E. E. Holmes and E. J. Ward

Analysis of multivariate time-series using the MARSS package

June 8, 2010

Mathematical Biology Program
Northwest Fisheries Science Center

Holmes, E. E. and E. J. Ward. 2010. 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 and eric.ward@noaa.gov

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., 2009; 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, 2010). Discussions with Mark also led to a complete rewrite of the model specification so that the package could be used for MARSS in general – rather than simply the form of MARSS model used in our (EEH and EJW) applications. Many collaborators have helped test the package; we thank especially Yasmin Lucero, Mark Scheuerell, Kevin See, and Brice Semmens. Development of the code into a R package would not have been possible without Kellie Wills, who wrote the much of the code outside of the algorithm functions.

The case studies used in this manual were developed for workshops on analysis of multivariate time-series data given at the Ecological Society meetings since 2005 and taught by us (EEH and EJW) along with Yasmin Lucero, Stephanie Hampton, Brice Semmens, and Mark Scheuerell. The case study on extinction estimation and trend estimation was initially developed by Brice Semmens and later extended by us for this manual. The algorithm behind the TMU figure in Case Study 8 was developed during a collaboration with Steve Ellner (Ellner and Holmes, 2008).

EEH and EJW are research scientists at the Northwest Fisheries Science Center in the Mathematical Biology program. This work was conducted as part of our jobs at the Northwest Fisheries Science Center, a research center for NOAA Fisheries which is a US federal government agency. A CAMEO grant from NOAA Fisheries supported Kellie Wills. During the initial stages of this work, EJW was supported on a post-doctoral fellowship from the National Research Council.

You are welcome to use the code and adapt it with attribution. It may not be used in any commercial applications. 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

MARSS stands for Multivariate Auto-Regressive(1) State-Space. The MARSS package is designed for 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, especially economics, engineering, genetics, physics and ecology. Appendix A gives a selection of textbooks on MARSS models that we have found particularly useful.

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

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

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

$$\mathbf{x}_1 \sim \text{MVN}(\boldsymbol{\pi}, \mathbf{V}_1) \tag{1.1c}$$

The model includes random variables, parameters and data:

$\mathbf{x}_t$	is a $m \times 1$ column vector of the hidden states at time t. It is a realization of the random variable $\mathbf{X}_t$.
$\mathbf{v}_t$	is a $m \times 1$ column vector of the process errors at time t. It is a realization of a multivariate normal random variable with mean 0 and $\Sigma = \mathbf{Q}$.
$\mathbf{y}_t$	is a $n \times 1$ column vector of the observed data at time t.
$\mathbf{w}_t$	is a $n \times 1$ column vector of the non-process errors at time t. It is a realization of a multivariate normal random variable with mean 0 and $\Sigma = \mathbf{R}$.
$\mathbf{B}$	is a parameter and is a $m \times m$ matrix.
$\mathbf{u}$	is a parameter and is a $m \times 1$ column vector.
$\mathbf{Q}$	is a parameter and is a $m \times m$ variance-covariance matrix.
$\mathbf{Z}$	is a parameter and is a $n \times m$ matrix.
$\mathbf{a}$	is a parameter and is a $n \times 1$ column vector.
$\mathbf{R}$	is a parameter and is a $n \times n$ variance-covariance matrix.
$\boldsymbol{\pi}$	is either a parameter or a fixed prior. It is a $m \times 1$ matrix.
$\mathbf{V}_1$	is a fixed value. It is a $m \times m$ variance-covariance matrix.

The meaning of the parameters in the MARSS models depends on the application for which the MARSS model is being used. In the case studies, we show examples of MARSS models used to analyze population count data and animal tracking data, and appendix A gives a selection of papers from the ecological literature. However, the MARSS package is not specific to population modeling applications. The functions in the MARSS package are generic functions for fitting MARSS models of the form in Equation (1.1).

