis with a north and south subpopulation. The plots of the residuals should not have trends with time. Compare with the residuals for the analysis with one subpopulation.

Example 10.3 code

Type `RShowDoc("Case_study_2.R",package="MARSS")` *to open a file with all the example code.*

```
#fit model
Z.model = factor(c(1,1,2,2,2))
U.model = "equal"
Q.model = "diagonal and equal"
R.model = "diagonal and equal"
kem3 = MARSS(dat, model=list(Z=Z.model,
  R=R.model, U=U.model, Q=Q.model))
#plot residuals
plotdat = t(dat)
matrix.of.biases = matrix(coef(kem3,type="matrix")$A,
  nrow=nrow(plotdat),ncol=ncol(plotdat),byrow=T)
par(mfrow=c(2,3))
for(i in 1:n){
  j=c(1,1,2,2,2)
  xs = kem3$states[j[i],]
  resids = plotdat[,i]-matrix.of.biases[,i]-xs
  plot(resids[!is.na(resids)],ylab="residuals")
  title(legendnames[i])
}
par(mfrow=c(1,1))
```

10.5 Other population structures

Now work through a number of different structures and fill out the table at the back of this case study. At the end you will see how your estimation of the mean population growth rate varies under different assumptions about the population and the data.

Example 10.4 (Five subpopulations)

Analyze the data using a model with five subpopulations, where each of the five census regions is sampling one of the subpopulations. Assume that the subpopulation are independent (diagonal $\mathbf{Q}$*), however let each subpopulation share the same population parameters,* u *and* q*. The Example 10.4 code shows how to set the* `MARSS()` *arguments for this case. You can use* `R.model="diagonal and equal"` *to make all the observation variances equal.*

Example 10.4 code

Type `RShowDoc("Case_study_2.R",package="MARSS")` *to open a file with all the example code.*

```
Z.model=factor(c(1,2,3,4,5))
U.model="equal"
Q.model="diagonal and equal"
R.model="diagonal and unequal"
kem=MARSS(dat, model=list(Z=Z.model,
  U=U.model, Q=Q.model, R=R.model) )
```

Example 10.5 (Two subpopulations with different population parameters)

Analyze the data using a model that assumes that the Strait of Juan de Fuca and San Juan Islands census regions represent a northern Puget Sound subpopulation, while the other three regions represent a southern Puget Sound subpopulation. This time assume that each population trajectory (north and south) has different u *and* q *parameters:* u_n,u_s *and* q_n,q_s*. Also assume that each of the five census regions has a different observation variance. Try to write your own code. If you get stuck (or want to check your work, you can open a script file with all the Case Study 2 examples by typing* `RShowDoc("Case_study_2.R",package="MARSS")` *at the R command line.*

In math form, this model is:

$$\begin{bmatrix} x_n \\ x_s \end{bmatrix}_t = \begin{bmatrix} x_n \\ x_s \end{bmatrix}_{t-1} + \begin{bmatrix} u_n \\ u_s \end{bmatrix} + \begin{bmatrix} w_n \\ w_s \end{bmatrix}_t, \begin{bmatrix} w_n \\ w_s \end{bmatrix}_t \sim \text{MVN}\left(0, \begin{bmatrix} q_n & 0 \\ 0 & q_s \end{bmatrix}\right) \tag{10.9}$$

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix}_t = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_n \\ x_s \end{bmatrix}_t + \begin{bmatrix} 0 \\ a_2 \\ 0 \\ a_4 \\ a_5 \end{bmatrix} + \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{bmatrix}_t \tag{10.10}$$

Example 10.6 (Hood Canal covaries with the other regions)

Analyze the data using a model with two subpopulations with the divisions being Hood Canal versus everywhere else. In math form, this model is:

$$\begin{bmatrix} x_p \\ x_h \end{bmatrix}_t = \begin{bmatrix} x_p \\ x_h \end{bmatrix}_{t-1} + \begin{bmatrix} u_p \\ u_h \end{bmatrix} + \begin{bmatrix} w_p \\ w_h \end{bmatrix}_t, \begin{bmatrix} w_p \\ w_h \end{bmatrix}_t \sim \text{MVN}\left(0, \begin{bmatrix} q & c \\ c & q \end{bmatrix}\right) \tag{10.11}$$

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix}_t = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_p \\ x_h \end{bmatrix}_t + \begin{bmatrix} 0 \\ a_2 \\ a_3 \\ a_4 \\ 0 \end{bmatrix} + \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{bmatrix}_t \tag{10.12}$$

To specify that **Q** *has one value on the diagonal (one variance) and one value on the off-diagonal (covariance) you can specify* `Q.model` *two ways:*

```
Q.model = "equalvarcov"
Q.model = matrix(c("q","c","c","q"),2,2)
```

Example 10.7 (Three subpopulations with shared parameter values)

Analyze the data using a model with three subpopulations as follows: north (regions 1 and 2), south (regions 3 and 4), Hood Canal (region 5). You can specify that some subpopulations share parameters while others do not. First, let's specify that each population is affected by independent environmental variability, but that the variance of that variability is the same for the two interior populations:

```
Q.model=matrix(list(0),3,3)
diag(Q.model)=c("coastal","interior","interior")
print(Q.model)
```

Notice that **Q** *is a diagonal matrix (independent year-to-year environmental variability) but the variance of two of the populations is the same. Notice too that the off-diagonal terms are numeric; they do not have quotes. We specified* **Q** *using a matrix of class list, so that we could have numeric values (fixed) and character values (estimated parameters).*

In a similar way, we specify that the observation errors are independent but that estimates from a plane do not have the same variance as those from a boat:

```
R.model=matrix(list(0),5,5)
diag(R.model)=c("boat","boat","plane","plane","plane")
```

For the long-term trends, we specify that x_1 *and* x_2 *share a long-term trend ("puget sound") while* x_3 *is allowed to have a separate trend ("hood canal").*

```
U.model=matrix(c("puget sound","puget sound","hood canal"),3,1)
```

10.6 Discussion

There are a number of corners that we cut in order to have case study code that runs quickly:

- We ran the code starting from one initial condition. For a real analysis, you should start from a large number of random initial conditions and use the one that gives the highest likelihood. Since the EM algorithm is a "hill-climbing" algorithm, this ensures that it does not get stuck on a local maxima. `MARSS()` will do this for you if you pass it the argument `control=list(MCInit=TRUE)`. This will use a Monte Carlo routine to try many different initial conditions. See the help file on `MARSS()` for more information (by typing `?MARSS` at the R prompt).
- We assume independent observation and process errors. Depending on your system, observation errors may be driven by large-scale environmental factors (temperature, tides, prey locations) that would cause your observation errors to covary across regions. If your observation errors strongly covary between regions and you treat them as independent, this could be bad for your analysis. Unfortunately, separating covariance across observation

versus process errors will require much data (to have any power). In practice, the first step is to think hard about what drives sightability for your species and what are the relative levels of process and observation variance. You may be able to subsample your data in a way that will make the observation errors more independent.

- The `MARSS()` argument `control` specifies the options for the EM algorithm. We left the default tolerance for the convergence test. You would want to set this lower for a real analysis. You will need to up the `maxit` argument correspondingly.
- We used the large-sample approximation for AIC instead of a bootstrap AIC that is designed to correct for small sample size in state-space models. The bootstrap metric, AICb, takes a long time to run. Use the call `MARSSaic(kem, output=c("AICbp"))` to compute AICb. We could have shown AICc, which is the small-sample size corrector for non-state-space models. Type `kem$AICc` to get that.

Finally, in a real (maximum-likelihood) analysis, one needs to be careful not to dredge the data. The temptation is to look at the data and pick a population structure that will fit that data. This can lead to including models in your analysis that have no biological basis. In practice, we spend a lot of time discussing the population structure with biologists working on the species and review all the biological data that might tell us what are reasonable structures. From that, a set of model structures to use are selected. Other times, a particular model structure needs to be used because the population structure is not in question rather it is a matter of using that pre-specified structure and using all the data to get parameter estimates for forecasting.

Results table

Ex.		pop. growth rate `U`	process variance `Q`	K `kem$num. params`	log-like `kem$ logLik`	AIC `kem$AIC`
1	one population different obs. vars uncorrelated					
2	one population identical obs vars uncorrelated					
3	N+S subpops identical obs vars uncorrelated;					
4	5 subpops unique obs vars u's + q's identical					
5	N+S subpops unique obs vars u's + q's identical					
6	PS + HC subpops unique obs vars u's + q's unique					
7	N + S + HC subpops unique obs vars u's + q's unique					

For AIC, lower is better and only the relative differences matter. A difference of 10 between two AICs means substantially more support for the model with lower AIC. A difference of 30 or 40 between two AICs is very large.

Questions

1. Do different assumptions about whether the observation error variances are all identical versus different affect your estimate of the long-term population growth rate (u)? You may want to rerun examples 3-7 with the `R.model` changed. `R.model="diagonal and unequal"` means measurement variances all different versus `"diagonal and equal"`.
2. Do assumptions about the underlying structure of the population affect your estimates of u? Structure here means number of subpopulations and which areas are in which subpopulation.

3. The confidence intervals for the first two analyses are very tight because the estimated process variance, $\mathbf{Q}$, was very small. Why do you think process variance (q) was forced to be so small? [Hint: We are forcing there to be one and only one true population trajectory and all the observation time series have to fit that one time series. Look at the AICs too.]

11

Case Study 3: Identifying spatial population structure and covariance

11.1 Harbor seals on the U.S. west coast

In this case study, we use time series of observations from nine sites along the west coast to examine large-scale spatial structure in harbor seals (Jeffries et al., 2003). Harbor seals are distributed along the west coast of the U.S. from California to Washington. The populations in Oregon and Washington have been surveyed for over 25 years at a number of haul-out sites (Figure 11.1). In general, these populations have been increasing steadily since the 1972 Marine Mammal Protection Act. It remains unknown whether they are at carrying capacity.

For management purposes, two stocks are recognized; the coastal stock consists of four sites (Northern/Southern Oregon, Coastal Estuaries, Olympic Peninsula), and the inland WA stock consists of the remaining five sites (Figure 11.1). Subtle differences exist in the demographics across sites (e.g. pupping dates), however mtDNA analyses and tagging studies have suggested that these sites may be structured on a much larger scale. Harbor seals are known for strong site fidelity, but at the same time travel large distances to forage.

Our goal for this case study is to address the following questions about spatial structure: 1) Does population abundance data support the existing management boundaries, or are there alternative groupings that receive more support? and 2) Does the Hood Canal site represent a distinct subpopulation? To address these questions, we will mathematically formulate different hypotheses about population structure via different MARSS models; each model represents a different population structure. We will then compare the data support for different models using model selection criteria, specifically AIC.

The mathematical form of the model used in this case study is

$$\begin{aligned} \mathbf{x}_t &= \mathbf{x}_{t-1} + \mathbf{u} + \mathbf{w}_t \text{ where } \mathbf{w}_t \sim \text{MVN}(0, \mathbf{Q}) \\ \mathbf{y}_t &= \mathbf{Z}\mathbf{x}_t + \mathbf{a} + \mathbf{v}_t \text{ where } \mathbf{v}_t \sim \text{MVN}(0, \mathbf{R}) \\ \mathbf{x}_0 &\sim \text{MVN}(\pi, \Lambda) \end{aligned} \tag{11.1}$$

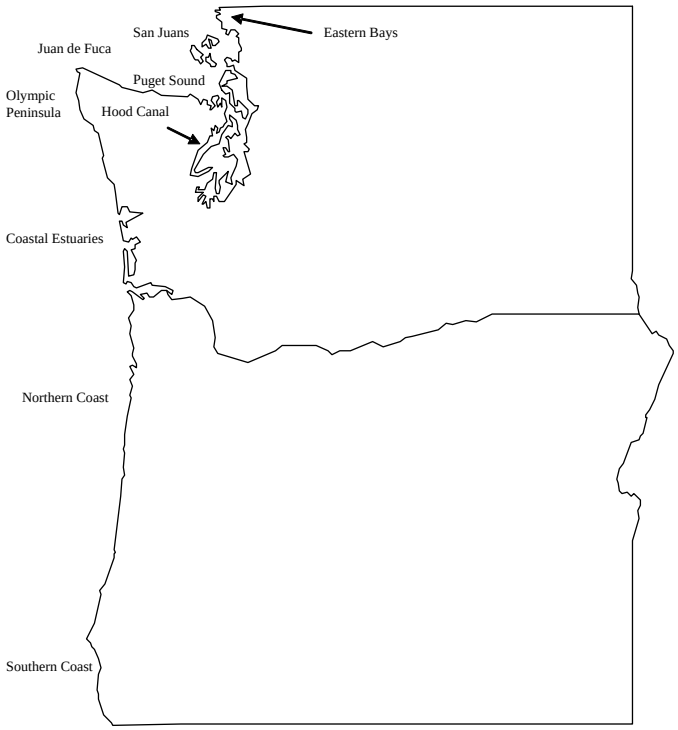

Fig. 11.1. Map of spatial distribution of 9 harbor seal sites in Washington and Oregon.

Your goal is to specify the form of **Z**, **u**, **Q**, **Z** and **R** for each of the hypotheses. The form of **a** is "scaling" in all cases and **B** is "identity". **B** is in front of **x** but is left off above since it is the identity matrix[1].

Type `RShowDoc("Case_study_3.R",package="MARSS")` to open a file with the R code to get you started on the analyses in this chapter.

11.2 How many distinct subpopulations?

We will analyze the support for five hypotheses about the population structure. These do not represent all possible structures but instead represent those that are considered most biologically plausible given the geography and the behavior of harbor seals.

[1] a diagonal matrix with 1s on the diagonal. The identity matrix (denoted **I**) is like "1"; $\mathbf{Ix} = \mathbf{x}$.

Hypothesis 1 Sites are grouped by stock ($m = 2$), unique process variances
Hypothesis 2 Sites are grouped by stock ($m = 2$), same process variance
Hypothesis 3 Sites are grouped by state ($m = 2$), unique process variances
Hypothesis 4 Sites are grouped by state ($m = 2$), same process variance
Hypothesis 5 All sites are part of the same panmictic population ($m = 1$)

Aerial survey methodology has been relatively constant across time and space, and we will assume that all sites have identical and independent observation error variance.

11.2.1 Specify the Z matrices

Write down the **Z** matrices for the hypotheses. Hint: Hypothesis 1 and 2 have the same **Z** matrix and Hypothesis 3 and 4 have the same **Z** matrix .

	H 1 and 2 **Z** subpop 1	H 1 and 2 **Z** subpop 2	H 3 and 4 **Z** subpop 1	H 3 and 4 **Z** subpop 2	H 5 **Z** subpop 1
`Coastal Estuaries`					
`Olympic Peninsula`					
`Str. Juan de Fuca`					
`San Juan Islands`					
`Eastern Bays`					
`Puget Sound`					
`Hood Canal`					
`OR North Coast`					
`OR South Coast`					

Next you need to specify the `model` argument so that `MARSS` knows the structure of your **Z**'s. Each **Z** is a matrix with 9 rows and different numbers of columns. Each row has one "1" in it while the rest of the elements are 0. You need to create this matrix and pass that in as the `Z` element for the model list. For example if your **Z** matrix looks like so:

$$\begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 4 \\ 1 & 0 \end{bmatrix} \tag{11.2}$$

You can specify this in R using

```
Z=matrix(0,5,2)
Z[c(1,2,5),1]=1  #which elements in col 1 are 1
Z[c(3,4),2]=1  #which elements in col 2 are 1
```

Write R code for each of the **Z** matrices for the hypotheses:

- Hypothesis 1 and 2: `Z.model=`
- Hypothesis 3 and 4: `Z.model=`
- Hypothesis 5: `Z.model=`

MARSS has a shortcut for making this kind of **Z** matrix, which is called a design matrix, using `factor`. To make the 5 design matrix shown above, we could write `factor(c(1,1,2,2,1))`. Each element is for one of the rows of **Z** and indicates which column the "1" appears in.

11.2.2 Specify the u and Q structures

For this case study, we will assume that subpopulations share the same growth rate. What should `U.model` be for each hypothesis? To specify shared **u** parameters, `U.model` is character matrix where shared elements in **u** have the same character name. Written in R it takes the form `matrix(c(#,#,...),m,1)`.

- Hypothesis 1-4: `U.model=`
- Hypothesis 5: `U.model=`

What about `Q.model`? To specify a diagonal **Q** matrix with shared values along the diagonal, we will need to have a matrix with numeric values (0) on the off-diagonals and character values (names of parameters to estimate) on the diagonal. We do this with a matrix of class list.

```
Q.model=matrix(list(0),m,1)
diag(Q.model)=c("q1","q1","q2","q2","...")
```

Notice that first, we set up `Q.model` as a list matrix with numeric 0s everywhere and then set the diagonals to a vector of character strings. Character strings that are the same are shared (identical) values.

Look at each hypothesis (above) and write down R code for the corresponding `Q.model`.

- Hypothesis 1: `Q.model=`
- Hypothesis 2: `Q.model=`
- Hypothesis 3: `Q.model=`
- Hypothesis 4: `Q.model=`
- Hypothesis 5: `Q.model=`

Lastly, specify `R.model`. As we mentioned above, we will assume that the observation errors are independent and the observation variance is the same across sites.

- Hypothesis 1-5: `R.model=`

11.2.3 Fit models and summarize results

Fit each model for each hypothesis to the seal data (look at the script `Case_Study_3.R` for the code to load the data). Each call to `MARSS()` will look like

```
kem = MARSS(sealData, model=list(Z = Z.model,
  Q = Q.model, R = R.model, U = U.model))
```

Fill in the following table, by fitting the five state-space models—for the five hypotheses—to the harbor seal data (using `MARSS()`). Use the `Case_Study_3.R` script so you do not have to type in all the commands.

Table 11.1. Table to fill out for the five hypotheses from the first analysis (Section 11.2). The code of the form `$foo$bar` shows you what to type at the command line to output each parameter or metric. Remember to add the name of the model fit, e.g. `kem$foo$bar` where `kem` is the name you gave to the model fit.

H	pop. growth rate `U`	proc. variance `Q`	obs. variance `R`	K `$num. params`	log-like `$logLik`	AIC `$AIC`	AICc `$AICc`
1							
2							
3							
4							
5							

11.2.4 Interpret results for question 1

What do these results indicate about the process error grouping and spatial grouping? A lower AIC means a more parsimonious model (highest likelihood given the number of parameters). A difference of 10 between AICs is large, and

means the model with the higher AIC is unsupported relative to the model with lower AIC.

Extra analysis (if you have time): Do your results change if you assume that observation errors are independent but have unique variances? The nine sites have different numbers of haul-outs and so the observation variances might be different. Repeat the analysis with unique observation variances for each site (this means changing `R.model`). You can also try the analysis with temporally co-varying subpopulations (good and bad years correlated) by setting `Q.model="unconstrained"` or `Q.model="equalvarcov"`.

11.3 Is Hood Canal separate?

The Hood Canal site may represent a distinct population, and has recently experienced a number of catastrophic events (hypoxic events, possibly leading to reduced prey availability, and several killer whale predation events, removing up to 50% of animals per occurrence). Build four models, assuming that each site (other than Hood Canal) is assigned to its current management stock, but Hood Canal is a different subpopulation ($m = 3$). Again, assume that observation error variance is identical and independent across sites.

Hypothesis 1 Subpopulations have the same process variance and growth rate
Hypothesis 2 Each subpopulation has a unique process variance and growth rate
Hypothesis 3 Hood Canal has the same process variance but different growth rate
Hypothesis 4 Hood Canal has unique process variance and unique growth rate

11.3.1 Specify the Z matrix and `Z.model`

The $\mathbf{Z}$ matrix for each hypothesis is the same. The coastal subpopulation consists of four sites (Northern/Southern Oregon, Coastal Estuaries, Olympic Peninsula), the Hood Canal subpopulation is the Hood Canal site, and the inland WA subpopulation consists of the remaining four sites. Thus $m = 3$ and $\mathbf{Z}$ is a 9×3 matrix:

$$
\begin{array}{r}
\\
\text{Coastal Estuaries} \\
\text{Olympic Peninsula} \\
\text{Str. Juan de Fuca} \\
\text{San Juan Islands} \\
\text{Eastern Bays} \\
\text{Puget Sound} \\
\text{Hood Canal} \\
\text{OR North Coast} \\
\text{OR South Coast}
\end{array}
\begin{array}{c}
\begin{array}{ccc} \text{subpop} & \text{subpop} & \text{subpop} \\ 1 & 2 & 3 \end{array} \\
\begin{bmatrix} \quad & \quad & \quad \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \end{bmatrix}
\end{array}
$$

Then write down R code to make the **Z** matrix.

11.3.2 Specify which parameters are shared across which subpopulations

`U.groups` specifies which u are shared across subpopulations. Look at the hypothesis descriptions above which will specify whether subpopulations share their population growth rate or have unique population growth rates.

- Hypothesis 1: `U.model=`
- Hypothesis 2: `U.model=`
- Hypothesis 3: `U.model=`
- Hypothesis 4: `U.model=`

`U.model` will be a $m \times 1$ matrix of character strings; `matrix(c("c1","c2",".....")),m,1)` Once you have more than two subpopulations, it can get hard to keep straight which `U.model` goes to which subpopulation. It is best to sketch your **Z** matrix (which tells you which site in the rows corresponds to which subpopulation in the columns). Then remember that the elements of `U.model` correspond one-to-one with the columns of **Z**:

`U.model=matrix(c(col 1 Z, col 2 Z, col 3 Z, ..),m,1)`.

Specify `Q.groups` showing which subpopulations share their process variance parameter.

- Hypothesis 1: `Q.model=`
- Hypothesis 2: `Q.model=`
- Hypothesis 3: `Q.model=`
- Hypothesis 4: `Q.model=`

`Q.model` will be specified using the matrix of class list again. For example,

```
Q.model=matrix(list(0),3,3)
diag(Q.model)=c("q1","q2","q3")
```

`R.model` is set so that the observation variances are the same for each site.

11.3.3 Fit the models and summarize results

Fit each model for each hypothesis to the seal data. Each call to `MARSS()` will look like

```
kem = MARSS(sealData, model=list(Z = Z.model,
  Q = Q.model, R = R.model, U = U.model))
```

Table 11.2. Table to fill out for the four hypotheses from the second analysis (Section 11.3). The code of the form `$foo$bar` shows you what to type at the command line to output each parameter or metric. Remember to add the name of the model fit, e.g. `kem$foo$bar` where `kem` is the name you gave to the model fit.

H	pop. growth rate `U`	proc. variance `Q`	obs. variance `R`	K `$num. params`	log-like `$logLik`	AIC `$AIC`	AICc `$AICc`
1							
2							
3							
4							

11.3.4 Interpret results for question 2

How do the residuals for the Hood Canal site compare from these models relative to the best model from question 1? Hint: If you have the vector of estimated population states (`Xpred = t(kem$states)`) and the data (`Xobs = sealData`), the residuals for site i can be plotted in R as:

```
Xpred = t(kem$states)
Xobs = sealData
plot(Xpred[, Z.model[i]] - Xobs[,i],
      ylab="Predicted-Observed Data")
```

In R , if you have a matrix `Y[1:numYrs, 1:n]`, you can extract column j by writing `Yj = Y[,j]`.

Relative to the previous models from question 1, do these scenarios have better or worse AIC scores (smaller AIC is better)? If you were to provide advice to managers, would you recommend that the Hood Canal population is a source or sink? What implications does this have for population persistence?

Code for Case Study 3

Type `RShowDoc("Case_study_3.R",package="MARSS")` to open a file in R with all the example code.

12

Case Study 4: Dynamic factor analysis (DFA)

12.1 Overview of dynamic factor analysis

In this case study, we use MARSS to do dynamic factor analysis (DFA), which allows us to look for a set of common underlying trends among a relatively large set of time series (Harvey, 1989, sec. 8.5). This is conceptually different than what we have been doing in the previous case studies. Here we are trying to explain temporal variation in a set of n observed time series using linear combinations of a set of m hidden random walks, where $m << n$. Zuur et al. (2003) show a number of examples of DFA applied to fisheries catch data and densities of zoobenthos.