1.1 What does the MARSS package do?

The MARSS package is designed to fit unconstrained and constrained MARSS models. A constrained MARSS model is one in which some of the parameters are constrained in the sense that they have fixed, free and/or shared values. For example, let **M** and **m** be arbitrary matrix and column vector parameters. The MARSS package allows one to specify and fit models where **M** and **m** have the following forms.

$$\mathbf{M} = \begin{bmatrix} a & 0.9 & c \\ -1.2 & a & 0 \\ 0 & c & b \end{bmatrix} \text{ and } \mathbf{m} = \begin{bmatrix} d \\ d \\ e \\ 2.2 \end{bmatrix}$$

Version 1.0 of the MARSS package fits models via maximum-likelihood using a Kalman-EM algorithm[1]. The Kalman-EM algorithm is used because it gives robust estimation for datasets replete with missing values and for models with various constraints. The MARSS package also supplies functions for bootstrap and approximate confidence intervals, parametric and non-parametric bootstrapping, model selection (AIC and bootstrap AIC), simulation, and bootstrap bias correction. Version 1.0 does not allow **B** or **Z** to be estimated and **a** is constrained to act as a scaling factor. Version 2.0 is currently being tested and it will allow **B**, **Z**, and **a** estimation along with less constrained forms of **Q** and **R**.

1.2 Important notes about the algorithms

MARSS 1.0 provides robost maximum-likelihood via an EM algorithm using the Kalman filter/smoother. No attempt has been made to make the algorithm computationally efficient and all code is in native R. Thus the model

[1] The package can also fit models via quasi-Newton methods based on R's `optim` function. This can be especially useful for finishing off a Kalman-EM estimate when the data to parameter ratio is high. However, when the ratio of data to parameters is lower (as in many ecological applications), the quasi-Newton algorithm tends to be fragile and sensitive to initial conditions.

fitting is slow (relatively). Writing the algorithms in C would speed them up considerably, but we have no plans to do that. 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 unlike quasi-Newton methods which can be sensitive to initial condition choices. The MARSS package allows one to use the BFGS method in `optim` to fit MARSS models. The `DLM` package (search for it on CRAN) also provides fitting via quasi-Newton methods (and Bayesian methods).

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. Another maximum-likelihood method is data-cloning which adapts MCMC algorithms used in Bayesian analysis for maximum likelihood estimation (Lele et al., 2007).

Data with cycles, from say age-structure or dynamical interactions, are difficult to analyze and both REML and Kalman-EM approaches will give poor estimates for this type of data. The slope method (Holmes, 2001), is more ad-hoc but is relatively robust to those problems. Holmes et al. (2007) used the slope method in a large study of data from endangered and threatened species; Ellner and Holmes (2008) showed that the slope estimates are close to the theoretical minimum uncertainty. However estimates using the slope method are not easily extended to multi-variate data and it is not a true maximum-likelihood method.

Missing values are seamlessly accommodated with the MARSS package. Simply specify the way missing values are denoted in the data set (default is `miss.value=-99`). The likelihood computations are exact and will deal appropriately with missing values. Parameter estimates and hidden state estimates use the Kalman filter/smoother and EM algorithm with missing value modifications. The presence of missing values, however, limits $\mathbf{R}$ to being a diagonal matrix (if estimated) and $\mathbf{Z}$ to being fixed. In addition, no innovations (non-parametric) bootstrapping can be done if there are missing values. Instead parametric bootstrapping must be used.

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 $\mathbf{R}$ or $\mathbf{Q}$ 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 $\mathbf{R}$ or $\mathbf{Q}$ 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, not close to zero, 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.

1.3 Troubleshooting

There are two numerical errors and warnings that you may see when fitting MARSS models: ill-conditioning and degeneracy. 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. MARSS will print warning messages if the algorithm is 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 $\boldsymbol{\pi}$ values in your model (Equation (1.1)). The way our algorithm deals with $\mathbf{V}_1$ 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.