A DFA model is a type of MARSS model with the following structure:

$$\begin{aligned} \mathbf{x}_t &= \mathbf{x}_{t-1} + \mathbf{w}_t \text{ where } \mathbf{w}_t \sim \text{MVN}(0, \mathbf{Q}) \\ \mathbf{y}_t &= \mathbf{Z}\mathbf{x}_t + \mathbf{a} + \mathbf{v}_t \text{ where } \mathbf{v}_t \sim \text{MVN}(0, \mathbf{R}) \\ \mathbf{x}_0 &\sim \text{MVN}(\boldsymbol{\pi}, \boldsymbol{\Lambda}) \end{aligned} \tag{12.1}$$

The general idea is that the observations ($\mathbf{y}$) are modeled as a linear combination of hidden trends ($\mathbf{x}$) and factor loadings ($\mathbf{Z}$) plus some offsets ($\mathbf{a}$). The DFA model in Equation 12.1 and the standard MARSS model in Equation 1.1 are equivalent–we have simply set the matrix $\mathbf{B}$ equal to an $m \times m$ identity matrix (i.e., a diagonal matrix with 1's on the diagonal and 0's elsewhere) and the vector $\mathbf{u} = \mathbf{0}$.

12.1.1 Writing out a DFA model in MARSS form

Imagine a case where we had a data set with six observed time series ($n = 6$) and we want to fit a model with three hidden trends ($m = 3$). If we write out our DFA model in MARSS matrix form (ignoring the error structures and initial conditions for now), it would look like this:

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}_t = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}_{t-1} + \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}_t$$

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \end{bmatrix}_t = \begin{bmatrix} z_{11} & z_{12} & z_{13} \\ z_{21} & z_{22} & z_{23} \\ z_{31} & z_{32} & z_{33} \\ z_{41} & z_{42} & z_{43} \\ z_{51} & z_{52} & z_{53} \\ z_{61} & z_{62} & z_{63} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}_t + \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \end{bmatrix} + \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \end{bmatrix}_t . \tag{12.2}$$

The process errors of the hidden trends would be

$$\begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}_t \sim \text{MVN}\left(\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} q_{11} & q_{12} & q_{13} \\ q_{12} & q_{22} & q_{23} \\ q_{13} & q_{23} & q_{33} \end{bmatrix} \right), \tag{12.3}$$

and the observation errors would be

$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \end{bmatrix}_t \sim \text{MVN}\left(\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} r_{11} & r_{12} & r_{13} & r_{14} & r_{15} & r_{16} \\ r_{12} & r_{22} & r_{23} & r_{24} & r_{25} & r_{26} \\ r_{13} & r_{23} & r_{33} & r_{34} & r_{35} & r_{36} \\ r_{14} & r_{24} & r_{34} & r_{44} & r_{45} & r_{46} \\ r_{15} & r_{25} & r_{35} & r_{45} & r_{55} & r_{56} \\ r_{16} & r_{26} & r_{36} & r_{46} & r_{56} & r_{66} \end{bmatrix} \right). \tag{12.4}$$

12.1.2 Constraints to ensure identifiability

If $\mathbf{Z}$, $\mathbf{a}$, and $\mathbf{Q}$ in Equation 12.1 are not constrained, then the DFA model above is unidentifiable (Harvey, 1989, sec 4.4). Harvey (1989, sec. 8.5.1) suggests the following parameter constraints to make the model identifiable:

- in the first $m-1$ rows of $\mathbf{Z}$, the z-value in the j-th column and i-th row is set to zero if $j > i$;
- $\mathbf{a}$ is constrained so that the first m values are set to zero; and
- $\mathbf{Q}$ is set equal to the identity matrix ($\mathbf{I}_m$).

Zuur et al. (2003), however, note that with Harvey's second constraint, the EM algorithm is not particularly robust, and it takes a long time to converge. Zuur et al. show that the EM estimates are much better behaved if you instead constrain each of the time-series in $\mathbf{x}$ to have a mean of zero across $t = 1$ to T. To do so, replace the estimates of the hidden states, $\mathbf{x}_t^T$, coming out of the Kalman smoother with $\mathbf{x}_t^T - \bar{\mathbf{x}}$ for $t = 1$ to T ($NOTE$: $\bar{\mathbf{x}}$ is the mean of $\mathbf{x}_t$ across t, not m). With this approach, you estimate all of the $\mathbf{a}$ elements,

which represent the average level of $\mathbf{y}_t$ relative to $\mathbf{Z}(\mathbf{x}_t - \bar{\mathbf{x}})$. We will demean our data, however, and thus will set all elements of $\mathbf{a}$ to zero.

Using these constraints, the DFA model in Equation 12.2 becomes

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}_t = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}_{t-1} + \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}_t$$

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \end{bmatrix}_t = \begin{bmatrix} z_{11} & 0 & 0 \\ z_{21} & z_{22} & 0 \\ z_{31} & z_{32} & z_{33} \\ z_{41} & z_{42} & z_{43} \\ z_{51} & z_{52} & z_{53} \\ z_{61} & z_{62} & z_{63} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}_t + \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \end{bmatrix} + \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \end{bmatrix}_t . \tag{12.5}$$

The process errors of the hidden trends in Equation 12.3 would then become

$$\begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}_t \sim \text{MVN}\left(\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \right), \tag{12.6}$$

but the observation errors in Equation 12.4 would stay the same, such that

$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \end{bmatrix}_t \sim \text{MVN}\left(\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} r_{11} & r_{12} & r_{13} & r_{14} & r_{15} & r_{16} \\ r_{12} & r_{22} & r_{23} & r_{24} & r_{25} & r_{26} \\ r_{13} & r_{23} & r_{33} & r_{34} & r_{35} & r_{36} \\ r_{14} & r_{24} & r_{34} & r_{44} & r_{45} & r_{46} \\ r_{15} & r_{25} & r_{35} & r_{45} & r_{55} & r_{56} \\ r_{16} & r_{26} & r_{36} & r_{46} & r_{56} & r_{66} \end{bmatrix} \right). \tag{12.7}$$

To complete our model, we still need the final form for the initial conditions of the state. Following Zuur et al. (2003), we set the initial state vector ($\mathbf{x}_0$) to have zero mean and a diagonal variance-covariance matrix with large variances, such that

$$\mathbf{x}_0 \sim \text{MVN}\left(\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 5 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 5 \end{bmatrix} \right). \tag{12.8}$$

12.2 The data

For this case study, we will analyze some of the Lake Washington plankton data included in the MARSS package. This dataset includes 33 years of monthly counts for 13 plankton species along with data on water temperature,

total phosphorous (TP), and pH. First, we load the data and then extract a subset of columns corresponding to the phytoplankton species only. For the purpose of speeding up model fitting times and to limit our analysis to years with no missing covariate data, we will only examine 10 years of data (1980-1989).

```
# load the data
# This loads in three lakeWAplankton datasets.  We want lakeWAplanktonTrans
data(lakeWAplankton)
# this is a transformed dataset where the 0s are replaced with NAs and the data z-scored
# use only the data from 1980 onward
plankdat=lakeWAplanktonTrans
plankdat=plankdat[plankdat[,"Year"]>=1980 & plankdat[,"Year"]<1990,]
# create vector of phytoplankton group names
phytoplankton = c("Cryptomonus", "Diatoms", "Greens",
                  "Bluegreens", "Unicells", "Other.algae")
# get only the phytoplankton
dat.spp.1980 = plankdat[,phytoplankton]
```

Next, we transpose the data and calculate the number of time series and their length.

```
# transpose data so time goes across columns
dat.spp.1980 = t(dat.spp.1980)
# get number of time series
N.ts = dim(dat.spp.1980)[1]
# get length of time series
TT = dim(dat.spp.1980)[2]
```

It is normal in this type of analysis to standardize each time series by first subtracting its mean and then dividing by its standard deviation (i.e., create a z-score $\mathbf{y}_t^*$ with mean = 0 and SD = 1), such that

$$\mathbf{y}_t^* = \Sigma^{-1}(\mathbf{y}_t - \bar{\mathbf{y}}),$$

Σ is a diagonal matrix with the standard deviations of each time series along the diagonal, and $\bar{\mathbf{y}}$ is a vector of the means. In R, this can be done as follows

```
Sigma = sqrt(apply(dat.spp.1980, 1, var, na.rm=TRUE))
y.bar = apply(dat.spp.1980, 1, mean, na.rm=TRUE)
dat.z = (dat.spp.1980 - y.bar) * (1/Sigma)
rownames(dat.z) = rownames(dat.spp.1980)
```

Figure 12.1 shows time series of Lake Washington phytoplankton data following z-score transformation.

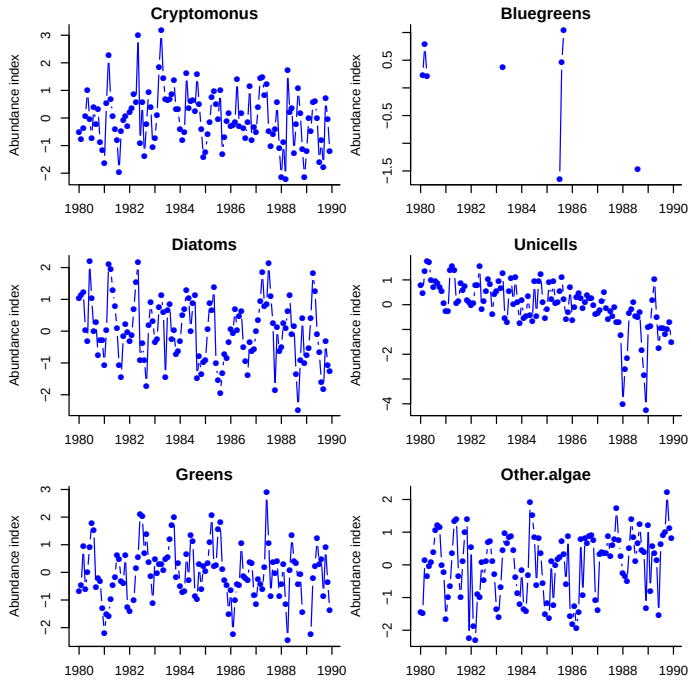

Fig. 12.1. Time series of Lake Washington phytoplankton data following z-score transformation.

12.3 Setting up the model in MARSS

As we have seen in other cases, setting up the model structure for MARSS requires that the parameter matrices have a one-to-one correspondence to the model as you would write it on paper (i.e., Equations 12.5 through 12.8). If a parameter matrix has a combination of fixed and estimated values, then you specify that using `matrix(list(), nrow, ncol)`. This is a matrix of class list and allows you to combine numeric and character values in a single matrix. MARSS recognizes the numeric values as fixed values and the character values as estimated values.

This is how we set up **Z** for MARSS:

```
Z.vals = list(
"z11",  0  ,  0  ,
"z21","z22",  0  ,
"z31","z32","z33",
"z41","z42","z43",
"z51","z52","z53",
```

```
"z61","z62","z63")
Z = matrix(Z.vals, nrow=N.ts, ncol=3, byrow=TRUE)
```

When specifying the list values, spacing and carriage returns were added to help show the correspondence with the **Z** matrix in Equation 12.3. If you print **Z** (at the `R` command line), you will see that it is a matrix with character values (the estimated elements) and numeric values (the fixed 0's).

```
print(Z)
```

```
     [,1]  [,2]  [,3]
[1,] "z11" 0     0
[2,] "z21" "z22" 0
[3,] "z31" "z32" "z33"
[4,] "z41" "z42" "z43"
[5,] "z51" "z52" "z53"
[6,] "z61" "z62" "z63"
```

Notice that the `0`'s do not have quotes around them. If they did, it would mean the `"0"` is a character value and would be interpreted as the name of a parameter to be estimated rather than a fixed numeric value.

The parameter vector **a** is set up similarly.

```
A.vals = list("a1","a2","a3","a4","a5","a6")
A = matrix(A.vals, nrow=N.ts, ncol=1)
```

However, there are no missing values in the Lake Washington plankton dataset and we have demeaned the data. Therefore we can set $\mathbf{a} = 0$.

The **Q** and **B** matrices are both set equal to the identity matrix using `diag()`.

```
Q = B = diag(1,3)
```

For our first analysis, we will assume that each time series of phytoplankton has a different observation variance, but that there is no covariance among time series. Thus, **R** should be a diagonal matrix that looks like:

$$\begin{bmatrix} r_{11} & 0 & 0 & 0 & 0 & 0 \\ 0 & r_{22} & 0 & 0 & 0 & 0 \\ 0 & 0 & r_{33} & 0 & 0 & 0 \\ 0 & 0 & 0 & r_{44} & 0 & 0 \\ 0 & 0 & 0 & 0 & r_{55} & 0 \\ 0 & 0 & 0 & 0 & 0 & r_{66} \end{bmatrix},$$

and each of the $r_{i,i}$ elements is a different parameter to be estimated. We can also specify this **R** structure using a list matrix as follows:

```
R.vals = list(
"r11",0,0,0,0,0,
0,"r22",0,0,0,0,
```

```
0,0,"r33",0,0,0,
0,0,0,"r44",0,0,
0,0,0,0,"r55",0,
0,0,0,0,0,"r66")
R = matrix(R.vals, nrow=N.ts, ncol=N.ts, byrow=TRUE)
```

You can print **R** at the R command line to see what it looks like:

```
print(R)
```

```
     [,1]  [,2]  [,3]  [,4]  [,5]  [,6]
[1,] "r11" 0     0     0     0     0
[2,] 0     "r22" 0     0     0     0
[3,] 0     0     "r33" 0     0     0
[4,] 0     0     0     "r44" 0     0
[5,] 0     0     0     0     "r55" 0
[6,] 0     0     0     0     0     "r66"
```

This form of variance-covariance matrix is commonly used, and therefore MARSS has a built-in shorthand for this structure. Alternatively, we could simply type:

```
R = "diagonal and unequal"
```

As mentioned in earlier chapters, there are other shorthand notations for many of the common parameter structures. Type ?MARSS at the R command line to see a list of the shorthand options for each parameter vector/matrix.

The parameter vectors π (termed x0 in MARSS) and **u** are each set to be a column vector of zeros. Either of the following can be used:

```
x0 = U = matrix(0, nrow=3, ncol=1)
x0 = U = "zero"
```

The Λ matrix (termed V0 in MARSS) is a diagonal matrix with 5's along the diagonal:

```
V0 = diag(5,3)
```

Finally, we make a list of the model parameters to pass to the MARSS() function and set the control list:

```
dfa.model = list(Z=Z, A="zero", R=R, B=B, U=U, Q=Q, x0=x0, V0=V0)
cntl.list = list(maxit=50)
```

For the examples in this chapter, we have set the maximum iterations to 50 to speed up model fitting. Note, however, that the parameter estimates will not have converged to their maximum likelihood values, which would likely take 100s, if not 1000+, iterations.

12.3.1 Fitting the model

We can now pass the DFA model list to MARSS() to estimate the **Z** matrix and underlying hidden states (**x**). The output is not shown because it is voluminous, but the model fits are plotted in Figure 12.2.

```
kemz.3 = MARSS(dat.z, model=dfa.model, control=cntl.list)
```

```
Warning! Reached maxit before parameters converged. Maxit was 50.
 neither abstol nor log-log convergence tests were passed.

MARSS fit is
Estimation method: kem
Convergence test: conv.test.slope.tol = 0.5, abstol = 0.001
WARNING: maxit reached at  50  iter before convergence.
 Neither abstol nor log-log convergence test were passed.
 The likelihood and params are not at the ML values.
 Try setting control$maxit higher.
Log-likelihood: -799.3133
AIC: 1640.627   AICc: 1642.214

                            Estimate
Z.z11                          0.398
Z.z21                          0.538
Z.z31                          0.257
Z.z41                          0.907
Z.z51                          0.538
Z.z61                          0.121
Z.z22                          0.687
Z.z32                         -0.182
Z.z42                          0.951
Z.z52                         -0.254
Z.z62                         -0.138
Z.z33                          0.278
Z.z43                          1.073
Z.z53                         -0.073
Z.z63                          0.420
R.(Cryptomonus,Cryptomonus)    0.688
R.(Diatoms,Diatoms)            0.103
R.(Greens,Greens)              0.717
R.(Bluegreens,Bluegreens)      0.150
R.(Unicells,Unicells)          0.182
R.(Other.algae,Other.algae)    0.561

Standard errors have not been calculated.
Use MARSSparamCIs to compute CIs and bias estimates.

Convergence warnings
 Warning: the  Z.z61  parameter value has not converged.
 Warning: the  Z.z62  parameter value has not converged.
 Warning: the  Z.z33  parameter value has not converged.
 Warning: the  R.(Diatoms,Diatoms)  parameter value has not converged.
```

```
Warning: the  R.(Other.algae,Other.algae)  parameter value has not converged.
Type MARSSinfo("convergence") for more info on this warning.
```

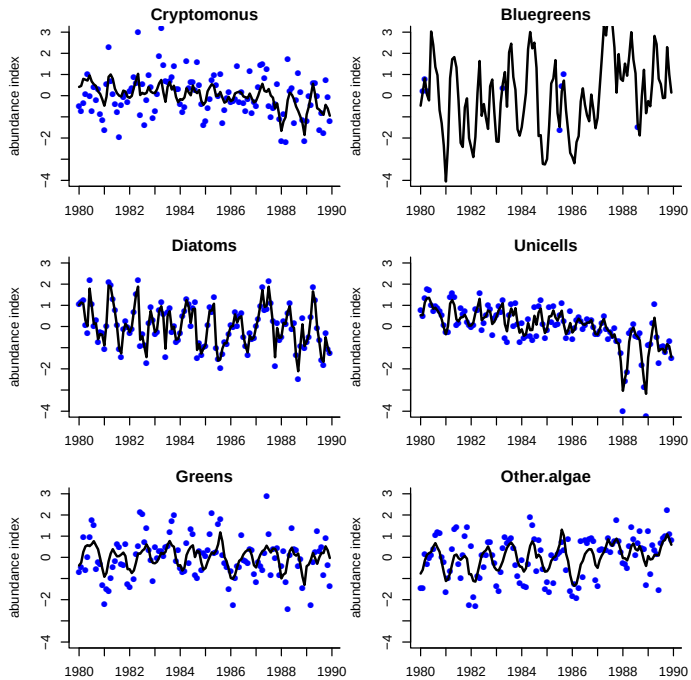

Fig. 12.2. Plots of Lake Washington phytoplankton data with model fits (dark lines) from a model with 3 trends and a diagonal and unequal variance-covariance matrix for the observation errors.

12.4 Using model selection to determine the number of trends

Following Zuur et al. (2003), we use model selection criteria (specifically AICc) to determine the number of underlying trends that have the highest data support. Our first model had three underlying trends ($m = 3$). Let's compare this to a model with two underlying trends. The forms for parameter matrix **R** and vector **a** will stay the same but we need to change the other parameter vectors and matrices because m is different.

After showing you the matrix math behind a DFA model, we will now use the `form` argument for a MARSS call to specify that we want to fit a

DFA model. This will set up the **Z** matrix and the other parameters for you. Specify how many trends you want by passing in `model=list(m=x)`. You can also pass in different forms for the **R** matrix in the usual way.

Here is how to fit two trends using `form="dfa"`:

```
model.list = list(m=2, R="diagonal and unequal")
kemz.2 = MARSS(dat.spp.1980, model=model.list,
    z.score=TRUE, form="dfa", control=cntl.list)
```

and compare its AICc value to that from the 3-trend model.

```
print(cbind(model=c("3 trends", "2 trends"),
      AICc=round(c(kemz.3$AICc, kemz.2$AICc))),
      quote=FALSE)
```

```
     model    AICc
[1,] 3 trends 1642
[2,] 2 trends 1628
```

It looks like a model with 3 trends has much more support from the data because its AICc value is about 30 units less than that for the 2-trend model.

12.4.1 Comparing many model structures

Now let's examine a larger suite of possible models. We will test from one to five underlying trends ($m = 1$ to 5) and three different structures for the **R** matrix:

1. diagonal and equal,
2. diagonal and unequal, and
3. unconstrained.

The following code builds our model matrices; you could also write out each matrix as we did in the first example, but this allows us to build and run all of the models together. (*NOTE*: the following piece of code will take a *very long* time to run!)

```
# set new control params
cntl.list = list(minit=200, maxit=1200, allow.degen=FALSE)
# set up forms of R matrices
levels.R = c("diagonal and equal",
               "diagonal and unequal",
               "unconstrained")
model.data = data.frame()
# fit lots of models & store results
# NOTE: this will take a long time to run!
for(R in levels.R) {
    for(m in 1:(N.ts-1)) {
        dfa.model = list(A="zero", R=R, m=m)
        kemz = MARSS(dat.z, model=dfa.model, control=cntl.list,
            form="dfa", z.score=TRUE)
```

```
    model.data = rbind(model.data,
                       data.frame(R=R,
                                  m=m,
                                  logLik=kemz$logLik,
                                  K=kemz$num.params,
                                  AICc=kemz$AICc,
                                  stringsAsFactors=FALSE))
    assign(paste("kemz", m, R, sep="."), kemz)
    } # end m loop
  } # end R loop
```

Model selection results are shown in Table 12.1. The model with lowest AICc has 4 trends and an unconstrained **R** matrix. It also appears that, in general, models with an unconstrained **R** matrix fit the data much better than those models with less complex structures for the observation errors (i.e., models with unconstrained forms for **R** had all of the AICc weight).

Table 12.1. Model selection results.

R	m	logLik	delta.AICc	Ak.wt	Ak.wt.cum
diagonal and equal	4	-788.4	0.0	0.35	0.35
diagonal and unequal	3	-786.4	0.1	0.32	0.67
diagonal and unequal	4	-783.7	1.2	0.19	0.87
unconstrained	1	-781.2	2.8	0.09	0.95
unconstrained	2	-776.6	4.6	0.03	0.99
diagonal and equal	5	-790.4	8.1	0.01	0.99
diagonal and equal	3	-795.9	8.6	0.00	1.00
diagonal and unequal	2	-796.4	11.6	0.00	1.00
unconstrained	3	-778.0	16.6	0.00	1.00
unconstrained	5	-773.3	18.5	0.00	1.00
diagonal and unequal	5	-792.1	22.4	0.00	1.00
unconstrained	4	-777.9	23.1	0.00	1.00
diagonal and equal	2	-808.1	24.6	0.00	1.00
diagonal and unequal	1	-808.7	25.7	0.00	1.00
diagonal and equal	1	-823.6	45.2	0.00	1.00

12.5 Using varimax rotation to determine the loadings and trends

As Harvey (1989, p. 450) discusses in section 8.5.1, there are multiple equivalent solutions to the dynamic factor loadings. We arbitrarily constrained **Z** in such a way to choose only one of these solutions, but fortunately the different solutions are equivalent, and they can be related to each other by a rotation

matrix $\mathbf{H}$. Let $\mathbf{H}$ be any $m \times m$ non-singular matrix. The following are then equivalent solutions:

$$\begin{aligned} \mathbf{y}_t &= \mathbf{Z}\mathbf{x}_t + \mathbf{a} + \mathbf{v}_t \\ \mathbf{x}_t &= \mathbf{x}_{t-1} + \mathbf{w}_t \end{aligned} \tag{12.9}$$

and

$$\begin{aligned} \mathbf{y}_t &= \mathbf{Z}\mathbf{H}^{-1}\mathbf{x}_t + \mathbf{a} + \mathbf{v}_t \\ \mathbf{H}\mathbf{x}_t &= \mathbf{H}\mathbf{x}_{t-1} + \mathbf{H}\mathbf{w}_t \end{aligned} \tag{12.10}$$

There are many ways of doing factor rotations, but a common approach is the varimax rotation which seeks a rotation matrix $\mathbf{H}$ that creates the largest difference between loadings. For example, let's say there are three trends in our model. In our estimated $\mathbf{Z}$ matrix, let's say row 3 is $(0.2, 0.2, 0.2)$. That would mean that data series 3 is equally described by trends 1, 2, and 3. If instead row 3 was $(0.8, 0.1, 0.1)$, this would make interpretation easier because we could say that data time series 3 was mostly described by trend 1. The varimax rotation finds the $\mathbf{H}$ matrix that makes the $\mathbf{Z}$ rows more like $(0.8, 0.1, 0.1)$ and less like $(0.2, 0.2, 0.2)$.

The varimax rotation is easy to compute because `R` has a built in function for this. To do so, we first get the model fits from the highest ranked model.

```
# get the "best" model
best.model = model.tbl[1,]
fitname = paste("kemz",best.model$m,best.model$R,sep=".")
best.fit = get(fitname)
```

Next, we retrieve the matrix used for varimax rotation.

```
# get the inverse of the rotation matrix
H.inv = varimax(coef(best.fit, type="matrix")$Z)$rotmat
```

Finally, we use $\mathbf{H}^{-1}$ to rotate the factor loadings and $\mathbf{H}$ to rotate the trends as in Equation 12.10.

```
# rotate factor loadings
Z.rot = coef(best.fit, type="matrix")$Z %*% H.inv
# rotate trends
trends.rot = solve(H.inv) %*% best.fit$states
```

Rotated factor loadings for the best model are shown in Figure 12.3. Oddly, some taxa appear to have no loadings on some trends (e.g., bluegreens on trend 2). The reason is that, merely for display purposes, we chose to plot only those loadings that are greater than 0.05, and it turns out that after varimax rotation, several loadings are close to 0. This is interesting given the pronounced shape of trend 4 (Figure 12.4). If we compare the loadings on trends 1-3 to those from a model with $m = 3$ (instead of $m = 4$), we would see that they are quite similar. This suggests that we might choose the model with $m = 3$ (2nd row in table) instead of $m = 4$.