The second numerical error you may see is a degeneracy warning. 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. Since the likelihood can spike up very fast near a degenerate solution, the log-likelihood value reported will be too small because it will include estimates of the degenerate $\mathbf{Q}$ or $\mathbf{R}$ diagonal elements that are very small but nonetheless non-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 and you can use the function `find.degenerate()` to find the degenerate elements or look at the `$errors` element of the output.

The algorithms in MARSS 1.0 are designed for cases where the $\mathbf{Q}$ and $\mathbf{R}$ diagonals are all non-miniscule. For example, the EM update for 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 $\boldsymbol{\pi}$. The solution is to use the degenerate likelihood function for the likelihood calculation and the EM update equations. However this solution will not be implemented until MARSS 2.0. These concerns affect the likelihood value reported at the maximum-likelihood parameters. The actual parameter estimates will change very little on the absolute scale, so your point estimates, confidence intervals, bias estimates, etc. are still valid. Model selection though will be dubious because the likelihoods reported by MARSS will not be the real maximums.

2 MARSS functions

The MARSS package is object-based. It has two main types of objects: a model object (`class=marssm`) and a maximum-likelihood fitted model object (`class=marssMLE`). A marssm 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

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 simple text strings to describe the model structure. It returns marssm and marssMLE objects which the user can later use in other functions, e.g. simulating or computing bootstrap confidence intervals.

`MLEobj=MARSS(data, constraint=list(), ..., fit=TRUE)` This function will fit a MARSS model to the data using a constraint list which is a list of strings describing the structure of the model parameter matrices. The default model has a one-to-one correspondence between the state processes and observation time series ($\mathbf{Z}$ is identity matrix). The default has a diagonal observation error matrix ($\mathbf{R}$) and an unconstrainted process-error matrix ($\mathbf{Q}$). 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 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 Kalman-EM algorithm to the data using a properly specified marssMLE object, which has data, the marssm 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.

2.3 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 marssm object and `MLEobj` means the argument is a marssMLE object. Type `?function.name` to see information on function usage and examples.

`kf=MARSSkf(data, parList, ...)` 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 `parList`. A variety of other Kalman filter/smoother information is also output (`kf` is a list of output); see `?MARSSkf` for more 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(parList)` This returns simulated data from a MARSS model specified via a list of parameter matrices in `parList` (this is a list with elements `Q`, `R`, `U`, etc). Typically one would pass in `parList` using the `par` element in an marssMLE object (i.e., `MLEobj$par`), but you could also construct the list manually.

`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 returns 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`. This is useful when you have bootstrapped parameter sets. You can then create proper marssMLE objects from the bootstrapped parameters using `boot.MLEobj=MARSSvectorizeparam(MLEobj, bootparamVec)` and simulate from those using `MARSSsimulate`.

2.4 Functions for marssm objects

`is.marssm(modelObj)` This will check that the free and fixed matrices in a marssm object are properly specified. This function is not typically needed if using `MARSS()` since `MARSS()` builds the marssm 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

The `MARSS()` function is an interface to the core functions and allows users to fit MARSS models using text strings to specify the model structure. The MARSS model takes the form

$$\mathbf{x}_t = \mathbf{B}\mathbf{x}_{t-1} + \mathbf{u} + \mathbf{e}_{t-1}, \text{ where } \mathbf{e}_{t-1} \sim \text{MVN}(0, \mathbf{Q}) \tag{3.1a}$$

$$\mathbf{y}_t = \mathbf{Z}\mathbf{x}_t + \mathbf{a} + \boldsymbol{\eta}_t, \text{ where } \boldsymbol{\eta}_t \sim \text{MVN}(0, \mathbf{R}) \tag{3.1b}$$