Recall that we set $\mathrm{Var}(\mathbf{w}_t) = \mathbf{Q} = \mathbf{I}_m$ in order to make our DFA model identifiable. Does the variance in the process errors also change following varimax rotation? Interestingly, no. Because $\mathbf{H}$ is a non-singular, orthogonal matrix, $\mathrm{Var}(\mathbf{H}\mathbf{w}_t) = \mathbf{H}\mathrm{Var}(\mathbf{w}_t)\mathbf{H}^\top = \mathbf{H}\mathbf{I}_m\mathbf{H}^\top = \mathbf{I}_m$.

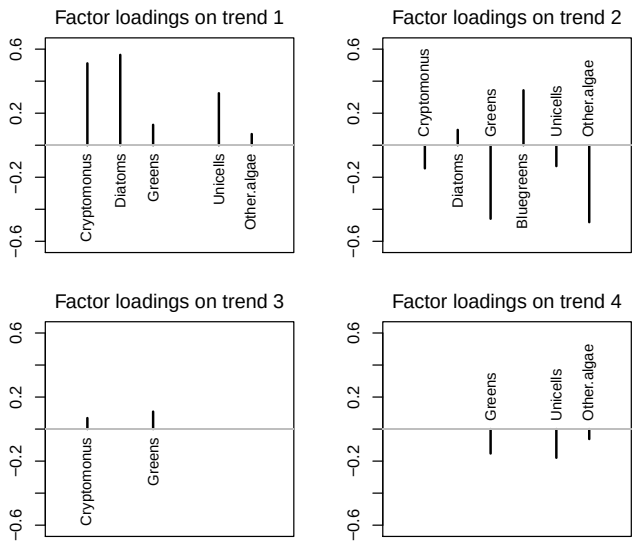

Fig. 12.3. Plot of the factor loadings (following varimax rotation) from the best model fit to the phytoplankton data.

12.6 Examining model fits

Now that we have found a "best" model and done the appropriate factor and trends rotations, we should examine some plots of model fits (see Figure 12.5). First, it looks like the model did an adequate job of capturing some of the high frequency variation (i.e., seasonality) in the time series. Second, some of the time series had much better overall fits than others (e.g., compare Diatoms versus Cryptomonas). Given the obvious seasonal patterns in the phytoplankton data, it might be worthwhile to first "detrend" the data and then repeat the model fitting exercise to see (1) how many trends would be favored, and (2) the shape of those trends.

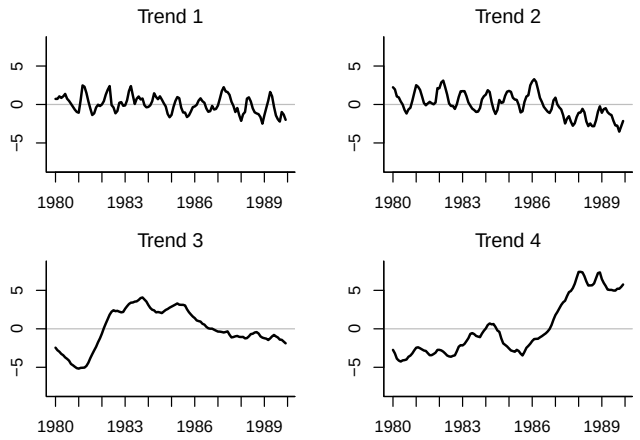

Fig. 12.4. Plot of the unobserved trends (following varimax rotation) from the best model fit to the phytoplankton data.

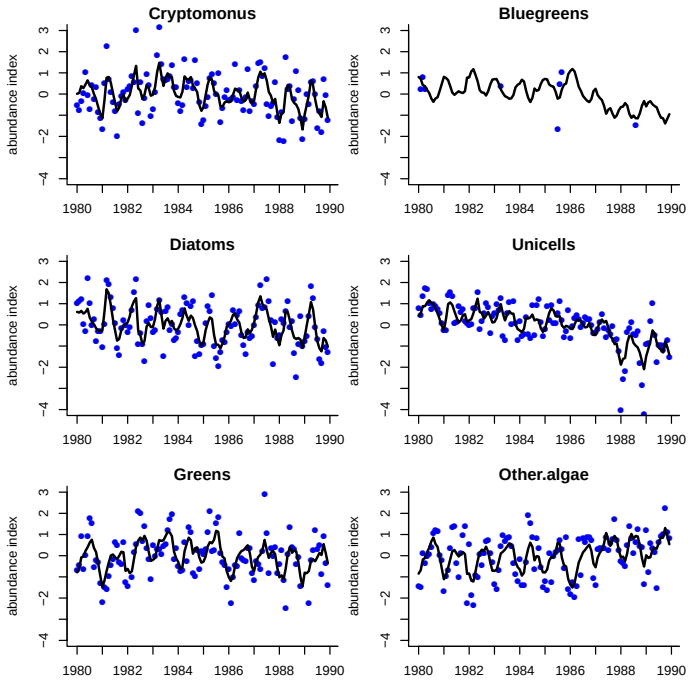

Fig. 12.5. Plot of the "best" model fits to the phytoplankton data.

12.7 Adding covariates

It is standard to add covariates to the analysis so that one removes known important drivers. The DFA with covariates is written:

$$\begin{aligned} \mathbf{x}_t &= \mathbf{x}_{t-1} + \mathbf{w}_t \text{ where } \mathbf{w}_t \sim \text{MVN}(0, \mathbf{Q}) \\ \mathbf{y}_t &= \mathbf{Z}\mathbf{x}_t + \mathbf{a} + \mathbf{D}\mathbf{d}_t + \mathbf{v}_t \text{ where } \mathbf{v}_t \sim \text{MVN}(0, \mathbf{R}) \\ \mathbf{x}_0 &\sim \text{MVN}(\boldsymbol{\pi}, \boldsymbol{\Lambda}) \end{aligned} \tag{12.11}$$

where the $q \times 1$ vector $\mathbf{d}_t$ contains the covariate(s) at time t, and the $n \times q$ matrix $\mathbf{D}$ contains the effect(s) of the covariate(s) on the observations. Using `form="dfa"` and `covariates=`<*covariate name(s)*>, we can easily add covariates to our DFA, but this means that the covariates are input, not data, and there can be no missing values (see Chapter 6 for how to include covariates with missing values).

The Lake Washington dataset has two environmental covariates that we might expect to have effects on phytoplankton growth, and hence, abundance: temperature (Temp) and total phosphorous (TP). We need the covariate inputs to have the same number of time steps as the variate data, and thus we limit the covariate data to the years 1980-1994 also.

```
temp = t(plankdat[,"Temp",drop=FALSE])
TP = t(plankdat[,"TP",drop=FALSE])
```

We will now fit 3 different models that each add covariate effects (i.e., Temp, TP, Temp & TP) to our "best" model from Table 12.1 where $m = 4$ and $\mathbf{R}$ is "unconstrained".

```
model.list=list(m=4, R="unconstrained")
kemz.temp = MARSS(dat.spp.1980, model=model.list, z.score=TRUE,
        form="dfa", control=cntl.list, covariates=temp)
kemz.TP = MARSS(dat.spp.1980, model=model.list, z.score=TRUE,
        form="dfa", control=cntl.list, covariates=TP)
kemz.both = MARSS(dat.spp.1980, model=model.list, z.score=TRUE,
        form="dfa", control=cntl.list, covariates=rbind(temp,TP))
```

Next we can compare whether the addition of the covariates improves the model fit. (*NOTE*: The following results were obtained by letting the EM algorithm run for a *very long* time, so your results may differ.)

```
print(cbind(model=c("no covars", "Temp", "TP", "Temp & TP"),
      AICc=round(c(best.fit$AICc, kemz.temp$AICc, kemz.TP$AICc,
      kemz.both$AICc))), quote=FALSE)
```

```
     model     AICc
[1,] no covars 1616
[2,] Temp      1591
[3,] TP        1639
[4,] Temp & TP 1597
```

This suggests that adding temperature or phosphorus to the model, either alone or in combination with one another, improves overall model fit.

12.8 Questions and further analyses

We analyzed the phytoplankton data alone. You can try analyzing the zooplankton data (type `head(plankdat)` to see the names). You can also try analyzing the phytoplankton and zooplankton together. You can also try different assumptions concerning the structure of **R**; we just tried unconstrained, diagonal and unequal, and diagonal and equal. To see all the `R` code behind the figures, type `RShowDoc("Case_study_4.R",package="MARSS")`. This opens a file with all the code. Copy and paste the code into a new file, and then you can edit that code. These models can take a long time to converge. In a real DFA, you will want to make sure to try different initial starting values (e.g., set `MCInit = TRUE`), and force the algorithm to run a long time by using `minit=x` and `maxit=(x+c)`, where `x` and `c` are something like 200 and 2000, respectively.

13

Case Study 5: Analyzing noisy animal tracking data

13.1 A simple random walk model of animal movement

A simple random walk model of movement with drift (directional movement) but no correlation is

$$x_{1,t} = x_{1,t-1} + u_1 + w_{1,t}, \;\; w_{1,t} \sim \mathrm{N}(0, \sigma_1^2) \tag{13.1}$$

$$x_{2,t} = x_{2,t-1} + u_2 + w_{2,t}, \;\; w_{2,t} \sim \mathrm{N}(0, \sigma_2^2) \tag{13.2}$$

where $x_{1,t}$ is the location at time t along one axis (here, longitude) and $x_{2,t}$ is for another, generally orthogonal, axis (in here, latitude). The parameter u_1 is the rate of longitudinal movement and u_2 is the rate of latitudinal movement. We add errors to our observations of location:

$$y_{1,t} = x_{1,t} + v_{1,t}, \;\; v_{1,t} \sim \mathrm{N}(0, \eta_1^2) \tag{13.3}$$

$$y_{2,t} = x_{2,t} + v_{2,t}, \;\; v_{2,t} \sim \mathrm{N}(0, \eta_2^2), \tag{13.4}$$

This model is comprised of two separate univariate state-space models. Note that y_1 depends only on x_1 and y_2 depends only on x_2. There are no actual interactions between these two univariate models. However, we can write the model down in the form of a multivariate model using diagonal variance-covariance matrices and a diagonal design ($\mathbf{Z}$) matrix. Because the variance-covariance matrices and $\mathbf{Z}$ are diagonal, the x_1:y_1 and x_2:y_2 processes will be independent as intended. Here are Equations 13.2 and 13.4 written as a MARSS model (in matrix form):

$$\begin{bmatrix} x_{1,t} \\ x_{2,t} \end{bmatrix} = \begin{bmatrix} x_{1,t-1} \\ x_{2,t-1} \end{bmatrix} + \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} + \begin{bmatrix} w_{1,t} \\ w_{2,t} \end{bmatrix}, \;\; \mathbf{w}_t \sim \mathrm{MVN}\left(0, \begin{bmatrix} \sigma_1^2 & 0 \\ 0 & \sigma_2^2 \end{bmatrix}\right) \tag{13.5}$$

$$\begin{bmatrix} y_{1,t} \\ y_{2,t} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_{1,t} \\ x_{2,t} \end{bmatrix} + \begin{bmatrix} v_{1,t} \\ v_{2,t} \end{bmatrix}, \;\; \mathbf{v}_t \sim \mathrm{MVN}\left(0, \begin{bmatrix} \eta_1^2 & 0 \\ 0 & \eta_2^2 \end{bmatrix}\right) \tag{13.6}$$

The variance-covariance matrix for $\mathbf{w}_t$ is a diagonal matrix with unequal variances, σ_1^2 and σ_2^2. The variance-covariance matrix for $\mathbf{v}_t$ is a diagonal matrix with unequal variances, η_1^2 and η_2^2. We can write this succinctly as

$$\mathbf{x}_t = \mathbf{x}_{t-1} + \mathbf{u} + \mathbf{w}_t, \;\; \mathbf{w}_t \sim \mathrm{MVN}(0, \mathbf{Q}) \tag{13.7}$$

$$\mathbf{y}_t = \mathbf{x}_t + \mathbf{v}_t, \;\; \mathbf{v}_t \sim \mathrm{MVN}(0, \mathbf{R}). \tag{13.8}$$

13.2 Loggerhead sea turtle tracking data

Loggerhead sea turtles (*Caretta caretta*) are listed as threatened under the United States Endangered Species Act of 1973. Over the last ten years, a number of state and local agencies have been deploying ARGOS tags on loggerhead turtles on the east coast of the United States. We have data on eight individuals over that period. In this case study, we use some turtle data from the WhaleNet Archive of STOP Data, however we have corrupted this data severely by adding random errors in order to create a "bad tag" problem. We corrupted latitude and longitude data by errors (Figure 13.1) and it would appear that our sea turtles are becoming land turtles (at least part of the time). For this case study, we will the MARSS model to estimate true positions and speeds from the corrupted data.

13.2.1 Read in the data and load `maps` package

Our noisy data are in `loggerheadNoisy`. They consist of daily readings of location (longitude/latitude). If data are missing for a day, then the entries for latitude and longitude for that day should be NA. However, to make this case study run quickly, we have interpolated all missing values in the original, uncorrupted, dataset (`loggerhead`). The first six lines of the corrupted data look like so

```
loggerheadNoisy[1:6,]
```

```
   turtle month day year       lon      lat
1 BigMama     5  28 2001 -81.45989 31.70337
2 BigMama     5  29 2001 -80.88292 32.18865
3 BigMama     5  30 2001 -81.27393 31.67568
4 BigMama     5  31 2001 -81.59317 31.83092
5 BigMama     6   1 2001 -81.35969 32.12685
6 BigMama     6   2 2001 -81.15644 31.89568
```

The file has data for eight turtles:

```
turtles=levels(loggerheadNoisy$turtle)
turtles
```

```
[1] "BigMama"  "Bruiser"  "Humpty"   "Isabelle" "Johanna"
[6] "MaryLee"  "TBA"      "Yoto"
```

We will analyze the position data for "Big Mama". We put the data for "Big Mama" into matrix `dat`. `dat` is transposed because we need time across the columns.

```
turtlename="BigMama"
dat = loggerheadNoisy[which(loggerheadNoisy$turtle==turtlename),5:6]
dat = t(dat) #transpose
```

We will use the `maps` R package to plot the data and results. You will need to install this R package in order to run the example code. Figure 13.1 shows the corrupted location data for Big Mama.

13.3 Estimate locations from bad tag data

We will begin by specifying the structure of the MARSS model and then use `MARSS()` to fit that model to the data. There are two state processes (one for latitude and the other for longitude), and there is one observation time series for each state process. As we saw in Equation 13.6, $\mathbf{Z}$ is the an identity matrix (a diagonal matrix with 1s on the diagonal). We could specify this structure as `Z.model="identity"` or `Z.model=factor(c(1,2))`. Although technically, this is unnecessary as this is the default form for $\mathbf{Z}$.

We will assume that the errors are independent and that there are different drift rates (u), process variances (σ^2) and observation variances for latitude and longitude (η^2).

```
Z.model="identity"
U.model="unequal"
Q.model="diagonal and unequal"
R.model="diagonal and unequal"
```

Fit the model to the data:

```
kem = MARSS(dat, model=list(Z = Z.model,
      Q = Q.model, R = R.model, U = U.model))
```

13.3.1 Compare state estimates to the real positions

The real locations (from which `loggerheadNoisy` was produced by adding noise) are in `loggerhead`. In Figure 13.2, we compare the tracks estimated from the noisy data with the original, good, data. There are only a few data points for the real data because in the real tag data, there are many missing days.

```
#load the map package; you have to install it first
library(maps)
# Read in our noisy data (no missing values)
pdat = loggerheadNoisy  #for plotting
turtlename="BigMama"
par(mai = c(0,0,0,0),mfrow=c(1,1))
map('state', region = c('florida', 'georgia', 'south carolina', 'north carolina',
    'virginia', 'delaware','new jersey','maryland'),xlim=c(-85,-70))
points(pdat$lon[which(pdat$turtle==turtlename)], pdat$lat[which(pdat$turtle==turtlename)],
    col="blue",pch=21, cex=0.7)
```

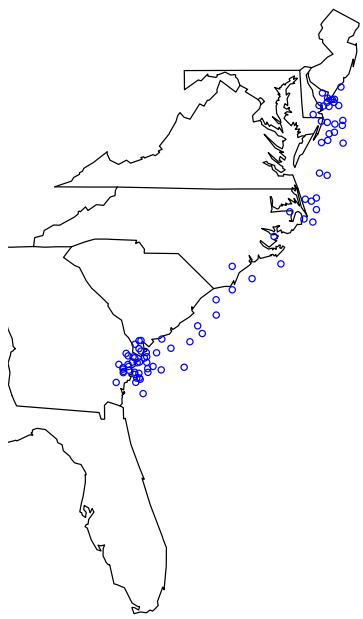

Fig. 13.1. Plot of the tag data from the turtle Big Mama. Errors in the location data make it seem that Big Mama has been moving overland.

13.3.2 Estimate speeds for each turtle

Turtle biologists designated one of these loggerheads "Big Mama," presumably for her size and speed. For each of the eight turtles, estimate the average miles traveled per day. To calculate the distance traveled by a turtle each day, you use the estimate (from `MARSS()`) of the lat/lon location of turtle at day t and at day $t-1$. To calculate distance traveled in miles from lat/lon start and finish locations, we will use the function `GCDF` defined below:

```
pred.lon = kem$states[1,]
pred.lat = kem$states[2,]
par(mai = c(0,0,0,0),mfrow=c(1,1))
library(maps)
pdat=loggerheadNoisy
turtlename="BigMama"
map('state', region = c('florida', 'georgia', 'south carolina', 'north carolina',
     'virginia', 'delaware','new jersey','maryland'),xlim=c(-85,-70))
points(pdat$lon[which(pdat$turtle==turtlename)], pdat$lat[which(pdat$turtle==turtlename)],
     col="blue",pch=21, cex=0.7)
lines(pred.lon, pred.lat,col="red", lwd=2)
goodturtles = loggerhead
gooddat = goodturtles[which(goodturtles$turtle==turtlename),5:6]
points(gooddat[,1], gooddat[,2],col="black", lwd=2, pch=3,cex=1.1)
legend("bottomright",c("bad locations", "estimated true location",
  "good location data"),pch=c(1,-1,3),lty=c(-1,1,-1),
   col=c("blue","red","black"), bty=FALSE)
```

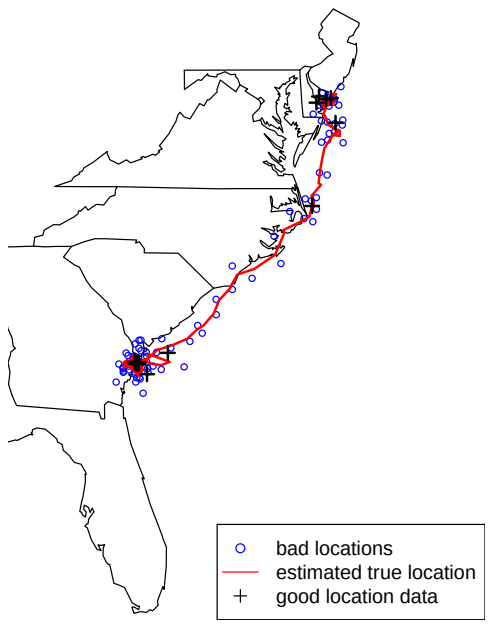

Fig. 13.2. Plot of the estimated track of the turtle Big Mama versus the good location data (before we corrupted it with noise).

```
GCDF <- function(lon1, lon2, lat1, lat2, degrees=TRUE, units="miles") {
   temp = ifelse(degrees==FALSE,
     acos(sin(lat1)*sin(lat2)+cos(lat1)*cos(lat2)*cos(lon2-lon1)),
     acos(sin(lat1/57.2958)*sin(lat2/57.2958)+cos(lat1/57.2958)*cos(lat2/57.2958)
       *cos(lon2/57.2958-lon1/57.2958)))
   r=3963.0 # (statute miles) , default
   if("units"=="nm") r=3437.74677 # (nautical miles)
   if("units"=="km") r=6378.7 # (kilometers)
   return (r * temp)
}
```

We can now compute the distance traveled each day by passing in lat/lon estimates from day $i-1$ and day i:

```
distance[i-1]=GCDF(pred.lon[i-1],pred.lon[i],
                   pred.lat[i-1],pred.lat[i])
```

`pred.lon` and `pred.lat` are the predicted longitudes and latitudes from `MARSS()`: rows one and two in `kem$states`. To calculate the distances for all days, we put this through a `for` loop:

```
distance = array(NA, dim=c(dim(dat)[2]-1,1))
for(i in 2:dim(dat)[2])
   distance[i-1]=GCDF(pred.lon[i-1],pred.lon[i],
      pred.lat[i-1],pred.lat[i])
```

The command `mean(distance)` gives us the average distance per day. We can also make a histogram of the distances traveled per day (Figure 13.3).

We can compare the histogram of daily distances to what we would get if we had not accounted for measurement error (Figure 13.4). We can also compare the mean miles per day:

```
#accounting for observation error
mean(distance)
```

```
[1] 15.53858
```

```
#assuming the data have no observation error
mean(distance.noerr)
```

```
[1] 34.80579
```

13.4 Using specialized packages to analyze tag data

If you have real tag data to analyze, you should use a state-space modeling package that is customized for fitting MARSS models to tracking data. The MARSS package does not have all the bells and whistles that you would want for analyzing tracking data, particularly tracking data in the marine

```
par(mfrow=c(1,1))
hist(distance) #make a histogram of distance traveled per day
```

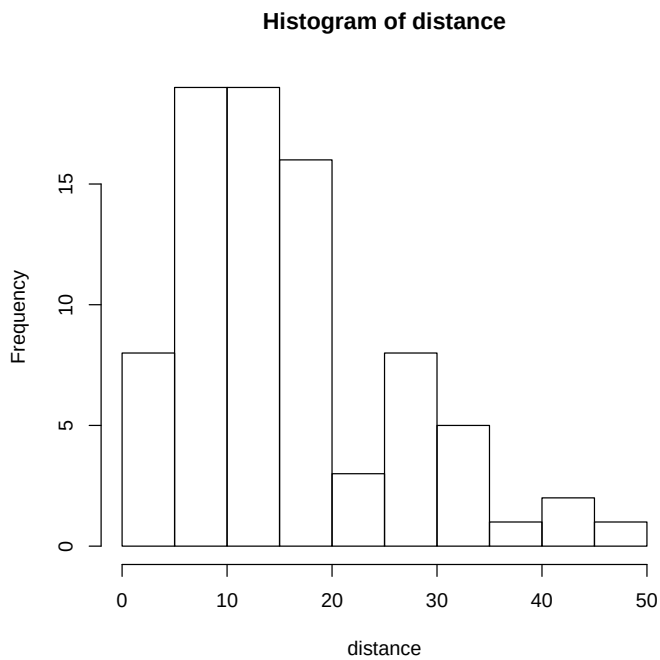

Fig. 13.3. Histogram of the miles traveled per day for Big Mama with estimates that account for measurement error in the data.

environment. These are a couple R packages that we have come across for this purpose:

UKFSST http://www.soest.hawaii.edu/tag-data/tracking/ukfsst/
KFTRACK http://www.soest.hawaii.edu/tag-data/tracking/kftrack/

`kftrack` is a full-featured toolbox for analyzing tag data with extended Kalman filtering. It incorporates a number of extensions that are important for analyzing track data: barriers to movement such as coastlines and non-Gaussian movement distributions. With `kftrack`, you can use the real tag data which has big gaps, i.e. days with no location. `MARSS()` will struggle with these data because it will estimate states for all the unseen days; `kftrack` only fits to the seen days.