$$\mathbf{x}_1 \sim \text{MVN}(\pi, \mathbf{V}_1) \tag{3.1c}$$

The $\mathbf{y}$ is a $n \times T$ matrix of observations and the $\mathbf{x}$ is a $m \times T$ matrix of hidden states. For example, a $\mathbf{y}$ data matrix of 3 inputs measured for 10 time steps would look like

$$\mathbf{y} = \begin{bmatrix} 1 & 2 & -99 & -99 & 3.2 & ... & 8 \\ 2 & 5 & 3 & -99 & 5.1 & ... & 5 \\ 1 & -99 & 2 & 2.2 & -99 & ... & 7 \end{bmatrix}$$

where -99 denotes a missing value. $\mathbf{x}$ might look like (here $m = 2$):

$$\mathbf{x} = \begin{bmatrix} 0.8 & 2.2 & 3 & 2.8 & 3.2 & ... & 7.1 \\ 1.5 & 2.5 & 2.5 & 2.2 & 3.1 & ... & 6 \end{bmatrix}$$

$\mathbf{Z}$ is a $n \times m$ design matrix of zeros and ones where the row sums equal 1[1]. $\mathbf{Z}$ specifies which observation time series, $y_{i,1:T}$, is associated with which hidden state process, $x_{j,1:T}$. $\mathbf{Z}$ is like a look up table with one row for each of the n observation time series and one column for each of the m hidden processes. A "1" in row i column j means that $\mathbf{y}$ time series i is measuring the j-th $\mathbf{x}$ trajectory. Otherwise the value in $\mathbf{Z}_{ij} = 0$.

In the `MARSS()` function, the user specifies the model by passing in a parameter constraint list:

```
MARSS(data, constraint=list(Z=Z.constraint, B=B.constraint,
       U=U.constraint, Q=Q.constraint, A=A.constraint,
       R=R.constraint, x0=pi.constraint, V0=V1.constraint) )
```

[1] In our examples, $\mathbf{Z}$ is a design matrix but it can actually be any fixed matrix.

`data` must be a $n \times T$ matrix, that is time goes across columns. The argument `constraint` is a list of text strings, factors or a matrix that specifies the form of the MARSS parameters. The defaults are

`Z.constraint="identity"` each y in $\mathbf{y}$ corresponds to one x in $\mathbf{x}$
`B.constraint="identity"` no interactions between the x's in $\mathbf{x}$
`U.constraint="unequal"` the u's in $\mathbf{u}$ are all different
`Q.constraint="diagonal and unequal"` process errors are independent but have different variances
`R.constraint="diagonal and equal"` the observations are i.i.d.
`A.constraint="scaling"` $\mathbf{a}$ is a set of scaling factors
`pi.constraint="unequal"` all initial states are different
`V0.constraint="zero"` the initial condition is fixed but unknown

The other possible `constraint` options for each parameter are listed below. We show the forms using $m = 3$ (the number of hidden state processes) as an example.

3.1 Process equation constraints

3.1.1 `B.constraint`

$\mathbf{B}$ is a $m \times m$ matrix. In MARSS 1.0, $\mathbf{B}$ must be fixed.

`B.constraint="identity"` The $\mathbf{B}$ matrix is the identity matrix:

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

`B.constraint=matrix(..., nrow=m, ncol=m)` Passing in a $m \times m$ matrix, means that $\mathbf{B}$ is fixed to the values in the matrix. The matrix must be numeric and all eigenvalues must fall within the unit circle[2]. Using the string "zero", sets $\mathbf{B} = 0$.

3.1.2 u constraints

The $\mathbf{u}$ constraint has the following options:

`U.constraint="equal"` There is only $\mathbf{u}$ parameter:

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

[2] `all(abs(eigen(B)$values)>=1)`.