To use `kftrack` to fit the turtle data, type

```
library(kftrack) # must be installed from a local zip file
loggerhead = loggerhead
```

```
# Compare to the distance traveled per day if you used the raw data
distance.noerr = array(NA, dim=c(dim(dat)[2]-1,1))
for(i in 2:dim(dat)[2])
  distance.noerr[i-1]=GCDF(dat[1,i-1],dat[1,i],dat[2,i-1],dat[2,i])
hist(distance.noerr) #make a histogram of distance traveled per day
```

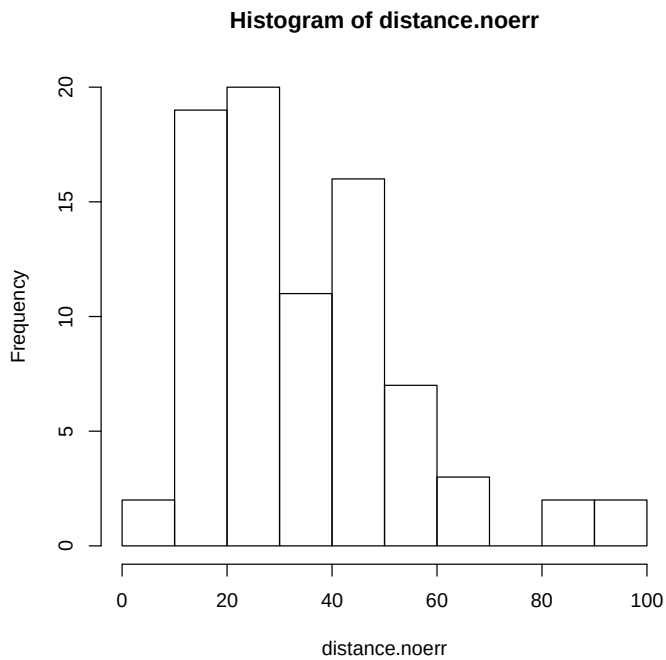

Fig. 13.4. Histogram of the miles traveled per day for Big Mama with estimates that account for measurement error in the data.

```
# Run kftrack with the first turtle (BigMama)
turtlename = "BigMama"
dat = loggerhead[which(loggerhead$turtle == turtlename),2:6]
model = kftrack(dat, fix.first=F, fix.last=F,
        var.struct="uniform")
```

13.5 Questions and further analyses

1. Repeat the analysis done for "Big Mama" for each of the other turtles and fill out the speed table (Table 13.1) with the mean daily distances traveled assuming the data have measurement error versus assuming they do not.

Table 13.1. Estimated speeds with location errors included in model versus speeds (mean distance per day) when we assume that the data have no location error.

Turtle	Location error included in model	Data assumed to be error free
Big Mama		
Bruiser		
Humpty		
Isabelle		
Johanna		
Mary Lee		
TBA		
Yoto		

2. Compare turtle tracks to a proposed fishing area (say). You can add areas to your maps in R by calling the following directly after making one of the map plots:

```
lines(c(-77,-78,-78,-77,-77),
      c(33.5,33.5,32.5,32.5,33.5),col="red",lwd=2)
lines(c(-75,-76,-76,-75,-75),
      c(38,38,37,37,38),col="red",lwd=2)
```

14

Case Study 6: Detection of outliers and structural breaks

14.1 River flow in the Nile River

This case study is based on a short example shown on page 147-148 in Koopman et al. (1999) using a 100-year record of river flow on the Nile River. The methods are based on Harvey et al. (1998) which is in turn based on techniques in Harvey and Koopman (1992) and Koopman (1993). The Nile dataset is included in R . Figure 14.1 shows the data. To see all the R code behind the figures in this chapter, type `RShowDoc("Case_study_6.R",package="MARSS")`.

14.2 Different models for the Nile flow levels

We begin by fitting different flow models to the data and compare these models with AIC. After that, we will use the model residuals to look for outliers and structural breaks.

14.2.1 Flat level model

We will start by modeling these data as a simple average river flow with variability around this level.

$$y_t = a + v_t \text{ where } v_t \sim \mathrm{N}(0, r) \tag{14.1}$$

where y_t is the river flow volume at year t and x is some constant average flow level (notice it has no t subscript).

To fit this model with MARSS, we will explicitly show all the MARSS parameters.

$$\begin{aligned} x_t &= 1 \times x_{t-1} + 0 + w_t \text{ where } w_t \sim \mathrm{N}(0,0) \\ y_t &= 0 \times x_t + a + v_t \text{ where } v_t \sim \mathrm{N}(0,r) \\ x_0 &= 0 \end{aligned} \tag{14.2}$$

```
#load the datasets package
library(datasets)
data(Nile)   #load the data
plot(Nile,ylab="Flow volume",xlab="Year")
```

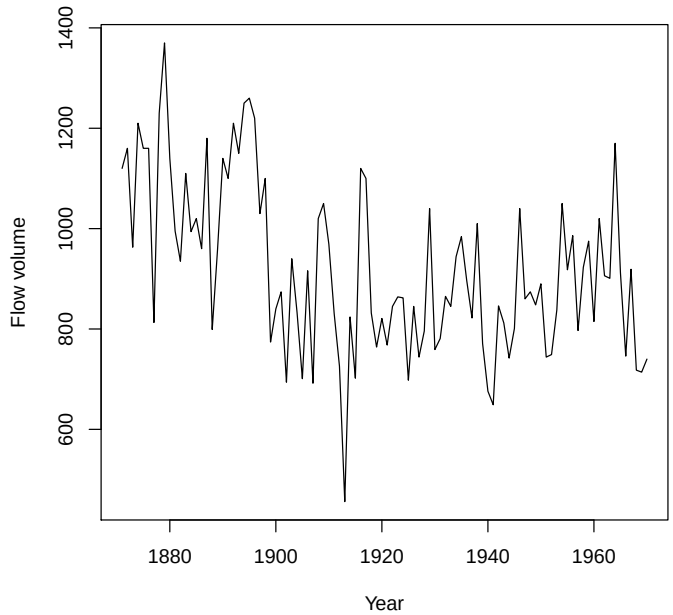

Fig. 14.1. The Nile River flow volume 1871 to 1970 (included dataset in R).

MARSS includes the state process x_t but we are setting $\mathbf{Z}$ to zero so that does not appear in our observation model. We need to fix all the state parameters to zero so that the algorithm doesn't "chase its tail" trying to fit x_t to the data.

An equivalent way to write this model is to use x_t as the average flow level and make it be a constant level by setting $q = 0$. The average flow appears as the x_0 parameter. In MARSS form, the model is:

$$\begin{aligned} x_t &= 1 \times x_{t-1} + 0 + w_t \text{ where } w_t \sim \mathrm{N}(0,0) \\ y_t &= 1 \times x_t + 0 + v_t \text{ where } v_t \sim \mathrm{N}(0,r) \\ x_0 &= a \end{aligned} \tag{14.3}$$

We will use this latter format since we will be building on this form. The model is specified as a list as follows and we denote this model "0":

```
mod.nile.0 = list(
Z=matrix(1), A=matrix(0), R=matrix("r"),
B=matrix(1), U=matrix(0), Q=matrix(0),
x0=matrix("a") )
```

We then fit the model with `MARSS()`:

```
#The data is in a ts format, and we need a matrix
dat = t(as.matrix(Nile))
#Now we fit the model
kem.0 = MARSS(dat, model=mod.nile.0)
```

```
Success! algorithm run for 15 iterations. abstol and log-log tests passed.
Alert: conv.test.slope.tol is 0.5.
Test with smaller values (<0.1) to ensure convergence.

MARSS fit is
Estimation method: kem
Convergence test: conv.test.slope.tol = 0.5, abstol = 0.001
Algorithm ran 15 (=minit) iterations and convergence was reached.
Log-likelihood: -654.5157
AIC: 1313.031   AICc: 1313.155

     Estimate
R.r     28352
x0.a      919

Standard errors have not been calculated.
Use MARSSparamCIs to compute CIs and bias estimates.
```

14.2.2 Linear trend in flow model

Figure 14.2 shows the fit for the flat average river flow model. Looking at the data, we might expect that a declining average river flow would be better. In MARSS form, that model would be:

$$\begin{aligned} x_t &= 1 \times x_{t-1} + u + w_t \text{ where } w_t \sim \mathrm{N}(0,0) \\ y_t &= 1 \times x_t + 0 + v_t \text{ where } v_t \sim \mathrm{N}(0,r) \\ x_0 &= a \end{aligned} \tag{14.4}$$

where u is now the average per-year decline in river flow volume. The model is specified as a list as follows and we denote this model "1":

```
mod.nile.1 = list(
Z=matrix(1), A=matrix(0), R=matrix("r"),
B=matrix(1), U=matrix("u"), Q=matrix(0),
x0=matrix("a") )
```

We then fit the model with `MARSS()`:

```
kem.1 = MARSS(dat, model=mod.nile.1)
```

```
Success! abstol and log-log tests passed at 18 iterations.
Alert: conv.test.slope.tol is 0.5.
Test with smaller values (<0.1) to ensure convergence.

MARSS fit is
Estimation method: kem
Convergence test: conv.test.slope.tol = 0.5, abstol = 0.001
Estimation converged in 18 iterations.
Log-likelihood: -642.3159
AIC: 1290.632   AICc: 1290.882

     Estimate
R.r  22213.60
U.u     -2.69
x0.a  1054.94

Standard errors have not been calculated.
Use MARSSparamCIs to compute CIs and bias estimates.
```

Figure 14.2 shows the fits for the two models with deterministic models (flat and declining) for mean river flow along with their AICc values (smaller AICc is better). The AICc for the model with a declining river flow is lower by over 20 (which is a lot).

14.2.3 Stochastic level model

Looking at the flow levels, we might suspect that a model that allows the average flow to change would model the data better and we might suspect that there have been sudden, and anomalous, changes in the river flow level. We will now model the average river flow at year t as a random walk, specifically an autoregressive process which means that average river flow is year t is a function of average river flow in year $t-1$.

$$\begin{aligned} x_t &= x_{t-1} + w_t \text{ where } w_t \sim \mathrm{N}(0,q) \\ y_t &= x_t + v_t \text{ where } v_t \sim \mathrm{N}(0,r) \\ x_0 &= \pi \end{aligned} \tag{14.5}$$

As before, y_t is the river flow volume at year t. With all the MARSS parameters shown, the model is:

$$\begin{aligned} x_t &= 1 \times x_{t-1} + 0 + w_t \text{ where } w_t \sim \mathrm{N}(0,q) \\ y_t &= 1 \times x_t + 0 + v_t \text{ where } v_t \sim \mathrm{N}(0,r) \\ x_0 &= \pi \end{aligned} \tag{14.6}$$

Thus, $\mathbf{Z} = 1$, $\mathbf{a} = 0$, $\mathbf{R} = r$, $\mathbf{B} = 1$, $\mathbf{u} = 0$, $\mathbf{Q} = q$, and $\mathbf{x}_0 = \pi$. The model is then specified as:

```
mod.nile.2 = list(
Z=matrix(1), A=matrix(0), R=matrix("r"),
B=matrix(1), U=matrix(0), Q=matrix("q"),
x0=matrix("pi") )
```

We could also use the text shortcuts to specify the model. Because **R** and **Q** are 1×1 matrices, "unconstrained", "diagonal and unequal", "diagonal and equal" and "equalvarcov" will all lead to a 1×1 matrix with one estimated element. For **a** and **u**, the following shortcut could be used:

```
A=U="zero"
```

Because $\mathbf{x}_0$ is 1×1, it could be specified as "unequal", "equal" or "unconstrained".

We fit the model with the `MARSS()` function. We are using the "BFGS" algorithm to polish off the estimates, since it will get the maximum faster than the default EM algorithm as long as we start it close to the maximum.

```
kem.2em = MARSS(dat, model=mod.nile.2, silent=TRUE)
kem.2 = MARSS(dat, model=mod.nile.2,
  inits=kem.2em$par, method="BFGS")
```

```
Success! Converged in 12 iterations.
Function MARSSkfas used for likelihood calculation.

MARSS fit is
Estimation method: BFGS
Estimation converged in 12 iterations.
Log-likelihood: -637.7451
AIC: 1281.49   AICc: 1281.74

      Estimate
R.r      15337
Q.q       1218
x0.pi     1112

Standard errors have not been calculated.
Use MARSSparamCIs to compute CIs and bias estimates.
```

This is the same model fit in Koopman et al. (1999, p. 148) except that we estimate x_1 as parameter rather than specifying x_1 via a diffuse prior. As a result, the log-likelihood value and **R** and **Q** are a little different than in Koopman et al. (1999).

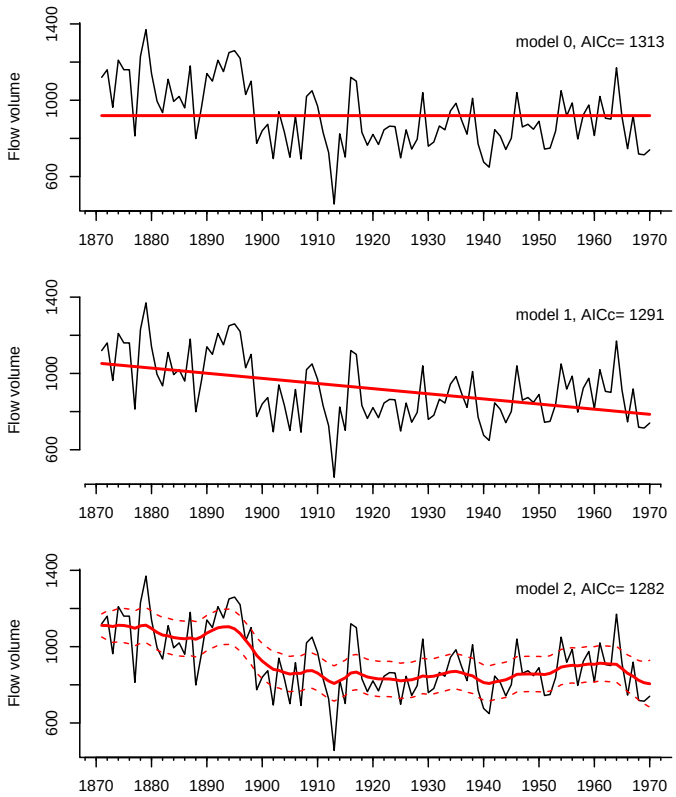

Fig. 14.2. The Nile River flow volume with the model estimated flow rates (solid lines). The bottom model is a stochastic level model, and the 2 standard deviations for the level are also shown. The other two models are deterministic level models so the state is not stochastic and does not have a standard deviation.

14.3 Observation and state residuals

Figure 14.2 shows the MARSS fits to the data. From these model fits, auxiliary residuals can be computed which contain information about whether the data and models fits at time t differ more than you would expect given the model and the model fits at time $t-1$. In this section, we follow the example shown on page 147-148 in Koopman et al. (1999) and use these residuals to look for outliers and sudden flow level changes. Using auxiliary residuals this way follows mainly from Harvey and Koopman (1992), but see also Koopman (1993, sec. 3), de Jong and Penzer (1998) and Penzer (2001) for discussions of using auxiliary residuals for detection of outliers and structural breaks.

The MARSS function will output the expected values of x_t conditioned on the maximum-likelihood values of q, r, and x_1 and on the data (y from $t = 1$ to T). In time-series literature, these are called the smoothed state estimates and they are output by the Kalman filter-smoother. We will call these smoothed estimates $\tilde{x}_{t|T}$ (and are `xtT` in the MARSS output). The time value after the | in the subscript indicates the data on which the estimate was conditioned (in this case, 1 to T). From these, we can compute the model predicted value of y_t, denoted or $\hat{y}_{t|T}$. This is the predicted value of y_t conditioned on $\tilde{x}_{t|T}$.

$$\begin{aligned}\tilde{x}_{t|T} &= \mathrm{E}(X_t|\hat{\theta}, y_1^T)\\ \hat{y}_{t|T} &= \mathrm{E}(Y_t|\hat{\theta}, \tilde{x}_{t|T})\\ &= \tilde{x}_{t|T} + \mathrm{E}(w_t|\hat{\theta}, y_1^T) = \tilde{x}_{t|T}\end{aligned} \tag{14.7}$$

where $\hat{\theta}$ are the maximum-likelihood estimates of the parameters. The $\hat{y}_{t|T}$ equation comes directly from equation (14.5). This expectation is not conditioned on the data y_1^T, directly. It is conditioned on $\tilde{x}_{t|T}$, which is conditioned on y_1^T.

14.3.1 Using observation residuals to detect outliers

The standardized smoothed observation residuals[1] are the difference between the data at time t and the model fit at time t conditioned on all the data standardized by the observation variance:

$$\begin{aligned}\hat{v}_t &= y_t - \hat{y}_{t|T}\\ e_t &= \frac{1}{\sqrt{\mathrm{var}(\hat{v}_t)}}\hat{v}_t\end{aligned} \tag{14.8}$$

These residuals should have (asymptotically) a t-distribution (Kohn and Ansley, 1989, sec. 3) and by looking at the residuals, we can identify potential outlier data points–or more accurately, we can identify data points that do not fit the model (Equation 14.5). The call `residuals()` will compute these residuals for a marssMLE object (output by a MARSS call). It returns the standardized residuals (also called auxiliary residuals) as a $n + m \times T$ matrix. The first n rows are the estimated $\mathbf{v}_t$ standardized observation residuals and the next m rows are the estimated $\mathbf{w}_t$ standardized state residuals (discussed below).

```
resids.0=residuals(kem.0)$std.residuals
resids.1=residuals(kem.1)$std.residuals
resids.2=residuals(kem.2)$std.residuals
```

[1] also called smoothations in the literature to distinguish them from innovations, which are $\mathbf{y}_t - \mathrm{E}(\mathbf{Y}_t|\tilde{x}_{t|t-1})$. Notice that for innovations the expectation is conditioned on the data up to time $t-1$ while for smoothations, we condition on all the data.

Figure 14.3 shows the observation residuals for the three models developed above. We immediately see that model 0 (flat level) and model 1 (linear declining level) have problems because the residuals are all positive for the first part of the time series and then all negative. The residuals should not be temporally correlated like that. Model 2 with a stochastic level shows well-behaving residuals with low temporal correlation between t and $t-1$. Looking at the residuals for model 2, we see that there are a number of years with flow levels that appear to be outliers (are beyond the dashed level lines).

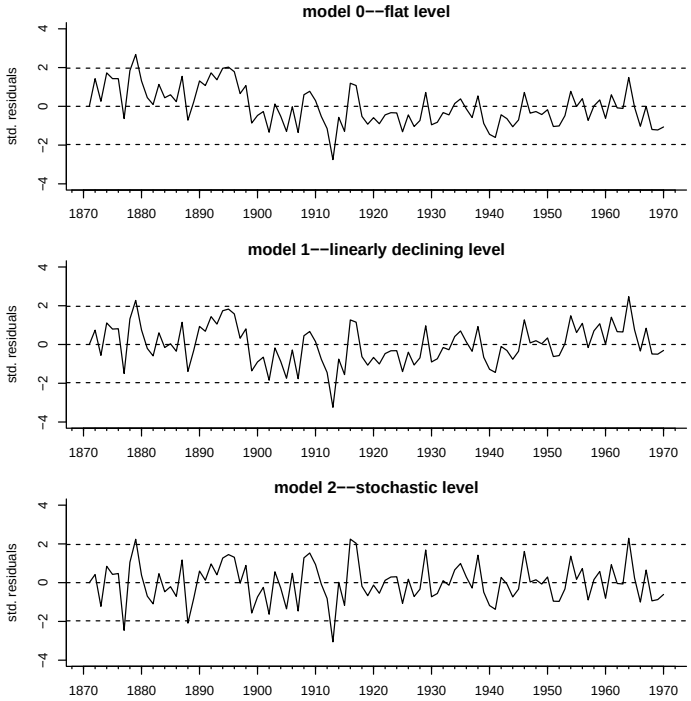

Fig. 14.3. The standardized observation residuals from models 0, 1, and 2. These residuals are the standardized $\hat{v}_t$. The dashed lines are the 95% CIs for a t-distribution.

14.3.2 Detecting sudden level changes

The standardized smoothed state residuals (f_t below) are the difference between the estimated state at time t and the estimated state at time $t-1$ conditioned on all the data standardized by the standard deviation:

$$\begin{aligned} \hat{w}_t &= \tilde{x}_{t|T} - \tilde{x}_{t-1|T} \\ f_t &= \frac{1}{\sqrt{\mathrm{var}(\hat{w}_t)}} \hat{w}_t \end{aligned} \tag{14.9}$$

These state residuals do not show simple changes in the average level; x_t is clearly changing in Figure 14.2, bottom panel. Instead we are looking for "breaks" or sudden changes in the level. The bottom panel of Figure 14.4 shows the standardized state residuals (f_t). This shows, as we can see by eye, the average flow level in the Nile appears to have suddenly changed around the turn of the century when the first Aswan dam was built. The top panel shows the standardized observation residuals for comparison.

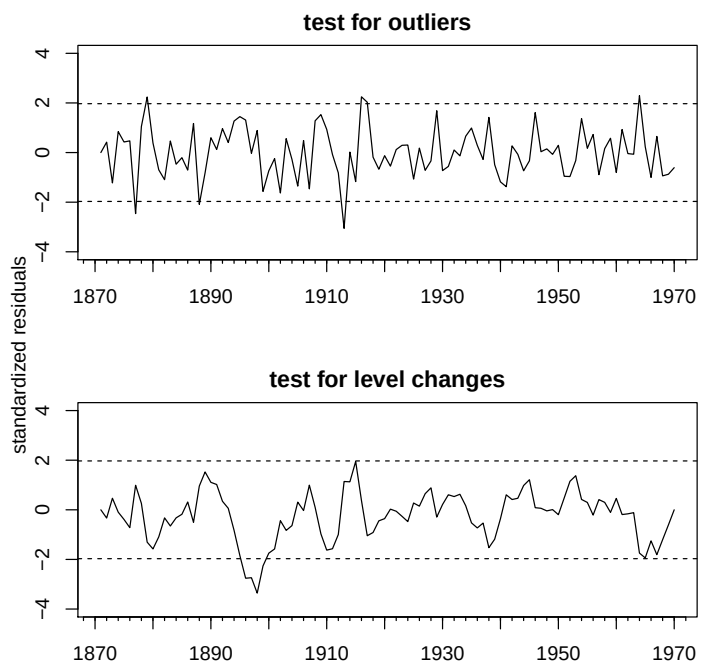

Fig. 14.4. Top panel, the standardized observation residuals. Bottom panel, the standardized state residuals. This replicates Figure 12 in Koopman et al. (1999).

15

Case Study 7: Estimation of species interaction strengths with and without covariates

15.1 Background

Multivariate autoregressive models (commonly termed MAR models) have been developed as a tool for analyzing community dynamics from time series data (Ives, 1995; Ives et al., 1999, 2003). This approach has been used estimate species interaction strengths, stability metrics, and environmental drivers for a variety of freshwater plankton systems (Ives, 1995; Ives et al., 1999, 2003; Hampton et al., 2008, 2006; Hampton and Schindler, 2006; Klug and Cottingham, 2001). These models are based on a process model for log abundances ($\mathbf{x}$) of the form

$$\mathbf{x}_t = \mathbf{B}\mathbf{x}_{t-1} + \mathbf{u} + \mathbf{w}_t \text{ where } \mathbf{w}_t \sim \text{MVN}(0, \mathbf{Q}) \tag{15.1}$$

$\mathbf{B}$ is the interaction matrix; self interaction strengths (density-dependence) are on the diagonal and inter-specific interaction strengths are on the off-diagonals such that $\mathbf{B}_{i,j}$ is the 'effect' of species j on species i. This model has a stochastic equilibrium—it fluctuates around mean, $(\mathbf{I}-\mathbf{B})^{-1}\mathbf{u}$. The term $\mathbf{u}$ determines the mean level but once the system is at equilibrium, it does not affect the fluctuations relative to the mean. To see this, compare two models with $b = 0.5$ and $u = 1$ versus $u = 0$. The mean for the first is $1/(1-0.5) = 2$ and for the second is 0. If we start both 1 above the mean, the next x is the same distance from the mean: $x_2 = 0.5(2+1)+1 = 2.5$ and $x_2 = 0.5(0+1)+0 = 0.5$. So both end up at 0.5 above the mean. So once the system is at equilibrium, it is 'scale invariant', where $\mathbf{u}$ is the scaling term. The way that Ives et al. (2003) write their process model (their Equation 10) is $\mathbf{X}_t = \mathbf{A} + \mathbf{B}\mathbf{X}_{t-1} + \mathbf{E}_t$. The $\mathbf{A}$ in Ives's equation is the $\mathbf{u}$ appearing in Equation 15.1 and the $\mathbf{E}_t$ is our $\mathbf{w}_t$.

Often the models include environmental covariates, but we will leave that off for the moment and address that at the end of this case study. If we add

a measurement process[1], we have a MARSS model:

$$\mathbf{y}_t = \mathbf{Z}\mathbf{x}_t + \mathbf{a} + \mathbf{v}_t \text{ where } \mathbf{v}_t \sim \text{MVN}(0, \mathbf{R}) \tag{15.2}$$

Typically, we have one time series per species and thus we assume that $m = n$ and $\mathbf{Z}$ is an $m \times m$ identity matrix (when $m = n$, $\mathbf{a}$ is set to 0). However, it is certainly possible to have multiple time series per species (for example data taken at multiple sites).