`U.constraint="unequal"` or `U.constraint="unconstrained"` These are equivalent. There are m **u** parameters:

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

`U.constraint=as.factor(c(...))` The **u** constraint is specified as a length m character or numeric vector of class factor. The vector of factors specifies which values in **u** are shared. `U.constraint=factor(c(1,1,2))` means that **u** has the following structure:

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

There are two **u** parameters in this case. The factor levels can be either numeric or character. `c(1,1,2)` is the same as `c("north","north","south")`.

`U.constraint=matrix(..., nrow=m, ncol=1)` Passing in a $m \times 1$ matrix, means that **u** is fixed to the values in the matrix. The matrix must be numeric. In MARSS version 1.0, **u** cannot vary in time, even if fixed. **u** can be set to all zeros (a $m \times 1$ matrix) by setting `U.constraint="zero"`; you might want to use this if you de-trended your data.
You can pass in a matrix with fixed values and NAs:

$$\begin{bmatrix} 0.1 \\ NA \\ NA \end{bmatrix}$$

`MARSS()` will interpret this as your `fixed$U` matrix. The fixed values will be set to the fixed values and the NAs will be estimated; the estimates are independent and not forced to be equal.

3.1.3 Q constraint

The **Q** constraint has the following options:

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

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

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

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

`Q.constraint="unconstrained"` There are values on the diagonal and the off-diagonals of **Q** and thevariances 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 variance parameters and $(m^2 - m)/2$ covariances in this case, so $(m^2 + m)/2$ parameters.

`Q.constraint="equalvarcov"` There is one process variance parameter and one covariance, so 2 parameters:

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

`Q.constraint=as.factor(c(...))` The **Q** constraint is specified as a length m character or numeric vector of class factor. This specifies that **Q** is diagonal and the vector of factors specifies which values on the diagonal are shared. For example, `Q.constraint=factor(c(2,1,2))` means that **Q** takes the form:

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

`Q.constraint=factor(c(1,1,2))` means that **Q** takes the form:

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

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

`Q.constraint=matrix(..., nrow=m, ncol=m)` Passing in a $m \times m$ matrix, means that **Q** is fixed to the values in the matrix. The matrix must be numeric. Note if $m = 1$, you still need to wrap its value in `matrix()` so that its class is matrix.
You can also pass in a diagonal matrix with fixed values and NAs:

$$\begin{bmatrix} NA & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & NA \end{bmatrix}$$

`MARSS()` will interpret this as your `fixed$Q` matrix. The fixed values will be set to the fixed values and the NAs will be estimated; the estimates are independent and not forced to be equal. For this case, the matrix must be diagonal.

3.1.4 π constraints

This sets the constraints on the initial conditions, $\mathbf{x}_1$. `pi.constraint` has the following options:

`pi.constraint="equal"` There is only initial state parameter.

$$\begin{bmatrix} \pi \\ \pi \\ \pi \end{bmatrix}$$

Warning: specifying shared π values will tend to produce ill-conditioned matrices in the algorithm and lead to numerical instability. Avoid using `pi.constraint="equal"` if possible.

`pi.constraint="unequal"` or `pi.constraint="unconstrained"` These are equivalent. There are m initial state parameters.

$$\begin{bmatrix} \pi_1 \\ \pi_2 \\ \pi_3 \end{bmatrix}$$

`pi.constraint=factor(c(...))` The initial states constraint is specified is a length m character or numeric vector of class factor. The vector of factors specifies which initial states have the same value. For example, `pi.constraint=factor(c(1,1,2))` means that the initial states have the following structure:

$$\begin{bmatrix} \pi_1 \\ \pi_1 \\ \pi_2 \end{bmatrix}$$

There are two initial state parameters in this case. The factor levels can be either numeric or character. `c(1,1,2)` is the same as `c("n","n","s")`. Warning: specifying shared π values will tend to produce ill-conditioned matrices in the algorithm and lead to numerical instability. Avoid if possible.

`pi.constraint=matrix(..., nrow=m, ncol=1)` Passing in a $m \times 1$ matrix, means that the initial states are fixed to the values in the matrix. The matrix must be numeric. You can set the initial states to zero by using `pi.constraint="zero"`. You can also pass in a matrix with fixed values and NAs:

$$\begin{bmatrix} 10 \\ NA \\ NA \end{bmatrix}$$

`MARSS()` will interpret this as your `fixed$x0` matrix. The fixed values will be set to the fixed values and the NAs will be estimated.

3.1.5 $\mathbf{V}_1$ constraints

The initial state variance must be fixed. The default behavior is to treat $\mathbf{x}_1$ as fixed but unknown[3]. In this case, $\mathbf{V}_1$=0. You can also set $\mathbf{V}_1$ to a non-zero value but in that case $\boldsymbol{\pi}$ must be fixed (meaning `x0.constraint` is passed in as a fixed matrix). This would translate to using fixed prior for the initial states.

`V0.constraint=matrix(..., nrow=m, ncol=m)` Passing in a $m \times m$ matrix, means that the initial state variance is fixed to the values in the matrix. The matrix must be numeric and be a proper variance-covariance matrix.

In general, unless a fixed prior is desired, users should just leave off `x0.constraint` and `V0.constraint` from the constraint list and use the default behavior which is $\mathbf{x}_1$ treated as fixed but unknown.

3.2 Observation equation constraints

3.2.1 Z constraint

In the `MARSS()` function, $\mathbf{Z}$ is normally a $n \times m$ design matrix that specifies which x_i hidden state time series corresponds to which y_j time series[4]. In this case, each y_j time series (each row in $\mathbf{y}$) corresponds to one and only one x_i time series (row in $\mathbf{x}$). The $\mathbf{Z}$ constraint is normally specified as a length n vector of class factor. The i-th element of this vector specifies which population trajectory the i-th observation time series belongs to. Below are some examples of $\mathbf{Z}$ matrices; see Chapter 6 and the case studies chapters for more examples.

`Z.constraint=factor(c(1,1,1))` All $\mathbf{y}$ time series are observing the same (and only) hidden state trajectory x. Thus $n = 3$ and $m = 1$:

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

`Z.constraint=factor(c(1,2,3))` Each $\mathbf{y}$ time series corresponds to a different hidden state trajectory. The is the default $\mathbf{Z}$ constraint and in this case $n = m$:

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

[3] Shumway and Stoffer use $\mathbf{x}$ at $t = 0$ as the initial state, but we follow Gharamani's approach and use $\mathbf{x}$ at $t = 1$ as the initial state. Both approaches give the same answer but the EM update equations are very slightly different.

[4] However, you can pass in other fixed but non-design $\mathbf{Z}$ matrices.

`Z.constraint=factor(c(1,1,2))` The first two **y** time series 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** constraint can be specified using either numeric or character factor levels. `c(1,1,2)` is the same as `c("north","north","south")`

`Z.constraint="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.constraint=factor(c(1,2,3))`.

`Z.constraint=matrix(..., nrow=n, ncol=m)` Passing in a $n \times m$ matrix, means that **Z** is fixed to the values in the matrix. The matrix must be numeric. **Z** does not need to be a design matrix.

3.2.2 a constraint

Only `A.constraint="scaling"` or a fixed **a** is allowed. If "scaling", **a** is a scaling factor where one of the y associated with each x is set to zero and the rest are estimated. Unless you know the correct scaling, because say you simulated the data or you de-meaned the data, you should use the default. To set **a** to a fixed value (not estimated), use `A.constraint=matrix(..., nrow=n, ncol=m)`. The matrix must be numeric. To fix **a** to zero, use `A.constraint="zero"`.

Note, you can circumvent the restriction on **a** by passing in a matrix with NAs (for estimated elements) or passing in a `fixed/free` pair. But be aware that it is very easy to make your model unsolveable (an infinite number of solutions)—if for example you try to estimate more that $n_i - 1$ a's, where n_i is the number of y's for one x. If **Z** is identity, then $n_i = 1$ and you can estimate no a's; all must be fixed.