In this case study, we will estimate the $\mathbf{B}$ matrix of species interactions for a simple wolf-moose system and for a four-species freshwater plankton system.

15.2 Two-species example using wolves and moose

Population dynamics of wolves and moose on Isle Royale, Michigan make an interesting case study of a two-species predator-prey interactions. These populations have been studied intensively since 1958(**?**) [2]. Unlike other populations of gray wolves, the Isle Royale population has a diet dominated by one prey item, moose. The only predator of moose on Isle Royale is the gray wolf, as this population is not hunted.

We will use the wolf and moose winter census data from Isle Royale to learn how to fit community dynamics models to time-series data. The long-term January (wolf) and February (moose) population estimates are provided at `http://www.isleroyalewolf.org`.

The mathematical form of the process model for the wolf-moose population dynamics is

$$\begin{gathered}\begin{bmatrix} x_{wolf} \\ x_{moose} \end{bmatrix}_t = \begin{bmatrix} b_{ww} & b_{w\to m} \\ b_{m\to w} & b_{mm} \end{bmatrix} \begin{bmatrix} x_{wolf} \\ x_{moose} \end{bmatrix}_{t-1} + \begin{bmatrix} u_{wolf} \\ u_{moose} \end{bmatrix} + \begin{bmatrix} w_{wolf} \\ w_{moose} \end{bmatrix}_t \\ \begin{bmatrix} w_{wolf} \\ w_{moose} \end{bmatrix}_t \sim \text{MVN}\left(0, \begin{bmatrix} q_{wolf} & 0 \\ 0 & q_{moose} \end{bmatrix}\right)\end{gathered} \tag{15.3}$$

15.2.1 Load in and plot the data

```
royale.dat = log(t(isleRoyal[,2:3]))
```

[1] You can fit a MAR model with no observation error by setting $\mathbf{R} = 0$, but a conditional least-squares algorithm is vastly faster than EM or BFGS for the $\mathbf{R} = 0$ case (assuming no missing data).

[2] There are many, many publications from this long-term study site; see `http://www.isleroyalewolf.org/wolfhome/tech_pubs.html` and the review here `http://www.isleroyalewolf.org/data/data/home.html`.

```
matplot(isleRoyal[,1],log(isleRoyal[,2:3]),
    ylab="log count",xlab="Year",type="l",
    lwd=3,bty="L",col="black")
legend("topright",c("Wolf","Moose"), lty=c(1,2), bty="n")
```

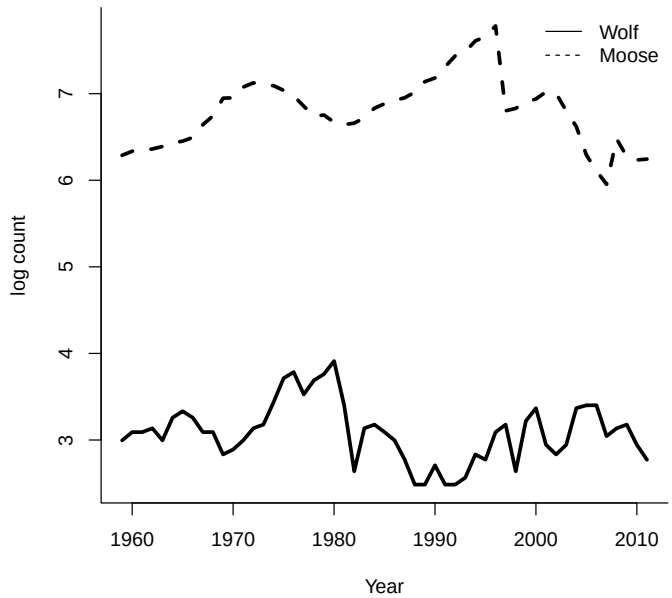

Fig. 15.1. Plot of the Isle Royale wolf and moose data.

15.2.2 Fit the model to the wolf-moose data

The naive way to fit the model is to use Equations 15.2 and 15.1 "as is":

```
royale.model.0=list(B="unconstrained",Q="diagonal and unequal",
      R="diagonal and unequal",U="unequal")
kem.0=MARSS(royale.dat,model=royale.model.0)
```

If you try this, you will notice that it does not converge but stops when it reaches `maxit` and prints a number of warnings about non-convergence. The problem is that when you try to estimate **B** and **u**, they are often confounded. This a well-known problem, and you will need to find a way to fix **u** at some value. If you are willing to assume that the process is at equilibrium (i.e. not recovering to equilibrium from a big perturbation), then you can simply demean the data and set **u** to 0. It is also common to standardize the variance

by dividing by the square root of the variance of the data. This is called z-scoring the data.

```
#if missing values are in the data, they should be NAs
z.royale.dat=(royale.dat-apply(royale.dat,1,mean,na.rm=TRUE))/
     sqrt(apply(royale.dat,1,var,na.rm=TRUE))
```

We also need to change a couple settings before fitting the model. In the default MARSS model, the initial value of **x** is treated as being at $t = 0$. If we are estimating the **B** matrix, we need to set this to be at $t = 1$ so that the initial **x** is constrained by the data[3] at $t = 1$. The reason is that we need to estimate the initial **x**. Even if we use a prior on the initial **x**, we are still estimating its value[4]. A model with a non-diagonal **B** matrix, does not "run backwards" well and the estimation of the initial **x** will run in circles. If we constrain it by data (at $t = 1$), we avoid this problem. So we will set `model$tinitx=1`.

The other setting we want to change is `allow.degen`. This sets the diagonals of **Q** or **R** to zero if they are heading towards zero. When the initial **x** is at $t = 1$, this can have non-intuitive (not wrong but puzzling; see Appendix B) consequences if **R** is going to zero. So, we will set `control$allow.degen=FALSE` and manually set **R** to 0 if needed.

```
royale.model.1=list(Z="identity", B="unconstrained",
      Q="diagonal and unequal", R="diagonal and unequal",
      U="zero", tinitx=1)
cntl.list=list(allow.degen=FALSE,maxit=200)
kem.1=MARSS(z.royale.dat, model=royale.model.1, control=cntl.list)
```

```
Warning! Reached maxit before parameters converged. Maxit was 200.
 neither abstol nor log-log convergence tests were passed.

MARSS fit is
Estimation method: kem
Convergence test: conv.test.slope.tol = 0.5, abstol = 0.001
WARNING: maxit reached at  200  iter before convergence.
 Neither abstol nor log-log convergence test were passed.
 The likelihood and params are not at the ML values.
 Try setting control$maxit higher.
Log-likelihood: -76.46247
AIC: 172.9249   AICc: 175.2407
```

[3] If there are many missing values at $t = 1$, we might still have problems and have to adjust accordingly.

[4] Also putting a prior on the initial **x**'s requires specifying their variance-covariance structure, which depends on the unknown **B**, and specifying some variance-covariance structure that conflicts with **B** will change your **B** estimates. So, in general, using a prior on the initial **x**'s when estimating **B** is a bad idea.

```
                 Estimate
R.(Wolf,Wolf)    0.001421
R.(Moose,Moose)  0.000378
B.(1,1)          0.762723
B.(2,1)         -0.173536
B.(1,2)          0.069223
B.(2,2)          0.833074
Q.(Wolf,Wolf)    0.456457
Q.(Moose,Moose)  0.173376
x0.Wolf         -0.267175
x0.Moose        -1.277329

Standard errors have not been calculated.
Use MARSSparamCIs to compute CIs and bias estimates.

Convergence warnings
 Warning: the  R.(Wolf,Wolf)  parameter value has not converged.
 Warning: the  R.(Moose,Moose)  parameter value has not converged.
 Warning: the  logLik  parameter value has not converged.
 Type MARSSinfo("convergence") for more info on this warning.
```

It looks like **R** is going to zero, meaning that the maximum-likelihood model is a process error only model. That is not too surprising given that the data look more like a random walk than white noise. We will set **R** manually to zero:

```
royale.model.2=list(Z="identity", B="unconstrained",
    Q="diagonal and unequal", R="zero", U="zero")
kem.2=MARSS(z.royale.dat, model=royale.model.2)
```

```
Success! algorithm run for 15 iterations. abstol and log-log tests passed.
Alert: conv.test.slope.tol is 0.5.
Test with smaller values (<0.1) to ensure convergence.

MARSS fit is
Estimation method: kem
Convergence test: conv.test.slope.tol = 0.5, abstol = 0.001
Algorithm ran 15 (=minit) iterations and convergence was reached.
Log-likelihood: -82.3988
AIC: 180.7976   AICc: 182.2821

                Estimate
B.(1,1)           0.7618
B.(2,1)          -0.1734
B.(1,2)           0.0692
B.(2,2)           0.8328
```

```
Q.(Wolf,Wolf)     0.4499
Q.(Moose,Moose)   0.1708
x0.Wolf          -0.2086
x0.Moose         -1.5769

Standard errors have not been calculated.
Use MARSSparamCIs to compute CIs and bias estimates.
```

15.2.3 Look at the estimated interactions

The estimated **B** elements are `coef(kem.2)$B`.

```
wolf.B=coef(kem.2,type="matrix")$B
rownames(wolf.B)=colnames(wolf.B)=rownames(royale.dat)
print(wolf.B, digits=2)
```

```
       Wolf Moose
Wolf   0.76 0.069
Moose -0.17 0.833
```

The `coef()` function returns the estimated parameters of the fitted object, but in this case we want to see the estimates in matrix form. Thus we use `type="matrix"`. Element row=i, col=j in **B** is the effect of species j on species i, so $\mathbf{B}_{2,1}$ is the effect of wolves on moose and $\mathbf{B}_{1,2}$ is the effect of moose on wolves. The **B** matrix suggests that wolves have a negative effect on moose and that moose have a positive effect on wolves—as one would expect. The diagonals are interpreted differently than the off-diagonals since the diagonals are $(b_{i,i}-1)$ so subtract off 1 from the diagonals to get the effect of species i on itself. If the species are density-independent, then $\mathbf{B}_{i,i}$ would equal 1. Smaller $\mathbf{B}_{i,i}$ means more density dependence.

15.2.4 Adding covariates

It is well-known that moose numbers are strongly affected by winter and summer climate. The Isle Royale data set provided with MARSS has climate data from climate stations in Northeastern Minnesota, near Isle Royale[5]. The covariate data include January-February, July-September and April-May average temperature and precipitation. Also included are three-year running means of these data, where the number for year x is the average of years x-1, x and x+1. We will include these covariates in the analysis to see how they change our interaction estimates. We have to adjust our covariates because the census numbers are from winter in year x and we want the climate data from the previous year to affect this winter's moose count. As usual, we will

[5] From the Western Regional Climate Center. See the help file for this dataset for references.

need to demean our covariate data so that we can set **u** equal to zero. We will standardize the variance also so that we can more easily compare the effects across different covariates.

The mathematical form of our new process model for the wolf-moose population dynamics is

$$\begin{bmatrix} x_{wolf} \\ x_{moose} \end{bmatrix}_t = \mathbf{B} \begin{bmatrix} x_{wolf} \\ x_{moose} \end{bmatrix}_{t-1} + \begin{bmatrix} 0 & 0 & 0 \\ C_{21} & C_{22} & C_{23} \end{bmatrix} \begin{bmatrix} \text{win temp} \\ \text{win precip} \\ \text{sum temp} \end{bmatrix}_{t-1} + \begin{bmatrix} w_{wolf} \\ w_{moose} \end{bmatrix}_t \quad (15.4)$$

The C_{21}, C_{22}, etc. terms are the effect of winter temperature, winter precipitation, previous summer temperature and previous summer precipitation on winter moose numbers. Since climate is known to mainly affect the moose, we set the climate effects to 0 for wolves (top row of **C**).

First we prepare the covariate data. The winter temperature and precipitation data is in columns 4 and 10, while the summer temperature data is in columns 6. We need to use the previous year's climate data with this winter's abundance data, so we will off-set the climate data.

```
clim.dat= t(isleRoyal[1:52,c(4,10,6)])
z.score.clim.dat=(clim.dat-apply(clim.dat,1,mean,na.rm=TRUE))/
     sqrt(apply(clim.dat,1,var,na.rm=TRUE))
```

A plot of the covariate data against each other indicates that there is not much correlation between winter temperature and precipitation (Figure 15.2, which is good for analysis purposes, but warm winters are somewhat correlated with warm summers. The latter will make it harder to interpret the effect of winter versus summer temperature although the correlation is not too strong fortunately.

Next we prepare the list with the structure of all the model matrices. We give descriptive names to the **C** elements so we can remember what each **C** element means.

```
royale.model.3=list(Z="identity", B="unconstrained",
    Q="diagonal and unequal", R="zero", U="zero",
    C=matrix(list(0,"Moose win temp",0,"Moose win precip",
         0,"Moose sum temp"),2,3),
    c=z.score.clim.dat)

kem.3=MARSS(z.royale.dat[,2:53], model=royale.model.3)
```

```
Success! algorithm run for 15 iterations. abstol and log-log tests passed.
Alert: conv.test.slope.tol is 0.5.
Test with smaller values (<0.1) to ensure convergence.

MARSS fit is
Estimation method: kem
Convergence test: conv.test.slope.tol = 0.5, abstol = 0.001
```

```
Algorithm ran 15 (=minit) iterations and convergence was reached.
Log-likelihood: -80.79155
AIC: 183.5831   AICc: 186.4527

                   Estimate
B.(1,1)              0.7667
B.(2,1)             -0.1609
B.(1,2)              0.0790
B.(2,2)              0.8343
Q.(Wolf,Wolf)        0.4567
Q.(Moose,Moose)      0.1679
x0.Wolf              0.1543
x0.Moose            -1.4008
C.Moose win temp     0.0242
C.Moose win precip  -0.0713
C.Moose sum temp    -0.0306

Standard errors have not been calculated.
Use MARSSparamCIs to compute CIs and bias estimates.
```

Then we fit the model. Because we have to use the climate data from the previous year, we lose a year of our count data.

```
royale.model.3=list(Z="identity", B="unconstrained",
    Q="diagonal and unequal", R="zero", U="zero",
    C=matrix(list(0,"Moose win temp",0,"Moose win precip",
         0,"Moose sum temp"),2,3),
    c=z.score.clim.dat)
```

```
kem.3=MARSS(z.royale.dat[,2:53], model=royale.model.3)
```

```
Success! algorithm run for 15 iterations. abstol and log-log tests passed.
Alert: conv.test.slope.tol is 0.5.
Test with smaller values (<0.1) to ensure convergence.

MARSS fit is
Estimation method: kem
Convergence test: conv.test.slope.tol = 0.5, abstol = 0.001
Algorithm ran 15 (=minit) iterations and convergence was reached.
Log-likelihood: -80.79155
AIC: 183.5831   AICc: 186.4527

                   Estimate
B.(1,1)              0.7667
B.(2,1)             -0.1609
B.(1,2)              0.0790
B.(2,2)              0.8343
```

```
Q.(Wolf,Wolf)        0.4567
Q.(Moose,Moose)      0.1679
x0.Wolf              0.1543
x0.Moose            -1.4008
C.Moose win temp     0.0242
C.Moose win precip  -0.0713
C.Moose sum temp    -0.0306

Standard errors have not been calculated.
Use MARSSparamCIs to compute CIs and bias estimates.
```

The results suggest what is already known about this system: cold winters and lots of snow are bad for moose as are hot summers.

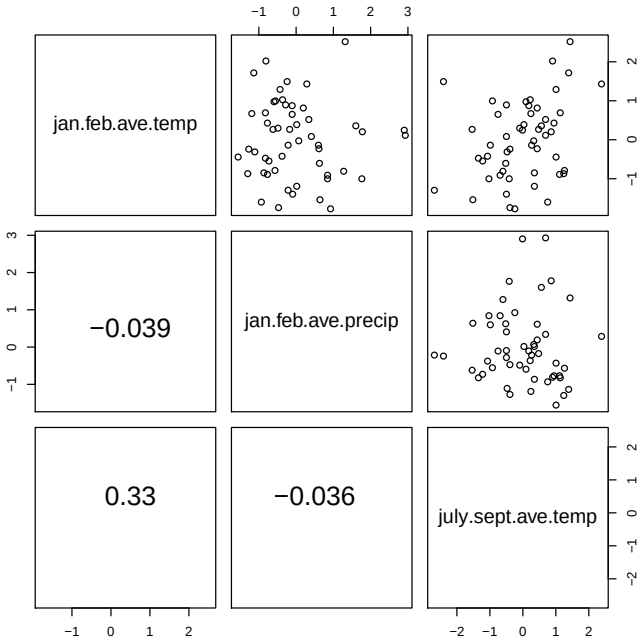

Fig. 15.2. Pairs plot of the covariate data for Isle Royale with correlations in the lower panel. The R code that produced this plot was `cor.fun=function(x, y)text(0.5,0.5,format(cor(x,y),digits=2),cex=2)` `pairs(t(z.score.clim.dat),lower.panel=cor.fun)`.

15.2.5 Change the model and data

You can explore the sensitivity of the **B** estimates when the measurement error is increased by adding white noise to the data:

```
bad.data=z.royale.dat+matrix(rnorm(100,0,sqrt(.2)),2,50)
kem.bad=MARSS(bad.data, model=model)
```

You can change the model by changing the constraints on **R** and **Q**.

15.3 Analysis a four-species plankton community

Ives et al. (2003) presented weekly data on the biomass of two species of phytoplankton and two species of zooplankton in two lakes, one with low planktivory and one with high planktivory. They used these data to estimate the interaction terms for the four species. Here we will reanalyze data and compare our results.

Ives et al. (2003) explain the data as: "The data consist of weekly samples of zooplankton and phytoplankton, which for the analyses were divided into two zooplankton groups (Daphnia and non-Daphnia) and two phytoplankton groups (large and small phytoplankton). Daphnia are large, effective herbivores, and small phytoplankton are particularly vulnerable to herbivory, so we anticipate strong interactions between Daphnia and small phytoplankton groups." Figure 15.3 shows the data. What you can see from the figure is that the data are only collected in the summer.

15.3.1 Load in the plankton data

```
# only use the plankton, daphnia, & non-daphnia
plank.spp = c("Large Phyto","Small Phyto","Daphnia","Non-daphnia")
plank.dat = ivesDataByWeek[,plank.spp]
#The data are not logged
plank.dat = log(plank.dat)
#Transpose to get time going across the columns
plank.dat = t(plank.dat)
#make a demeaned version
d.plank.dat = (plank.dat-apply(plank.dat,1,mean,na.rm=TRUE))
```

As before, we will demean the data so we can set **u** to 0. We do not standardize by the variance, however because we are going to fix the **R** variance later as Ives et al. did.

15.3.2 Specify a MARSS model for the plankton data

We will start by fitting a model with the following assumptions:

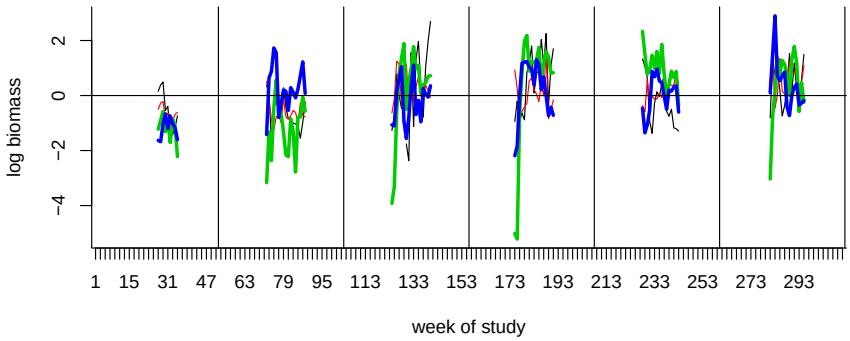

Fig. 15.3. Plot of the de-meaned plankton data. Zooplankton are the thicker lines. Phytoplankton are the thinner lines.

- All phytoplankton share the same process variance.
- All zooplankton share the same process variance.
- Phytoplankton and zooplankton have different measurement variances
- Measurement errors are independent.
- Process errors are independent.

```
Z="identity"
U="zero"
B="unconstrained"
Q=matrix(list(0),4,4); diag(Q)=c("Phyto","Phyto","Zoo","Zoo")
R=matrix(list(0),4,4); diag(R)=c("Phyto","Phyto","Zoo","Zoo")
plank.model.0=list(Z=Z, U=U, Q=Q, R=R, B=B)
```

Why did we set `U="zero"`? Equation 15.1 is a stationary model; it fluctuates about a mean. The **u** in Equation 15.1 is a scaling term that just affects the mean level—once the system is at equilibrium. If we assume that the mean of **y** (the mean of our data) is a good estimate of the mean of the system (the **x**), then we can set **u** equal to zero.

15.3.3 Fit the plankton model and look at the estimated B matrix

The call to fit the model is standard with the addition of setting `model$tinitx` so that the initial states (**x**) are set at $t = 1$ instead of $t = 0$.

```
plank.model.0$tinitx=1
kem.plank.0 = MARSS(d.plank.dat, model=plank.model.0 )
```

Now we can print the **B** matrix, with a little cleaning up so it looks prettier.

```
#Cleaning up the B matrix for printing
B.0 = coef(kem.plank.0, type="matrix")$B[1:4,1:4]
rownames(B.0) = colnames(B.0) = c("LP","SP","D","ND")
print(B.0,digits=2)
```

```
      LP   SP       D     ND
LP  0.77 0.29 -0.0182  0.131
SP  0.19 0.51  0.0052 -0.045
D  -0.43 2.29  0.4916  0.389
ND -0.33 1.35 -0.2180  0.831
```

LP stands for large phytoplankton, SP for small phytoplankton, D for Daphnia and ND for non-Daphnia.

We can compare this to the Ives et al. estimates (in their Table 2, bottom right) and see a few differences:

```
     LP    SP     D    ND
LP 0.48 -0.39    --    --
SP   --  0.25 -0.17 -0.11
D    --    --  0.74  0.00
ND   --  0.10  0.00  0.60
```

First, thing you will notice is that the Ives et al. matrix is missing values. The matrix they show is after a model selection step to determine which interactions had little data support and thus could be set to zero. Also, they fixed apriori the interactions between Daphnia and non-Daphnia at zero because they do not prey on each other. The second thing you will notice is that the estimates are not particularly similar. Next we will try some other ways of fitting the data that are closer to the way that Ives et al. fitted the data.

By the way, if you are curious what would happen if we removed all those NAs, you can run the following code.

```
test.dat = d.plank.dat[, !is.na(d.plank.dat[1, ])]
test = MARSS(test.dat, model = plank.model.0)
```

Removing all the NAs would mean that the end of summer 1 is connected to the beginning of summer 2. This adds some steep steps in the Daphnia time series where Daphnia ended the summer high and started the next summer low.

15.3.4 Look at different ways to fit the model

We will try a series of changes to get closer to the way Ives et al. fit the data, and you will see how different assumptions change (or do not change) our species interaction estimates.

First, we change **Q** to be unconstrained. Making **Q** diagonal in model 0 meant that we were assuming that whatever environmental factor is driving variation in phytoplankton numbers is uncorrelated with the environmental

factor driving zooplankton variation. That is probably not true since they are all in the same lake. This case takes awhile to run.

```
plank.model.1=plank.model.0
plank.model.1$Q="unconstrained"
kem.plank.1 = MARSS(d.plank.dat, model=plank.model.1)
```

Notice that the **Q** specification changed to "unconstrained". Everything else stays the same as in model 0. The code now runs longer, and the **B** estimates are not particulaly closer to Ives et al.

```
        LP    SP      D    ND
LP  0.4961 0.061  0.079 0.123
SP -0.1833 0.896  0.067 0.011
D   0.1180 0.350  0.638 0.370
ND  0.0023 0.370 -0.122 0.810
```

Next, we will set some of the interactions to zero as in Table 2 in Ives et al. (2003). In that table, certain interactions were fixed at 0 (denoted with 0s), and some were made 0 after fitting (the blanks). We will fix all to zero. To do this, we need to write out the **B** matrix as a list matrix so that we can have estimated and fixed values (the 0s) in the **B** specification.

```
B.2=matrix(list(0),4,4) #set up the list matrix
diag(B.2)=c("B11","B22","B33","B44") #give names to diagonals
#and names to the estimated non-diagonals
B.2[1,2]="B12"; B.2[2,3]="B23"; B.2[2,4]="B24"; B.2[4,2]="B42"
print(B.2)
```

```
     [,1]  [,2]  [,3]  [,4]
[1,] "B11" "B12" 0     0
[2,] 0     "B22" "B23" "B24"
[3,] 0     0     "B33" 0
[4,] 0     "B42" 0     "B44"
```

As you can see, the **B** matrix now has elements that will be estimated (the names in quotes) and fixed values (the numbers with no quotes). When preparing your list matrix, make sure your fixed values do not have have quotes around them. If they do, they are strings (class character) not numbers (class numeric), and MARSS will interpret a string as the name of something to be estimated. If you use the same name for an element, then MARSS will force those elements to be shared (have the same value). This one will take a while to run also.

```
#model 2
plank.model.2=plank.model.1
plank.model.2$B = B.2
kem.plank.2= MARSS(d.plank.dat, model=plank.model.2)
```

Now we are getting closer to the Ives et al. estimates:

```
     LP    SP      D     ND
LP 0.65 -0.33     --     --
SP   --  0.54 0.0016 -0.026
D    --    -- 0.8349     --
ND   --  0.13     --  0.596
```

Ives et al. did not estimate **R**. Instead they used a fixed observation variance of 0.04 for phytoplankton and 0.16 for zooplankton[6]. We fit the model with their fixed **R** as follows:

```
#model 3
plank.model.3=plank.model.2
plank.model.3$R=diag(c(.04,.04,.16,.16))
kem.plank.3= MARSS(d.plank.dat, model=plank.model.3)
```

As you can see from Table 15.1, we are getting closer to Ives et al. estimates, but we are still a bit off. Now we need to add the environmental covariates: phosphorous and fish biomass.

15.3.5 Adding covariates

A standard way that you will see covariate data added to a MARSS model is the following:

$$\begin{aligned} \mathbf{x}_t &= \mathbf{B}\mathbf{x}_{t-1} + \mathbf{u} + \mathbf{C}\mathbf{c}_t + \mathbf{w}_t, \text{ where } \mathbf{w}_t \sim \text{MVN}(0, \mathbf{Q}) \\ \mathbf{y}_t &= \mathbf{Z}\mathbf{x}_t + \mathbf{a} + \mathbf{D}\mathbf{d}_t + \mathbf{v}_t, \text{ where } \mathbf{v}_t \sim \text{MVN}(0, \mathbf{R}) \end{aligned} \tag{15.5}$$

$\mathbf{c}_t$ and $\mathbf{d}_t$ are covariate data, like temperature. At time t and $\mathbf{C}$ is a matrix with the (linear) effects of $\mathbf{c}_t$ on $\mathbf{x}_t$, and $\mathbf{D}$ is a matrix with the (linear) effects of $\mathbf{d}_t$ on $\mathbf{y}_t$.

Ives et al. (2003) only include covariates in their process model, and their process model (their Equation 27) is written $\mathbf{X}_t = \mathbf{A} + \mathbf{B}\mathbf{X}_{t-1} + \mathbf{C}\mathbf{U}_t + \mathbf{E}_t$. In our Equation 15.5, $\mathbf{U}_t = \mathbf{c}_t$, and $\mathbf{C}$ is a $m \times p$ matrix, where p is the number of covariates in $\mathbf{c}_t$. For this case study, we set their $\mathbf{A}$ (our $\mathbf{u}$) to zero by demeaning the $\mathbf{y}$ and implicitly assuming that the mean of the $\mathbf{y}$ is a good estimate of the mean of the $\mathbf{x}$'s. Thus the model where covariates only affect the underlying process is

$$\begin{aligned} \mathbf{x}_t &= \mathbf{B}\mathbf{x}_{t-1} + \mathbf{C}\mathbf{c}_t + \mathbf{w}_t, \text{ where } \mathbf{w}_t \sim \text{MVN}(0, \mathbf{Q}) \\ \mathbf{y}_t &= \mathbf{x}_t + \mathbf{v}_t, \text{ where } \mathbf{v}_t \sim \text{MVN}(0, \mathbf{R}) \end{aligned} \tag{15.6}$$

To fit this model, we first need to prepare the covariate data. We will just use the phosphorous data.

[6] You can compare this to the estimated observation variances by looking at `coef(kem.plank.2)$R`

```
#transpose to make time go across columns
#drop=FALSE so that R doesn't change our matrix to a vector
phos = t(log(ivesDataByWeek[,"Phosph",drop=FALSE]))
d.phos = (phos-apply(phos,1,mean,na.rm=TRUE))
```

Why log the covariate data? It is what Ives et al. did, so we follow their method. However, in general, you want to think about what relationship you want to assume between the covariates and their effects. For example, log (or square-root) transformations mean that extremes have less impact relative to their untransformed value and that a small absolute change, say from 0.01 to 0.0001 in the untransformed value, can mean large difference in the effects since $\log(0.0001) < \log(0.01)$.

Phosporous is assumed to only affect phytoplankton so the other terms in **C**, corresponding to the zooplankton, are set to 0. The **C** matrix is defined as follows:

$$\mathbf{C} = \begin{bmatrix} C_{LP,phos} \\ C_{SP,phos} \\ 0 \\ 0 \end{bmatrix} \tag{15.7}$$

To add **C** and **c** to our latest model, we add **C** and **c** to the model list used in the MARSS call:

```
plank.model.4=plank.model.3
plank.model.4$C=matrix(list("C11","C21",0,0),4,1)
plank.model.4$c=d.phos
```

Then we fit the model as usual:

```
kem.plank.4= MARSS(d.plank.dat, model=plank.model.4)
```

```
Success! abstol and log-log tests passed at 55 iterations.
Alert: conv.test.slope.tol is 0.5.
Test with smaller values (<0.1) to ensure convergence.

MARSS fit is
Estimation method: kem
Convergence test: conv.test.slope.tol = 0.5, abstol = 0.001
Estimation converged in 55 iterations.
Log-likelihood: -393.189
AIC: 834.3781   AICc: 837.9284

               Estimate
B.B11            0.6138
B.B12           -0.4619
B.B22            0.3320
B.B42            0.0479
B.B23           -0.0182
```

```
B.B33            0.8889
B.B24           -0.0476
B.B44            0.6643
Q.(1,1)          0.7376
Q.(2,1)          0.2159
Q.(3,1)          0.0796
Q.(4,1)          0.0293
Q.(2,2)          0.2688
Q.(3,2)         -0.1271
Q.(4,2)         -0.0878
Q.(3,3)          0.8654
Q.(4,3)          0.4685
Q.(4,4)          0.3906
x0.Large Phyto   0.1615
x0.Small Phyto  -0.5273
x0.Daphnia      -1.1121
x0.Non-daphnia  -1.8082
C.C11            0.1385
C.C21            0.1580

Standard errors have not been calculated.
Use MARSSparamCIs to compute CIs and bias estimates.
```

15.3.6 Including a covariate observation model

The difficulty with the standard approach to including covariates (Equation 15.5) is that it limits what kind of covariate data you can use and how you model that covariate data. You have to assume that your covariate data has no error, which is probably not true. Assuming that your covariate has no error reduces the reported uncertainty in your covariate effect because you did not include uncertainty in those values. The standard approach also does not allow missing values in your covariate data, which is why we did not include the fish covariate data in the last model. Also you cannot combine instrument time series; for example, if you have two temperature recorders with different error rates and biases. Also, what if you have one noisy temperature recorder in the first part of your time series and then you switch to a much better recorder in the second half of your time series? All these problems require pre-analysis massaging of the covariate data, leaving out noisy and gappy covariate data, and making what can feel like arbitrary choices about which covariate time series to include which is especially worrisome when the covariates are then incorporated in the model as known without error.

Instead one can include an observation and process model for the covariates just like for the non-covariate data. Now the covariates are included in $\mathbf{y}_t$ and are modeled with their own state process(es) in $\mathbf{x}_t$. A MARSS model with a covariate observation and process model is shown below. The elements

with superscript (v) are for the variates and those with superscript (c) are for the covariates. The superscripts just help us keep straight which of the state processes and parameters corresponding to the parts that correspond abundances and which correspond to the environmental covariates.

$$\begin{bmatrix}\mathbf{x}^{(v)}\\ \mathbf{x}^{(c)}\end{bmatrix}_t = \begin{bmatrix}\mathbf{B}^{(v)} & \mathbf{C}\\ 0 & \mathbf{B}^{(c)}\end{bmatrix}\begin{bmatrix}\mathbf{x}^{(v)}\\ \mathbf{x}^{(c)}\end{bmatrix}_{t-1} + \begin{bmatrix}\mathbf{u}^{(v)}\\ \mathbf{u}^{(c)}\end{bmatrix} + \mathbf{w}_t,\ \mathbf{w}_t \sim \text{MVN}\left(0, \begin{bmatrix}\mathbf{Q}^{(v)} & 0\\ 0 & \mathbf{Q}^{(c)}\end{bmatrix}\right)$$

$$\begin{bmatrix}\mathbf{y}^{(v)}\\ \mathbf{y}^{(c)}\end{bmatrix}_t = \begin{bmatrix}\mathbf{Z}^{(v)} & 0\\ 0 & \mathbf{Z}^{(c)}\end{bmatrix}\begin{bmatrix}\mathbf{x}^{(v)}\\ \mathbf{x}^{(c)}\end{bmatrix}_t + \begin{bmatrix}\mathbf{a}^{(v)}\\ \mathbf{a}^{(c)}\end{bmatrix} + \mathbf{v}_t,\ \mathbf{v}_t \sim \text{MVN}\left(0, \begin{bmatrix}\mathbf{R}^{(v)} & 0\\ 0 & \mathbf{R}^{(c)}\end{bmatrix}\right) \tag{15.8}$$

Note that when you fit your covariate and non-covariate data jointly as in Equation 15.8, your non-covariate data affect the estimates of the covariate models. When you maximize the likelihood, you do so conditioned on all the data. The likelihood that is output is the likelihood of the non-covariate and covariate data. Depending on your system, you might not want the covariate model affected by the non-covariate data. In this case, you can fit the covariate model separately:

$$\begin{aligned}\mathbf{x}_t^{(c)} &= \mathbf{B}^{(c)}\mathbf{x}_{t-1}^{(c)} + \mathbf{u}^{(c)} + \mathbf{w}_t,\ \mathbf{w}_t \sim \text{MVN}(0, \mathbf{Q}^{(c)})\\ \mathbf{y}_t^{(c)} &= \mathbf{Z}^{(c)}\mathbf{x}_t^{(c)} + \mathbf{a}^{(c)},\ \mathbf{v}_t \sim \text{MVN}(0, \mathbf{R}^{(c)})\end{aligned} \tag{15.9}$$

At this point, you have another choice. Do you want the estimated covariates states, the $\mathbf{x}^{(c)}$, to be affected by the non-covariate data? For example, you have temperature data. You can estimates true temperature for the temperature only from the temperature data or you can decide that the non-covariate data has information about the true temperature, because the non-covariate states are affected by the true temperature. If you want the covariate states to only be affected by the covariate data, then use Equation 15.5 with $\mathbf{u}_t$ set from your estimates of $\mathbf{x}^{(c)}$ from Equation 15.9. Or if you want the non-covariate data to affect the estimates of the covariate states, use Equation 15.8 with the parameters estimated from Equation 15.9.

15.3.7 The MARSS model with covariates following Ives et al.

Ives et al. used Equation 15.5 for phosphorous and Equation 15.8 for fish biomass. Phosphorous was treated as observed with no error since it was experimentally manipulated and there were no missing values. Fish biomass was treated as having observation error and was modeled as a autoregressive process with unknown parameters as in Equation 15.8.

Their MARSS model takes the form:

$$\begin{aligned}\mathbf{x}_t &= \mathbf{B}\mathbf{x}_{t-1} + \mathbf{C}\mathbf{c}_t + \mathbf{w}_t,\ \text{where } \mathbf{w}_t \sim \text{MVN}(0, \mathbf{Q})\\ \mathbf{y}_t &= \mathbf{x}_t + \mathbf{v}_t,\ \text{where } \mathbf{v}_t \sim \text{MVN}(0, \mathbf{R})\end{aligned} \tag{15.10}$$

where **x** and **y** are redefined as

$$\begin{bmatrix} \text{large phyto} \\ \text{small phyto} \\ \text{Daphnia} \\ \text{Non-Daphnia zooplank} \\ \text{fish biomass} \end{bmatrix} \tag{15.11}$$

The covariate fish biomass appears in **x** because it will be modeled, and its interaction terms (Ives et al.'s **C** terms) appear in **B**. Phosphorous appears in the $\mathbf{c}_t$ terms because it is treated as a known additive term and its interaction terms appear in **C**. Recall that we set **u** to 0 by demeaning the plankton data, so it does not appear above. The **Z** matrix does not appear in front of the $\mathbf{x}_t$ since there is a one-to-one correspondence the **x**'s and **y**'s, and thus **Z** is the identity matrix.

The **B** matrix is

$$\mathbf{B} = \begin{bmatrix} \mathbf{B}^{(v)} & \mathbf{C} \\ 0 & \mathbf{B}^{(c)} \end{bmatrix} = \begin{bmatrix} b_{LP} & b_{LP,SP} & 0 & 0 & 0 \\ 0 & b_{SP} & b_{SP,D} & b_{SP,ND} & 0 \\ 0 & 0 & b_D & 0 & C_{D,fish} \\ 0 & b_{ND,SP} & 0 & b_{ND,ND} & C_{ND,fish} \\ 0 & 0 & 0 & 0 & b_{fish} \end{bmatrix} \tag{15.12}$$

The **B** elements have some interactions fixed at 0 as in our last model fit. The c's are the interactions between the fish and the species. We will estimate a **B** term for fish since Ives et al. did, but this is an odd thing to do for the fish data since these data were interpolated from two samples per season.

The **Q** matrix is the same as that in our last model fit, with the addition of an element for the variance for the fish biomass:

$$\mathbf{Q} = \begin{bmatrix} \mathbf{Q}^{(v)} & 0 \\ 0 & \mathbf{Q}^{(c)} \end{bmatrix} = \begin{bmatrix} q_{LP} & q_{LP,SP} & q_{LP,D} & q_{LP,ND} & 0 \\ q_{LP,SP} & q_{SP} & q_{SP,D} & q_{SP,ND} & 0 \\ q_{LP,D} & q_{SP,D} & q_D & q_{D,ND} & 0 \\ q_{LP,ND} & q_{SP,ND} & q_{D,ND} & q_{ND} & 0 \\ 0 & 0 & 0 & 0 & q_{fish} \end{bmatrix} \tag{15.13}$$

Again it is odd to estimate a variance term for data interpolated from two points, but we follow Ives et al. here.

Ives et al. set the observation variance for the logged fish biomass data to 0.36 (page 320 in Ives et al. (2003)). The observation variances for the plankton data was set as in our previous model.

$$\mathbf{R} = \begin{bmatrix} 0.04 & 0 & 0 & 0 & 0 \\ 0 & 0.04 & 0 & 0 & 0 \\ 0 & 0 & 0.16 & 0 & 0 \\ 0 & 0 & 0 & 0.16 & 0 \\ 0 & 0 & 0 & 0 & 0.36 \end{bmatrix} \tag{15.14}$$

15.3.8 Setting the model structure for the model with fish covariate data

First we need to add the logged fish biomass to our data matrix.

```
#transpose to make time go across columns
#drop=FALSE so that R doesn't change our matrix to a vector
fish = t(log(ivesDataByWeek[,"Fish biomass",drop=FALSE]))
d.fish = (fish-apply(fish,1,mean,na.rm=TRUE))
#plank.dat.w.fish = rbind(plank.dat,fish)
d.plank.dat.w.fish = rbind(d.plank.dat,d.fish)
```

Next make the **B** matrix. Some elements are estimated and others are fixed at 0.

```
B=matrix(list(0),5,5)
diag(B)=list("B11","B22","B33","B44","Bfish")
B[1,2]="B12";B[2,3]="B23"; B[2,4]="B24"
B[4,2]="B42";
B[1:4,5]=list(0,0,"C32","C42")
print(B)
```

```
     [,1]  [,2]  [,3]  [,4]  [,5]
[1,] "B11" "B12" 0     0     0
[2,] 0     "B22" "B23" "B24" 0
[3,] 0     0     "B33" 0     "C32"
[4,] 0     "B42" 0     "B44" "C42"
[5,] 0     0     0     0     "Bfish"
```

Now we have a **B** matrix that looks like that in Equation 15.12.

We need to add an extra row to **C** for the fish biomass row in **x**:

```
C=matrix(list("C11","C21",0,0,0),5,1)
```

Then we set up the **R** matrix.

```
R=matrix(list(0),5,5)
diag(R)=list(0.04,0.04,0.16,0.16,0.36)
```

Last, we need to set up the **Q** matrix:

```
Q=matrix(list(0),5,5);
Q[1:4,1:4]=paste(rep(1:4,times=4),rep(1:4,each=4),sep="")
Q[5,5]="fish"
Q[lower.tri(Q)]=t(Q)[lower.tri(Q)]
print(Q)
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,] "11" "12" "13" "14" 0
[2,] "12" "22" "23" "24" 0
```

```
[3,] "13" "23" "33" "34" 0
[4,] "14" "24" "34" "44" 0
[5,] 0    0    0    0    "fish"
```

15.3.9 Fit the model with covariates

The model is the same as the previous model with updated process parameters and updated **R**. We will pass in the updated data matrix with the fish biomass added:

```
plank.model.5=plank.model.4
plank.model.5$B=B
plank.model.5$C=C
plank.model.5$Q=Q
plank.model.5$R=R
kem.plank.5=MARSS(d.plank.dat.w.fish, model=plank.model.5)
```

This is the new **B** matrix using covariates.

```
     LP     SP      D     ND
LP 0.61 -0.465     --     --
SP   --  0.333 -0.019 -0.048
D    --     --  0.896     --
ND   --  0.044     --  0.675
```

Now we are getting are getting close to Ives et al.'s estimates. Compare model 5 in Table 15.1 to the first column.

Table 15.1. The parameter estimates under the different plankton models. Models 0 to 3 do not include covariates, so the C elements are blank. Bij is the effect of species *i* on species *j*. 1=large phytoplankton, 2=small phytoplankton, 3=Daphnia, 4=non-Daphnia zooplankton. The Ives et al. (2003) estimates are from their table 2 for the low planktivory lake with the observation model.

	Ives et al.	Model 0	Model 1	Model 2	Model 3	Model 4	Model 5
B11	0.48	0.77	0.50	0.65	0.62	0.61	0.61
B22	0.25	0.51	0.90	0.54	0.51	0.33	0.33
B33	0.74	0.49	0.64	0.83	0.89	0.89	0.90
B44	0.60	0.83	0.81	0.60	0.67	0.66	0.67
B12	-0.39	0.29	0.06	-0.33	-0.32	-0.46	-0.46
B23	-0.17	0.01	0.07	0.00	-0.02	-0.02	-0.02
B24	-0.11	-0.04	0.01	-0.03	0.02	-0.05	-0.05
B42	0.10	1.35	0.37	0.13	0.09	0.05	0.04
C11	0.25					0.14	0.14
C21	0.25					0.16	0.16
C32	-0.14						-0.04
C42	-0.04						-0.01

NOTE! When you include your covariates in your state model (the **x** part), the reported log-likelihood is for the variate plus the covariate data. If you want just the log-likelihood for the variates, then your replace the covariate data with NAs and run the Kalman filter with your estimated model:

```
tmp=kem.plank.5
tmp$marss$data[5,]=NA
LL.variates=MARSSkf(tmp)$logLik
```

`MARSSkf` is the Kalman filter function and it needs a fitted model as output by a `MARSS` call. We set up a temporary fitted model, `tmp`, equal to our fitted model and then set the covariate data in that to NAs. Note we need to do this for the marssMODEL object used by MARSSkf, which will be in `$marss`. We then pass that temporary fitted model to `MARSSkf` to get the log-likelihood of just the variates.

15.3.10 Discussion

The estimates for model 4 are fairly close to the Ives et al. estimates, but still a bit different. There are two big difference between model 4 and the Ives et al. analysis. Ives et al. had data from three lakes and the estimate of **Q** used the data from all lakes.

Combining data, whether it be from different areas or years, can be done in a MARSS model as follows. Let $\mathbf{y}_1$ be the first data set (say from site 1) and $\mathbf{y}_2$ be the second data set (say from site 2). Then a MARSS model with shared parameters values across datasets would be

$$\begin{aligned} \mathbf{x}_t^+ &= \mathbf{B}^+\mathbf{x}_{t-1}^+ + \mathbf{u}^+\mathbf{w}_t, \text{ where } \mathbf{w}_t \sim \text{MVN}(0, \mathbf{Q}^+) \\ \mathbf{y}_t^+ &= \mathbf{Z}^+\mathbf{x}_t^+ + \mathbf{a}^+ + \mathbf{v}_t, \text{ where } \mathbf{v}_t \sim \text{MVN}(0, \mathbf{R}^+) \end{aligned} \tag{15.15}$$

where the + matrices are stacked matrices from the different sites (1 and 2):

$$\begin{aligned} \begin{bmatrix} \mathbf{x}_{1,t} \\ \mathbf{x}_{2,t} \end{bmatrix} &= \begin{bmatrix} \mathbf{B} & 0 \\ 0 & \mathbf{B} \end{bmatrix} \begin{bmatrix} \mathbf{x}_{1,t-1} \\ \mathbf{x}_{2,t-1} \end{bmatrix} + \begin{bmatrix} \mathbf{u} \\ \mathbf{u} \end{bmatrix} + \mathbf{w}_t, \ \mathbf{w}_t \sim \text{MVN}\left(0, \begin{bmatrix} \mathbf{Q} & \mathbf{q} \\ \mathbf{q} & \mathbf{Q} \end{bmatrix}\right) \\ \begin{bmatrix} \mathbf{y}_{1,t} \\ \mathbf{y}_{2,t} \end{bmatrix} &= \begin{bmatrix} \mathbf{Z} & 0 \\ 0 & \mathbf{Z} \end{bmatrix} \begin{bmatrix} \mathbf{x}_{1,t} \\ \mathbf{x}_{2,t} \end{bmatrix} + \begin{bmatrix} \mathbf{a} \\ \mathbf{a} \end{bmatrix} + \mathbf{v}_t, \ \mathbf{v}_t \sim \text{MVN}\left(0, \begin{bmatrix} \mathbf{R} & 0 \\ 0 & \mathbf{R} \end{bmatrix}\right) \end{aligned} \tag{15.16}$$

The **q** in the process variance allows that the environmental variability might might be correlated between datasets, i.e. if they are replicate plots that are nearby, say. If you did not want all the parameters shared, then you replace the **B** in $\mathbf{B}^+$ with $\mathbf{B}_1$ and $\mathbf{B}_2$, say.

The second big difference is that Ives et al. did not demean their data, but estimated **u**. We could have done that too, but with all the NAs in the data (during winter), estimating **u** is not robust and takes a long time. You can try the analysis on the data that has not been demeaned and set `U="unequal"`.

The results are not particularly different, but it takes a long, long,...long time to converge.

You can also try using the actual fish data instead of the interpolated data. Fish biomass was estimated at the end and start of the season, so only the values at the start and finish of strings of fish numbers are the real data. The others are interpolated. You can fill in those interpolated values with NAs (missing values) and rerun model 4. The results are not appreciably different, but the effect of fish drops a bit as you might expect when you have less fish information. You don't see it here, but your estimated confidence in the fish effects would also drop since this estimate is based on less fish data.

16

Case Study 8: Combining data from multiple time series

16.1 Overview

In this section, we consider the case where multiple time series exist and we want to use all the datasets to estimate a common underlying state-process or common underlying parameters. In ecological applications, this situation arises when 1) They are time series of observations from the same species as the original time series (e.g. aerial and land based surveys of the same same species) or 2) They are time series collected in the same survey, but represent observations of multiple species (e.g. we might be doing a fisheries trawl survey that collects multiple species in each trawl). Why should we consider using other time series? In the first scenario, where methodology differs between time series of the same species, observation error may be survey-specific. These time series may represent observations of multiple populations, or these may represent multiple observations of the same population. In the second scenario, each species should be treated as a separate process (given its own state vector), but because the survey methodology is the same across species, it might be reasonable to assume a shared observation error variance. If these species have a similar response to environmental stochasticity, it might be possible to also assume a shared process variance.

In both of the above examples, MARSS models offer a way to linking multiple time series. If parameters are allowed to be shared among the state processes (trend parameters, process variances) or observation processes (observation variances), parameter estimates will be more precise than if we treated each time series as independent. By improving estimates of variance parameters, we will also be better able to discriminate between process and observation error variances.

In this case study, we will show examples of using MARSS to analyze multiple time series on the same species but either different populations or different survey methods. The multivariate first-order autoregressive state process is written as usual as:

$$\mathbf{x}_t = \mathbf{B}\mathbf{x}_{t-1} + \mathbf{u} + \mathbf{w}_t \text{ where } \mathbf{w}_t \sim \text{MVN}(0, \mathbf{Q}) \tag{16.1}$$

The true population sizes at time t are represented by the state $\mathbf{x}_t$, whose dimensions are equal to the number of state processes (m). The $m \times m$ matrix $\mathbf{B}$ allows interaction between processes (density dependence and competition, for instance), $\mathbf{u}$ is a vector describing the mean trend, and the correlation of the process deviations is determined by the structure of the matrix $\mathbf{Q}$.

The multivariate observation error model is expressed as,

$$\mathbf{y}_t = \mathbf{Z}\mathbf{x}_t + \mathbf{a} + \mathbf{v}_t \text{ where } \mathbf{v}_t \sim \text{MVN}(0, \mathbf{R}) \tag{16.2}$$

where $\mathbf{y}_t$ is a vector of observations at time t, $\mathbf{Z}$ is a design matrix of 0s and 1s, $\mathbf{a}$ is a vector of bias adjustments, and the correlation structure of observation matrices is specified with the matrix $\mathbf{R}$. Including $\mathbf{Z}$ and $\mathbf{a}$ will not be required for every model, but is useful when some processes are observed multiple times.

16.2 Salmon spawner surveys

In our first application combining multiple time series, we will analyze a dataset on Chinook salmon (*Oncorhynchus tshawytscha*). This data comes from the Okanogan River in Washington state, a major tributary of the Columbia River (headwaters in British Columbia). As an index of the abundance of spawning adults, biologists have conducted redd surveys during summer months (redds are nests or collection of rocks on stream bottoms where females deposit eggs). Aerial surveys of redds on the Okanogan have been conducted 1956-2008. Alternative ground surveys of were initiated in 1990, and have been conducted annually.

16.2.1 Read in and plot the raw data

We will be using the logged counts.

```
head(okanaganRedds)
```

```
     Year aerial ground
[1,] 1956     37     NA
[2,] 1957     53     NA
[3,] 1958     94     NA
[4,] 1959     50     NA
[5,] 1960     29     NA
[6,] 1961     NA     NA
```

```
logRedds = log(t(okanaganRedds)[2:3,])
```

Year is in the first column, and the counts (in normal space) are in columns 2:3. Missing observations are represented by NAs.

```
# Code for plotting raw Okanagan redd counts
plot(okanaganRedds[,1], okanaganRedds[,2],
    xlab = "Year", ylab="Redd counts",main="", col="red", pch=1)
points(okanaganRedds[,1], okanaganRedds[,3], col="blue", pch=2)
legend('topleft', inset=0.1, legend=c("Aerial survey","Ground survey"),
 col=c("red","blue"), pch=c(1,2))
```

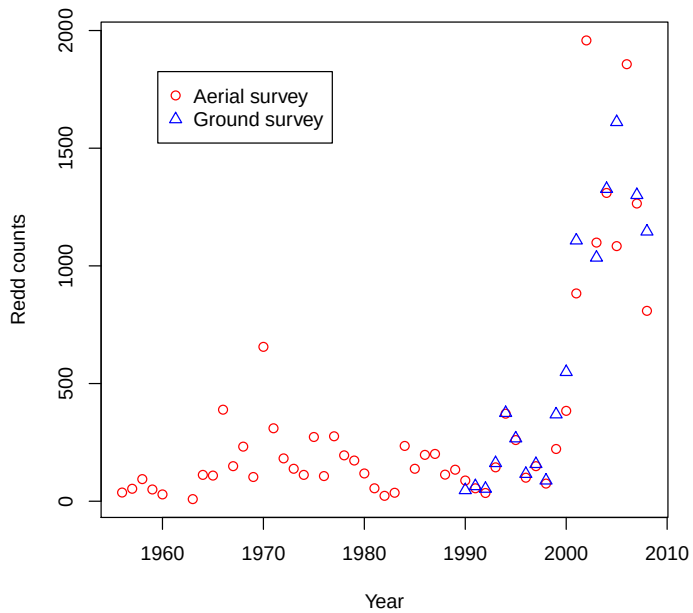

Fig. 16.1. The two time series look to be pretty close to one another in the years where there is overlap.

16.2.2 Test hypotheses about whether the data can be combined

Do these surveys represent observations of the same underlying process? We can evaluate data support for this question by testing a few relatively simple models. Using the logged data, we will start with a simple model that assumes the underlying population process is univariate (there is one underlying population trajectory) and each survey is an independent observation of this population process. Mathematically, the model is:

$$\begin{gathered} x_t = x_{t-1} + u + w_t, \text{ where } w_t \sim \mathrm{N}(0,q) \\ \begin{bmatrix} y_{aer} \\ y_{gnd} \end{bmatrix}_t = \begin{bmatrix} 1 \\ 1 \end{bmatrix} x_t + \begin{bmatrix} 0 \\ a_2 \end{bmatrix} + \begin{bmatrix} v_{aer} \\ v_{gnd} \end{bmatrix}_t, \text{ where } \mathbf{v}_t \sim \mathrm{MVN}\left((0, \begin{bmatrix} r & 0 \\ 0 & r \end{bmatrix}\right) \end{gathered} \tag{16.3}$$

The **a** structure means that the a for one of the y's is fixed at 0 and the other a is estimated relative to that fixed a. In MARSS, this is the "scaling" structure for **a**. We specify this model in MARSS as follows. Since **x** is univariate, **Q** and **u** are just scalars (single numbers) and have no structure (like 'diagonal'). Thus we can leave them off in our specification.

```
model1=list()
model1$R="diagonal and equal"
model1$Z=matrix(1,2,1) #matrix of 2 rows and 1 column
model1$A="scaling" #the default
# Fit the single state model, where the time series are assumed
# to be from thesame population.
kem1 = MARSS(logRedds, model=model1)
```

We can print the AIC and AICc values for this model by typing `kem1$AIC` and `kem1$AICc`.

How would we modify the above model to let the observation error variances to be unique? We can do this in our second model:

```
model2=model1 #model2 is based on model1
model2$R="diagonal and unequal"
kem2 = MARSS(logRedds, model=model2)
```

It is possible that these surveys are measuring different population processes, so for our third model, we will fit a model with two different population process with the same process parameters. For simplicity, we will keep the trend and variance parameters the same. Mathematically, the model we are fitting is:

$$\begin{aligned}\begin{bmatrix}x_1\\x_2\end{bmatrix}_t &= \begin{bmatrix}x_1\\x_2\end{bmatrix}_{t-1} + \begin{bmatrix}u\\u\end{bmatrix} + \mathbf{w}_t, \text{ where } \mathbf{w}_t \sim \text{MVN}(0, \begin{bmatrix}q & 0\\0 & q\end{bmatrix})\\ \begin{bmatrix}y_{aer}\\y_{gnd}\end{bmatrix}_t &= \begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}\begin{bmatrix}x_1\\x_2\end{bmatrix}_t + \begin{bmatrix}0\\0\end{bmatrix} + \begin{bmatrix}v_{aer}\\v_{gnd}\end{bmatrix}_t, \text{ where } \mathbf{v}_t \sim \text{MVN}\left(0, \begin{bmatrix}r & 0\\0 & r\end{bmatrix}\right)\end{aligned} \tag{16.4}$$

We specify this in MARSS as

```
model3=list()
model3$Q="diagonal and equal"
model3$R="diagonal and equal"
model3$U="equal"
model3$Z="identity"
model3$A="zero"
kem3 = MARSS(logRedds, model=model3)
```

Based on AIC, it looks like the best model is also the simplest one, with one state vector (model1). This suggests that the two different surveys are not only measuring the same thing, but have the same observation error variance. Finally,we will make a plot of the model-predicted states (with +/- 2 s.e.s) and the log-transformed data (Figure 16.2).

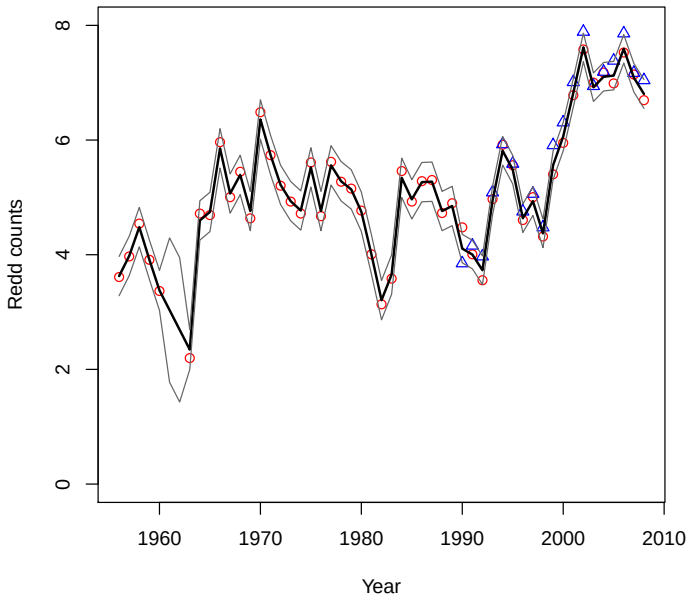

Fig. 16.2. The data support the hypothesis that the two redd-count time series are observations of the same population. The points are the data and the thick black line is the estimated underlying state.

16.3 American kestrel abundance indices

In this example, we evaluate uncertainty in the structure of process variability (environmental stochasticity) using some bird survey data. Breeding Bird Surveys have been conducted in the U.S. and Canada for 50+ years. In this analysis, we focus on 3 time series of American kestrel (*Falco sparverius*) 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.

16.3.1 Read in and look at the data

Figure 16.3 shows the data. The data are already log transformed.

```
birddat = t(kestrel[,2:4])
head(kestrel)
```

```
     Year British.Columbia Alberta Saskatchewan
[1,] 1969            0.754   0.460        0.000
[2,] 1970            0.673   0.899        0.192
[3,] 1971            0.734   1.133        0.280
[4,] 1972            0.589   0.528        0.386
[5,] 1973            1.405   0.789        0.451
[6,] 1974            0.624   0.528        0.234
```

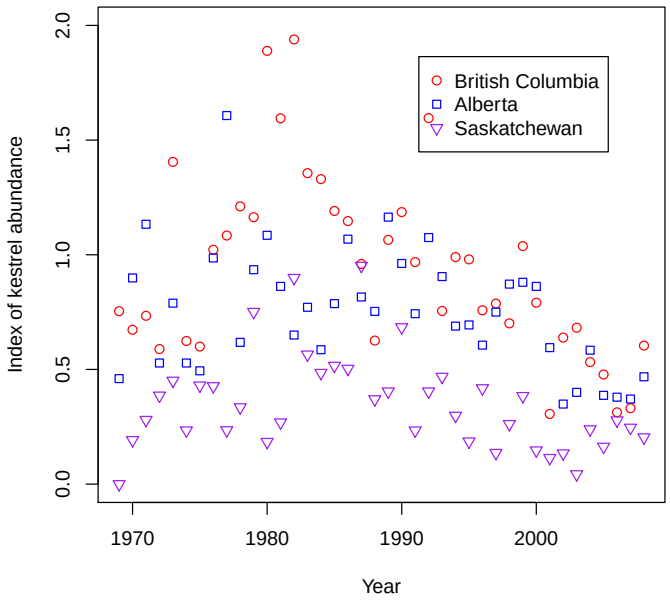

Fig. 16.3. The kestrel data.

We know that the surveys use the same design, so we will force observation error to be shared. Our uncertainty lies in whether these time series are sampling the same population, and how environmental stochasticity varies by subpopulation (if there are subpopulations). Our first model has one population trajectory (meaning there is one big panmictic BC/AB/SK population) and each of these three surveys is an observation of this big population with equal observation variances. Mathematically, the model is:

$$x_t = x_{t-1} + u + w_t, \text{ where } w_t \sim \mathrm{N}(0, q)$$
$$\begin{bmatrix} y_{BC} \\ y_{AB} \\ y_{SK} \end{bmatrix}_t = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} x_t + \begin{bmatrix} 0 \\ a_2 \\ a_3 \end{bmatrix} + \begin{bmatrix} v_{BC} \\ v_{AB} \\ v_{SK} \end{bmatrix}_t, \text{ where } \mathbf{v}_t \sim \mathrm{MVN}\left(0, \begin{bmatrix} r & 0 & 0 \\ 0 & r & 0 \\ 0 & 0 & r \end{bmatrix}\right) \tag{16.5}$$

In MARSS, we denote the model:

```
model.b1=list()
model.b1$R="diagonal and equal"
model.b1$Z=matrix(1,3,1)
kem.b1 = MARSS(birddat, model=model.b1, control=list(minit=100) )
```

As for the redd count example, we do not need to specify the structure of **Q** and **u** since they are scalar and have no structure.

```
kem.b1$AICc
```

```
[1] 20.9067
```

Let's compare this to a model where we assume that there is a separate population for British Columbia, Alberta, and Saskatchewan but they have the same process parameters (trend and process variance). Mathematically, this model is:

$$\begin{bmatrix} x_{BC} \\ x_{AB} \\ x_{SK} \end{bmatrix}_t = \begin{bmatrix} x_{BC} \\ x_{AB} \\ x_{SK} \end{bmatrix}_{t-1} + \begin{bmatrix} u \\ u \\ u \end{bmatrix} + \mathbf{w}_t, \text{ where } \mathbf{w}_t \sim \mathrm{MVN}(0, \begin{bmatrix} q & 0 & 0 \\ 0 & q & 0 \\ 0 & 0 & q \end{bmatrix})$$
$$\begin{bmatrix} y_{BC} \\ y_{AB} \\ y_{SK} \end{bmatrix}_t = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{BC} \\ x_{AB} \\ x_{SK} \end{bmatrix}_t + \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} v_{BC} \\ v_{AB} \\ v_{SK} \end{bmatrix}_t, \text{ where } \mathbf{v}_t \sim \mathrm{MVN}\left(0, \begin{bmatrix} r & 0 & 0 \\ 0 & r & 0 \\ 0 & 0 & r \end{bmatrix}\right) \tag{16.6}$$

This is specified as:

```
model.b2=list()
model.b2$Q="diagonal and equal"
model.b2$R="diagonal and equal"
model.b2$Z="identity"
model.b2$A="zero"
model.b2$U="equal"
kem.b2 = MARSS(birddat, model=model.b2)
```

The AICc for this model is

```
kem.b2$AICc
```

```
[1] 22.96714
```

Because these populations are surveyed over a relatively large geographic area, it is reasonable to expect that environmental variation may differ between populations. For our third model, we will fit a model with separate processes that are allowed to have unequal process parameters.

```
model.b3=model.b2 #is is based on model.b2
#all we change is the structure of Q
model.b3$Q="diagonal and unequal"
model.b3$U="unequal"
kem.b3 = MARSS(birddat, model=model.b3)
```

```
kem.b3$AICc
```

```
[1] 23.75125
```

Finally for a fourth model, we will consider lumping Alberta/Saskatchewan, because the time series suggest similar trends. Mathematically, this model is:

$$\begin{bmatrix} x_{BC} \\ x_{AB-SK} \end{bmatrix}_t = \begin{bmatrix} x_{BC} \\ x_{AB-SK} \end{bmatrix}_{t-1} + \begin{bmatrix} u \\ u \end{bmatrix} + \mathbf{w}_t, \text{ where } \mathbf{w}_t \sim \text{MVN}\left(0, \begin{bmatrix} q & 0 \\ 0 & q \end{bmatrix}\right)$$
$$\begin{bmatrix} y_{BC} \\ y_{AB} \\ y_{SK} \end{bmatrix}_t = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_{BC} \\ x_{AB-SK} \end{bmatrix}_t + \begin{bmatrix} 0 \\ 0 \\ a_3 \end{bmatrix} + \begin{bmatrix} v_{BC} \\ v_{AB} \\ v_{SK} \end{bmatrix}_t, \text{ where } \mathbf{v}_t \sim \text{MVN}\left(0, \begin{bmatrix} r & 0 & 0 \\ 0 & r & 0 \\ 0 & 0 & r \end{bmatrix}\right) \tag{16.7}$$

This model is specified as

```
model.b4=list()
model.b4$Q="diagonal and unequal"
model.b4$R="diagonal and equal"
model.b4$Z=factor(c("BC","AB-SK","AB-SK"))
model.b4$A="scaling"
model.b4$U="unequal"
kem.b4 = MARSS(birddat, model=model.b4)
```

```
kem.b4$AICc
```

```
[1] 14.76889
```

This last model was superior to the others, improving the AICc value compared to model 1 by 8 units. Figure 16.4 shows the fits for this model.

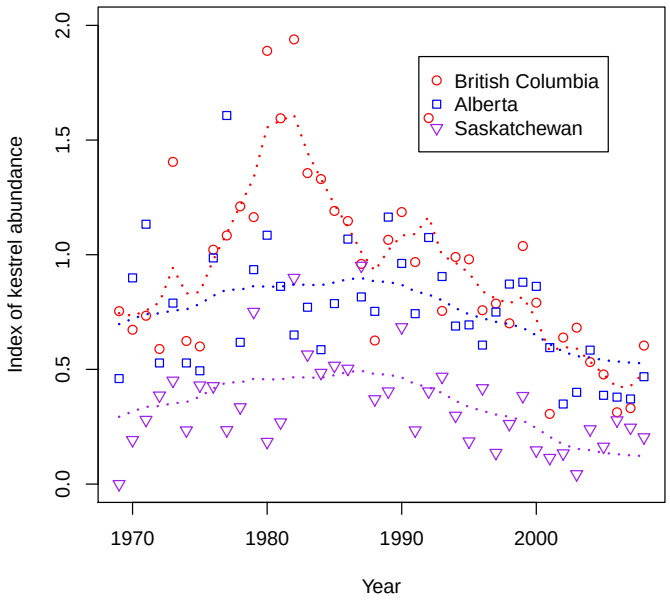

Fig. 16.4. Plot model 4 fits to the kestrel data.

A

Textbooks and articles that use MARSS modeling for population modeling

Textbooks Describing the Estimation of Process and Non-process Variance

There are many textbooks on Kalman filtering and estimation of state-space models. The following are a sample of books on state-space modeling that we have found especially helpful.

Shumway, R. H., and D. S. Stoffer. 2006. Time series analysis and its applications. Springer-Verlag.

Harvey, A. C. 1989. Forecasting, structural time series models and the Kalman filter. Cambridge University Press.

Durbin, J., and S. J. Koopman. 2001. Time series analysis by state space methods. Oxford University Press.

Kim, C. J. and Nelson, C. R. 1999. State space models with regime switching. MIT Press.

King, R., G. Olivier, B. Morgan, and S. Brooks. 2009. Bayesian analysis for population ecology. CRC Press.

Giovanni, P., S. Petrone, and P. Campagnoli. 2009. Dynamic linear models in R. Springer-Verlag.

Pole, A., M. West, and J. Harrison. 1994. Applied Bayesian forecasting and time series analysis. Chapman and Hall.

Bolker, B. 2008. Ecological models and data in R. Princeton University Press.

West, M. and Harrison, J. 1997. Bayesian forecasting and dynamic models. Springer-Verlag.

Tsay, R. S. 2010. Analysis of financial time series. Wiley.

Maximum-likelihood papers

This is just a sample of the papers from the population modeling literature.

de Valpine, P. 2002. Review of methods for fitting time-series models with process and observation error and likelihood calculations for nonlinear, non-Gaussian state-space models. Bulletin of Marine Science 70:455-471.

de Valpine, P. and A. Hastings. 2002. Fitting population models incorporating process noise and observation error. Ecological Monographs 72:57-76.

de Valpine, P. 2003. Better inferences from population-dynamics experiments using Monte Carlo state-space likelihood methods. Ecology 84:3064-3077.

de Valpine, P. and R. Hilborn. 2005. State-space likelihoods for nonlinear fisheries time series. Canadian Journal of Fisheries and Aquatic Sciences 62:1937-1952.

Dennis, B., J.M. Ponciano, S.R. Lele, M.L. Taper, and D.F. Staples. 2006. Estimating density dependence, process noise, and observation error. Ecological Monographs 76:323-341.

Ellner, S.P. and E.E. Holmes. 2008. Resolving the debate on when extinction risk is predictable. Ecology Letters 11:E1-E5.

Erzini, K. 2005. Trends in NE Atlantic landings (southern Portugal): identifying the relative importance of fisheries and environmental variables. Fisheries Oceanography 14:195-209.

Erzini, K., Inejih, C. A. O., and K. A. Stobberup. 2005. An application of two techniques for the analysis of short, multivariate non-stationary time-series of Mauritanian trawl survey data ICES Journal of Marine Science 62:353-359.

Hinrichsen, R.A. and E.E. Holmes. 2009. Using multivariate state-space models to study spatial structure and dynamics. In Spatial Ecology (editors Robert Stephen Cantrell, Chris Cosner, Shigui Ruan). CRC/Chapman Hall.

Hinrichsen, R.A. 2009. Population viability analysis for several populations using multivariate state-space models. Ecological Modelling 220:1197-1202.

Holmes, E.E. 2001. Estimating risks in declining populations with poor data. Proceedings of the National Academy of Sciences of the United States of America 98:5072-5077.

Holmes, E.E. and W.F. Fagan. 2002. Validating population viability analysis for corrupted data sets. Ecology 83:2379-2386.

Holmes, E.E. 2004. Beyond theory to application and evaluation: diffusion approximations for population viability analysis. Ecological Applications 14:1272-1293.

Holmes, E.E., W.F. Fagan, J.J. Rango, A. Folarin, S.J.A., J.E. Lippe, and N.E. McIntyre. 2005. Cross validation of quasi-extinction risks from real time series: An examination of diffusion approximation methods. U.S. Department of Commerce, NOAA Tech. Memo. NMFS-NWFSC-67, Washington, DC.

Holmes, E.E., J.L. Sabo, S.V. Viscido, and W.F. Fagan. 2007. A statistical approach to quasi-extinction forecasting. Ecology Letters 10:1182-1198.

Kalman, R.E. 1960. A new approach to linear filtering and prediction problems. Journal of Basic Engineering 82:35-45.

Lele, S.R. 2006. Sampling variability and estimates of density dependence: a composite likelihood approach. Ecology 87:189-202.

Lele, S.R., B. Dennis, and F. Lutscher. 2007. Data cloning: easy maximum likelihood estimation for complex ecological models using Bayesian Markov chain Monte Carlo methods. Ecology Letters 10:551-563.

Lindley, S.T. 2003. Estimation of population growth and extinction parameters from noisy data. Ecological Applications 13:806-813.

Ponciano, J.M., M.L. Taper, B. Dennis, S.R. Lele. 2009. Hierarchical models in ecology: confidence intervals, hypothesis testing, and model selection using data cloning. Ecology 90:356-362.

Staples, D.F., M.L. Taper, and B. Dennis. 2004. Estimating population trend and process variation for PVA in the presence of sampling error. Ecology 85:923-929.

Zuur, A. F., and G. J. Pierce. 2004. Common trends in Northeast Atlantic squid time series. Journal of Sea Research 52:57-72.

Zuur, A. F., I. D. Tuck, and N. Bailey. 2003. Dynamic factor analysis to estimate common trends in fisheries time series. Canadian Journal of Fisheries and Aquatic Sciences 60:542-552.

Zuur, A. F., R. J. Fryer, I. T. Jolliffe, R. Dekker, and J. J. Beukema. 2003. Estimating common trends in multivariate time series using dynamic factor analysis. Environmetrics 14:665-685.

Bayesian papers

This is a sample of the papers from the population modeling and animal tracking literature.

Buckland, S.T., K.B. Newman, L. Thomas and N.B. Koestersa. 2004. State-space models for the dynamics of wild animal populations. Ecological modeling 171:157-175.

Calder, C., M. Lavine, P. Müller, J.S. Clark. 2003. Incorporating multiple sources of stochasticity into dynamic population models. Ecology 84:1395-1402.

Chaloupka, M. and G. Balazs. 2007. Using Bayesian state-space modelling to assess the recovery and harvest potential of the Hawaiian green sea turtle stock. Ecological Modelling 205:93-109.

Clark, J.S. and O.N. Bjørnstad. 2004. Population time series: process variability, observation errors, missing values, lags, and hidden states. Ecology 85:3140-3150.

Jonsen, I.D., R.A. Myers, and J.M. Flemming. 2003. Meta-analysis of animal movement using state space models. Ecology 84:3055-3063.

Jonsen, I.D, J.M. Flemming, and R.A. Myers. 2005. Robust state-space modeling of animal movement data. Ecology 86:2874-2880.

Meyer, R. and R.B. Millar. 1999. BUGS in Bayesian stock assessments. Can. J. Fish. Aquat. Sci. 56:1078-1087.

Meyer, R. and R.B. Millar. 1999. Bayesian stock assessment using a state-space implementation of the delay difference model. Can. J. Fish. Aquat. Sci. 56:37-52.

Meyer, R. and R.B. Millar. 2000. Bayesian state-space modeling of age-structured data: fitting a model is just the beginning. Can. J. Fish. Aquat. Sci. 57:43-50.

Newman, K.B., S.T. Buckland, S.T. Lindley, L. Thomas, and C. Fernández. 2006. Hidden process models for animal population dynamics. Ecological Applications 16:74-86.

Newman, K.B., C. Fernández, L. Thomas, and S.T. Buckland. 2009. Monte Carlo inference for state-space models of wild animal populations. Biometrics 65:572-583

Rivot, E., E. Prévost, E. Parent, and J.L. Baglinière. 2004. A Bayesian state-space modelling framework for fitting a salmon stage-structured population dynamic model to multiple time series of field data. Ecological Modeling 179:463-485.

Schnute, J.T. 1994. A general framework for developing sequential fisheries models. Canadian J. Fisheries and Aquatic Sciences 51:1676-1688.

Swain, D.P., I.D. Jonsen, J.E. Simon, and R.A. Myers. 2009. Assessing threats to species at risk using stage-structured state-space models: mortality trends in skate populations. Ecological Applications 19:1347-1364.

Thogmartin, W.E., J.R. Sauer, and M.G. Knutson. 2004. A hierarchical spatial model of avian abundance with application to cerulean warblers. Ecological Applications 14:1766-1779.

Trenkel, V.M., D.A. Elston, and S.T. Buckland. 2000. Fitting population dynamics models to count and cull data using sequential importance sampling. J. Am. Stat. Assoc. 95:363-374.

Viljugrein, H., N.C. Stenseth, G.W. Smith, and G.H. Steinbakk. 2005. Density dependence in North American ducks. Ecology 86:245-254.

Ward, E.J., R. Hilborn, R.G. Towell, and L. Gerber. 2007. A state-space mixture approach for estimating catastrophic events in time series data. Can. J. Fish. Aquat. Sci., Can. J. Fish. Aquat. Sci. 644:899-910.

Wikle, C.K., L.M. Berliner, and N. Cressie. 1998. Hierarchical Bayesian space-time models. Journal of Environmental and Ecological Statistics 5:117-154

Wikle, C.K. 2003. Hierarchical Bayesian models for predicting the spread of ecological processes. Ecology 84:1382-1394.

B

Package MARSS: Warnings and errors

The following are brief descriptions of the warning and error message you may see and what they mean (or might mean).

B update is outside the unit circle

If you are estimating $\mathbf{B}$, then if the absolute value of all the eigenvalues of $\mathbf{B}$ are less than 1, the system is stationary (meaning the $\mathbf{X}$'s have some multivariate distribution that does not change over time). In this case, we say that $\mathbf{B}$ is within the unit circle. A pure univariate random walk for example would have $\mathbf{B}$=1 and it is not stationary. The distribution of $\mathbf{X}$ for the pure random walk has a variance that increases with time. If on the other hand $|\mathbf{B}| < 1$, you have an Ornstein-Uhlenbeck process and is stationary, with a stationary variance of $\mathbf{Q}/(1-\mathbf{B}^2)$ (note $\mathbf{B}$ is a scalar here because in this example $\mathbf{X}$ is univariate). If any of the eigenvalues (real part) are greater than 1, then the system will "explode"—it rapidly diverges.

In the EM algorithm, there is nothing to force $\mathbf{B}$ to be on or within the unit circle (real part of the eigenvalues less than or equal to 1). It is possible at one of the EM iterations the $\mathbf{B}$ update will be outside the unit circle. The problem is that if you get too far outside the unit circle, the algorithm becomes numerically unstable since small errors are magnified by the "explosive" $\mathbf{B}$ term. If you see the 'B outside the unit circle' warning, it is fine as long as it is temporary and the log-likelihood does not start decreasing (you will see a separate warning if that happens).

If you do see $\mathbf{B}$ outside the unit circle and the log-likelihood decreases, then it probably means that you have poorly specified the model somehow. An easy way to do this is to poorly specify the initial conditions, π and Λ. If, say, you try to specify a vague prior on $\mathbf{x}_0$ (or $\mathbf{x}_1$) with π equal to zero and Λ equal to a diagonal matrix with a large variance on the diagonal, you will likely run into trouble if $\mathbf{B}$ has off-diagonal terms. The reason is that by specifying that Λ is diagonal, you specified that the individual X's in $\mathbf{X}_0$ are

independent, yet if $\mathbf{B}$ has off-diagonal terms, the stationary distribution of $\mathbf{X}_1$ is NOT independent. If you force the diagonal terms on Λ to be big enough, you can force the maximum-likelihood estimate of $\mathbf{B}$ to be outside the unit circle since this is the only way to account for $\mathbf{X}_0$ independent and $\mathbf{X}_1$ highly correlated.

The problem is that you will not know the stationary distribution of the $\mathbf{X}$'s (from which $\mathbf{X}_0$ was presumably drawn) without knowing the parameters you are trying to estimate. One approach is the estimate both π and Λ by setting `x0="unconstrained"` and `V0="unconstrained"` in the model specification. Estimating both π and Λ cannot be done robustly for all MARSS models, and in general, one probably wants to specify the model in such a way as to fix one or both of these. Another, more robust approach, is to treat $\mathbf{x}_1$ as fixed but unknown (instead of $\mathbf{x}_0$). You do this by setting `model$tinitx=1`, so that π refers to $t = 1$ not $t = 0$. Then estimate π and fix $\Lambda = 0$. This eliminates Λ from the model and often eliminates the problems with prior specification—as the expense of m more parameters. Note, when you set $\Lambda = 0$, Λ is truly eliminated from the model by MARSS; the likelihood function is different, so do not expect $\Lambda = 0$ and $\Lambda \sim 0$ to have the same likelihood under all conditions.

Warning! Reached maxit before parameters converged

The maximum number of EM iterations is set by `control$maxit`. If you get this warning, it means that one of the parameters or log-likelihood had not yet reached the convergence stopping criteria before `maxit` was reached. There are many situations where you might want to set `control$maxit` lower than the value needed to reach convergence. For example, if you are using the EM algorithm to produce initial values for a different algorithm (like a Bayesian MCMC algorithm or a Newton method) then you can set `maxit` low, say 20 or 50.

Stopped at iter=xx in MARSSkem() because numerical errors were generated in MARSSkf

This means the Kalman filter/smoother algorithm became unstable and most likely one of the variances became ill-conditioned. When that happens the inverses of those matrices are poor, and you will start to get negative values on the diagonals of your variance-covariance matrices. Once that happens, the inverse of that var-covariance matrix produces an error. If you get this error, turn on tracing with `control$trace=1`. This will store the error messages so you can see what is going on. It may be that you have specified the model in such a way that some of the variances are being forced very close to 0, which makes the var-covariance matrix ill-conditioned. The output from the MARSS call will be the parameter values just before the error occurred.

Warning: the xyz parameter value has not converged

The algorithm checks whether the log-likelihood and each individual parameter has converged. If a parameter has not converged, you can try upping `control$maxit` and see if it converges. If you set, `maxit` high, but the parameter is still not converging, then it suggests that one of the variance parameters is so small that the EM update steps for that parameter are tiny. For example, as $\mathbf{Q}$ goes to zero, the update steps for $\mathbf{u}$ go to zero. As Λ goes to zero, the update steps for π go to zero. The first thing to do is to reflect on whether you are inadvertently specifying the model in such a way that one of the variances is forced to zero. For example, if the total variance in $\mathbf{X}$ is 0.1 and you fix $\mathbf{R} = 0.2$ then $\mathbf{Q}$ must go to zero. The second thing to do is to try using a Newton algorithm, using your last EM values as the initial conditions for the Newton algorithm. The initial values are set using the `inits` argument for the `MARSS()` function.

MARSSkem: The soln became unstable and logLik DROPPED

This is a more serious error as in the EM algorithm, the log-likelihood should never drop. The first thing to do is check if you have specified a bizarre model or data, inadvertently. Plot the data you are trying to fit. Often, this error arises when a user has inadvertently scrambled their data order during a demeaning or variance-standardization step. Second, check the model you are trying to fit. Use `test=MARSS(data, model=xyz, fit=FALSE)` and then `summary(test$model)`. This shows you what MARSS() thinks your model is. You may be trying to fit an illogical model.

If those checks looks good, then pass `control$trace=1` into the `MARSS()` call. This will report a fuller set of warnings. Look if the error "B is outside the unit circle" appears. If so, you are probably specifying a strange $\mathbf{B}$ matrix. Are you forcing the $\mathbf{B}$ matrix to be outside the unit circle (eigenvalues $>$ 1)? If so, you need to rethink your $\mathbf{B}$ matrix contraints. If you do not see that error, look at `test$iter.record$logLik`. If the log-likelihood is steadily dropping (at each iteration) or drops by large amounts (much larger than the machine precision), that is bad and means that the EM algorithm did not work. If however the log-likelihood is just fluctuating by small amounts about some steady value, that is ok as it means that the values converged but the parameters are such that there are slight numerical fluctuations. Try passing `control$safe=TRUE` in the `MARSS()` call. This can sometimes help as it inserts a call to the Kalman filter after each individual parameter update.

Stopped at iter=xx in MARSSkem: solution became unstable. R (or Q) update is not positive definite

First check if you have specified an illegally constrained variance-covariance matrix. For example, if the variances (diagonal) are constrained to be equal, you cannot specify the covariances (off-diagonals) as unequal. Or if you specify that some of the covariances are equal, you cannot specify the variances as all unequal. These are illegal constraints on a variance-covariance matrix from a statistical perspective (nothing to do with MARSS specifically).

This could also be due to numerical instability as **B** leaves the unit circle or one of the variance matrix becomes ill-conditioned. Try turning on tracing with `control$trace=1` and turn on safe with `control$safe=TRUE`. This will print out the error warnings at each parameter update step. Then consider whether you have inadvertently specified the model in such a way as to force this behavior in the **B** parameter.

You might also get this error if you inadvertantly specified an improper structure for **R** or **Q**. For example, if you used `R=diag(c(1,1,"r"))` with the intent of specifying a diagonal matrix with fixed variance 1 at $\mathbf{R}[1,1]$ and $\mathbf{R}[2,2]$ and an estimated $\mathbf{R}[3,3]$, you would have actually specified a character matrix with `"0"` on the off-diagonals and `c("1","1","r")` on the diagonal. MARSS() interprets all elements in quotes as names of parameters to be estimated. Thus it will estimate one off-diagonal covariance and two diagonal variances. That happens to put illegal constraints on estimation of a variance-covariance matrix having nothing to do with MARSS() but with estimation of variance-covariance matrices in general.

iter=xx MARSSkf: logLik computation is becoming unstable. Condition num. of Sigma[t=1] = Inf and of R = Inf.

This means, generally, that V0 is very small, say 0, and R is very small and very close to zero.

Warning: setting diagonal to 0 blocked at iter=X. logLik was lower in attempt to set 0 diagonals on X

This is a warning not an error. What is happening is that one of the variances (in **Q** or **R**) is getting small and the EM algorithm is attempting to set the value to 0 (because `control$degen.allow=TRUE`). But when it tried to do this, the new likelihood with the variance equal to 0 was lower and the variance was not set to 0.

A model with a variance minuscule and a model with the same variance equal to 0 are not the same model. In the first, a stochastic process with

small variance exists but in the second, the analogous process is deterministic. And in the first case, you can get a situation where the likelihood term L(x|mean=mu,sigma=0) appears. That term will be infinite when x=mu. So in the model with variance minuscule, you will get very large likelihood values as the variance term gets smaller and smaller. In the analogous model with that variance set to 0, that likelihood term does not appear so the likelihood does not go to infinity.

This is not an error nor pathological behavior; the models are fundamentally different. Nonetheless, this will pose a dilemma when you want to chose the best model based on maximum likelihood. The model with minuscule variance will have infinite likelihood but the same behavior as the one with variance 0. In our experience, this dilemma arises when one has a lot of missing data near the beginning of the time series and is affected by how you specify the prior on the initial state. Try setting the prior at $t = 0$ versus $t = 1$. Try using a diffuse prior. You absolutely want to compare estimates using the BFGS and EM algorithms in this case, because the different algorithms differ in their ability to find the maximum in this strange case. Neither is uniformly better or worse. It seems to depend on which variance (**Q** or **R**) is going to zero.

Warning: kf returned error at iter=X in attempt to set 0 diagonals for X

This is a warning that the EM algorithm tried to set one of the diagonals of element X to 0 because `allow.degen` is TRUE and element X is going to zero. However when this was tried, the Kalman filter returned an error. Typically, this happens when both **R** and **Q** elements are both trying to be set to 0. If the maximum-likelihood estimate is that both **R** and **Q** are zero, it probably means that your MARSS model is not a very good description of the data.

Warning: At iter=X attempt to set 0 diagonals for R blocked for elements where corresponding rows of A or Z are not fixed.

You have `control$degen.allow=TRUE` and one of the **R** diagonal elements is getting very small. MARSS attempts to set these **R** elements to 0, but if row i of **R** is 0, then the corresponding i rows of **a** and **Z** must be fixed. This is for the EM algorithm. It might work with the BFGS algorithm, or might spit out garbage without telling you. Always be a suspect, when the EM and BFGS behavior is different. That is a good sign that something is wrong with how your model describes the data. It's not a problem with the algorithms per se; rather for certain pathological models, the algorithms behave differently from each other.

Stopped at iter=X in MARSSkem. XYZ is not invertible.

There are a series of checks in MARSS that check if matrix inversions are possible before doing the inversion. These errors crop up most often when **Q** or **R** are getting very small. At some point, they can get so small that inversions become unstable. If this error is given, then the output will be the last parameter estimates before the error. Try setting `control$allow.degen=FALSE`. Sometimes the error occurs when a diagonal element of **Q** or **R** is being set to 0. You will also have to set `control$maxit` to something smaller because the EM algorithm will not stop since the problematic diagonal element will walk slowly and inexorably to 0.

References

Biernacki, C., Celeux, G., and Govaert, G. 2003. Choosing starting values for the EM algorithm for getting the highest likelihood in multivariate gaussian mixture models. *Computational Statistics and Data Analysis* 41:561–575.

Brockwell, P. J. and Davis, R. A. 1991. Time series: theory and methods. Springer-Verlag, New York, NY.

Cavanaugh, J. and Shumway, R. 1997. A bootstrap variant of AIC for state-space model selection. *Statistica Sinica* 7:473–496.

Cheang, W. K. and Reinsel, G. C. 2000. Bias reduction of autoregressive estimates in time series regression model through restricted maximum likelihood. *Journal of the American Statistical Association* 95:1173–1184.

de Jong, P. and Penzer, J. 1998. Diagnosing shocks in time series. *Journal of the American Statistical Association* 93:796–806.

Dempster, A., Laird, N., and Rubin, D. 1977. Likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B* 39:1–38.

Dennis, B., Munholland, P. L., and Scott, J. M. 1991. Estimation of growth and extinction parameters for endangered species. *Ecological Monographs* 61:115–143.

Dennis, B., Ponciano, J. M., Lele, S. R., Taper, M. L., and Staples, D. F. 2006. Estimating density dependence, process noise, and observation error. *Ecological Monographs* 76:323–341.

Ellner, S. P. and Holmes, E. E. 2008. Resolving the debate on when extinction risk is predictable. *Ecology Letters* 11:E1–E5.

Gerber, L. R., Master, D. P. D., and Kareiva, P. M. 1999. Grey whales and the value of monitoring data in implementing the u.s. endangered species act. *Conservation Biology* 13:1215–1219.

Ghahramani, Z. and Hinton, G. E. 1996. Parameter estimation for linear dynamical systems. Technical Report CRG-TR-96-2, University of Totronto, Dept. of Computer Science.

Hampton, S. E., Izmest'Eva, L. R., Moore, M. V., Katz, S. L., Dennis, B., and Silow, E. A. 2008. Sixty years of environmental change in the world's largest freshwater lake – Lake Baikal, Siberia. *Global Change Biology* 14:1947–1958.

Hampton, S. E., Scheuerell, M. D., and Schindler, D. E. 2006. Coalescence in the lake washington story: Interaction strengths in a planktonic food web. *Limnology and Oceanography* 51:2042–2051.

Hampton, S. E. and Schindler, D. E. 2006. Empirical evaluation of observation scale effects in community time series. *Oikos* 113:424–439.

Harvey, A., Koopman, S. J., and Penzer, J. 1998. Messy time series: a unified approach. *Advances in Econometrics* 13:103–143.

Harvey, A. C. 1989. Forecasting, structural time series models and the Kalman filter. Cambridge University Press, Cambridge, UK.

Harvey, A. C. and Koopman, S. J. 1992. Diagnostic checking of unobserved components time series models. *Journal of Business and Economic Statistics* 10:377–389.

Harvey, A. C. and Shephard, N. 1993. Structural time series models. Elsevier Science Publishers B V, Amsterdam.

Hinrichsen, R. 2009. Population viability analysis for several populations using multivariate state-space models. *Ecological Modelling* 220:1197–1202.

Hinrichsen, R. and Holmes, E. E. 2009. Using multivariate state-space models to study spatial structure and dynamics. CRC/Chapman Hall.

Holmes, E. E. 2001. Estimating risks in declining populations with poor data. *Proceedings of the National Academy of Sciences of the United States of America* 98:5072–5077.

Holmes, E. E. 2004. Beyond theory to application and evaluation: diffusion approximations for population viability analysis. *Ecological Applications* 14:1272–1293.

Holmes, E. E. 2012. Derivation of the EM algorithm for constrained and unconstrained marss models. Technical report, Northwest Fisheries Science Center, Mathematical Biology Program.

Holmes, E. E., Sabo, J. L., Viscido, S. V., and Fagan, W. F. 2007. A statistical approach to quasi-extinction forecasting. *Ecology Letters* 10:1182–1198.

Holmes, E. E., Ward, E. J., and Wills, K. 2012. Marss: Multivariate autoregressive state-space models for analyzing time-series data. *The R Journal* 4:11–19.

Holmes, E. E. and Ward, E. W. 2010. Analyzing noisy, gappy, and multivariate population abundance data: modeling, estimation, and model selection in a maximum-likelihood framework. Technical report, Northwest Fisheries Science Center, Mathematical Biology Program.

Ives, A. R. 1995. Measuring resilience in stochastic systems. *Ecological Monographs* 65:217–233.

Ives, A. R., Abbott, K. C., and Ziebarth, N. L. 2010. Analysis of ecological time series with arma(p,q) models. *Ecology* 91:858–871.

Ives, A. R., Carpenter, S. R., and Dennis, B. 1999. Community interaction webs and zooplankton responses to planktivory manipulations. *Ecology* 80:1405–1421.

Ives, A. R., Dennis, B., Cottingham, K. L., and Carpenter, S. R. 2003. Estimating community stability and ecological interactions from time-series data. *Ecological Monographs* 73:301–330.

Jeffries, S., Huber, H., Calambokidis, J., and Laake, J. 2003. Trends and status of harbor seals in washington state 1978-1999. *Journal of Wildlife Management* 67:208–219.

Kalman, R. E. 1960. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering* 82:35–45.

Klug, J. L. and Cottingham, K. L. 2001. Interactions among environmental drivers: Community responses to changing nutrients and dissolved organic carbon. *Ecology* 82:3390–3403.

Kohn, R. and Ansley, C. F. 1989. A fast algorithm for signal extraction, influence and cross-validation in state-space models. *Biometrika* 76:65–79.

Koopman, S. J. 1993. Distrubance smoother for state space models. *Biometrika* 80:117–126.

Koopman, S. J., Shephard, N., and Doornik, J. A. 1999. Statistical algorithms for models in state space using ssfpack 2.2. *Econometrics Journal* 2:113–166.

Lele, S. R., Dennis, B., and Lutscher, F. 2007. Data cloning: easy maximum likelihood estimation for complex ecological models using bayesian markov chain monte carlo methods. *Ecology Letters* 10:551–563.

McLachlan, G. J. and Krishnan, T. 2008. The EM algorithm and extensions. John Wiley and Sons, Inc., Hoboken, NJ, 2nd edition.

Penzer, J. 2001. Critical values for time series diagnostics. Technical report, Department of Statistics, London School of Economics.

Rauch, H. E. 1963. Solutions to the linear smoothing problem. *IEEE Transactions on Automatic Control* 8:371–372.

Rauch, H. E., Tung, F., and Striebel, C. T. 1965. Maximum likelihood estimation of linear dynamical systems. *Journal of AIAA* 3:1445–1450.

Schweppe, F. C. 1965. Evaluation of likelihood functions for Gaussian signals. *IEEE Transactions on Information Theory* IT-r:294–305.

Shumway, R. and Stoffer, D. 2006. Time series analysis and its applications. Springer-Science+Business Media, LLC, New York, New York, 2nd edition.

Shumway, R. H. and Stoffer, D. S. 1982. An approach to time series smoothing and forecasting using the EM algorithm. *Journal of Time Series Analysis* 3:253–264.

Staples, D. F., Taper, M. L., and Dennis, B. 2004. Estimating population trend and process variation for PVA in the presence of sampling error. *Ecology* 85:923–929.

Staudenmayer, J. and Buonaccorsi, J. R. 2005. Measurement error in linear autoregressive models. *Journal of the American Statistical Association* 10:841–852.

Stoffer, D. S. and Wall, K. D. 1991. Bootstrapping state-space models: Gaussian maximum likelihood estimation and the Kalman filter. *Journal of the American Statistical Association* 86:1024–1033.

Taper, M. L. and Dennis, B. 1994. Density dependence in time series observations of natural populations: estimation and testing. *Ecological Monographs* 64:205–224.

Tsay, R. S. 2010. Analysis of financial time series. Wiley Series in Probability and Statistics. Wiley.

Ward, E. J., Chirakkal, H., González-Suárez, M., Aurioles-Gamboa, D., Holmes, E. E., and Gerber, L. 2010. Inferring spatial structure from time-series data: using multivariate state-space models to detect metapopulation structure of California sea lions in the Gulf of California, Mexico. *Journal of Applied Ecology* 1:47–56.

Zuur, A. F., Fryer, R. J., Jolliffe, I. T., Dekker, R., and Beukema, J. J. 2003. Estimating common trends in multivariate time series using dynamic factor analysis. *Environmetrics* 14:665–685.

Index

animal tracking, 149
  kftrack, 155

bootstrap
  innovations, 13, 26, 27
  MARSSboot function, 13
  parametric, 13, 26, 27

confidence intervals, 98
  Hessian approximation, 13, 98
  MARSSparamCIs function, 13
  non-parametric bootstrap, 13
  parametric bootstrap, 13, 98
covariates, 51, 182, 185

density-independent, 81
diagnostics, 109

error
  observation, 82
  process, 81, 82
errors
  degenerate, 7
  ill-conditioned, 7
estimation, 85
  BFGS, 32
  Dennis method, 86
  EM, 12, 25, 85
  Kalman filter, 13, 23
  Kalman smoother, 13, 23
  maximum-likelihood, 85, 86
  Newton methods, 26
  quasi-Newton, 12, 32
  REML, 5

extinction, 81
  diffusion approximation, 90
  uncertainty, 94

functions
  coef, 44
  is.marssMLE, 12
  is.marssMODEL, 13
  MARSS, 11, 31, 34, 36, 38
  MARSSaic, 13, 26, 27, 49
  MARSSboot, 13, 26, 47
  MARSShessian, 13
  MARSSkem, 12, 25, 26
  MARSSkf, 13, 23, 24, 44
  MARSSkfas, 24
  MARSSkfss, 24
  MARSSmcinit, 12, 26
  MARSSoptim, 12
  MARSSparamCIs, 6, 13, 26, 42
  MARSSsimulate, 13, 27, 49
  MARSSvectorizeparam, 13
  optim, 12
  summary, 13, 40

initial conditions
  setting for BFGS, 33

Kalman filter and smoother, 44

lag-1 covariance smoother, 44
likelihood, 13, 24, 49
  and missing values, 25
  innovations algorithm, 24
  MARSSkf function, 49

missing value modifications, 24
multimodal, 26
troubleshooting, 7, 26

MAR(p), 69
MARSS model, 1, 4, 149
DFA example, 131
multivariate example, 101, 121, 149
print, 40
summary, 40
univariate example, 81
missing values, 5
and AICb, 27
and parametric bootstrap, 26
likelihood correction, 25
model selection, 27, 121
AIC, 27, 107, 109, 118, 119, 125, 129
AICc, 27, 118
bootstrap AIC, 27, 118
bootstrap AIC, AICbb, 27, 49
bootstrap AIC, AICbp, V, 27, 49, 118
MARSSaic function, 13, 49
model specification
in MARSS, 15

objects
marssMLE, 11
marssMODEL, 11, 13
outliers, 159

prior, 2, 19, 30
diffuse, 163
troubleshooting, 6, 33, 172, 206, 208

simulation, 27, 49, 82
MARSSsimulate function, 13, 49
standard errors, 13
structural breaks, 159