3.2.3 R constraint

The **R** constraint is completely analogous to the **Q** constraint, except that it is $n \times n$ instead of $m \times m$. Its allowable constraints are affected by the presence of missing data points in **y**. If data are missing, then **R** must be diagonal.

4

Model specification in the core functions

Most users will not directly work with the core functions nor build marssm objects from scratch. Instead, they will usually interact with the core functions via the function `MARSS()` described in chapter 3. With the `MARSS()` function, the user specifies the model structure with text strings ("diagonal", "unconconstrained", etc.) and `MARSS()` builds the marssm object. However, a basic understanding of the structure of marssm objects is necessary if one wants to fit more flexible models or to interact directly with the core functions.

The first step of model specification is to write down (e.g. on paper) the model in matrix form (Equation 1.1) with notes on the dimensions (rows and columns) of each parameter and for $\mathbf{x}$ and $\mathbf{y}$. In the core functions, the parameters in the MARSS model must be passed as matrices of the correct dimension, and the parameters in the R functions correspond one-to-one to the mathematical equation. For example, $\mathbf{u}$ must be passed in as a matrix of dimension `c(m,1)`. The function will return an error if anything else is passed in (including a matrix with `dim=c(1,m)` or a vector of length m).

4.1 The fixed and free components of the model parameters

In a marssm object, each parameter must be specified by a pair of matrices: `free` which gives the location and sharing of the estimated elements in the parameter matrix and `fixed` which specifies the location and value of the fixed elements in the parameter matrix. For example, $\mathbf{Q}$ is specified by `free$Q` and `fixed$Q`.

The fixed matrix specifies the values (numeric) of the fixed (meaning not estimated) elements. In the fixed matrix, the free (meaning estimated or fitted) elements are denoted with `NA`. The following shows some common examples of the fixed matrix using `fixed$Q` as the example. Each of the other fixed matrices for the other parameters uses the same pattern.

- **Q** is unconstrained, so there are no fixed values

$$\texttt{fixed\$Q} = \begin{bmatrix} NA & NA & NA \\ NA & NA & NA \\ NA & NA & NA \end{bmatrix}$$

- **Q** is a diagonal matrix, so the off-diagonals are fixed at 0. The diagonal elements will be estimated.

$$\texttt{fixed\$Q} = \begin{bmatrix} NA & 0 & 0 \\ 0 & NA & 0 \\ 0 & 0 & NA \end{bmatrix}$$

- **Q** is fixed, i.e. will not be estimated rather all values in the **Q** matrix are fixed.

$$\texttt{fixed\$Q} = \begin{bmatrix} 0.1 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 0.1 \end{bmatrix}$$

The free matrix specifies which elements are estimated and specifies how (and whether) the free elements are shared. In the free matrix, the fixed elements are denoted `NA`. The following shows some common examples of `free` using `free$Q` as the example. `free` can be passed in as a character matrix or a numeric matrix, but if numeric, the numeric will be changed to character (thus 0 becomes "0" and is the name "0" not the number 0).

- **Q** is a diagonal matrix in which there is only one, shared, value on the diagonal. Thus there is only one **Q** parameter.

$$\texttt{free\$Q} = \begin{bmatrix} 1 & NA & NA \\ NA & 1 & NA \\ NA & NA & 1 \end{bmatrix} \quad \texttt{or} \quad \begin{bmatrix} ``a" & NA & NA \\ NA & ``a" & NA \\ NA & NA & ``a" \end{bmatrix}$$

  Here "1" does not mean "number 1" but rather that the name of the shared parameter is "1". On the example at the right, the name of the shared parameter is "a".
- **Q** is a diagonal matrix in which each of the diagonal elements are different.

$$\texttt{free\$Q} = \begin{bmatrix} 1 & NA & NA \\ NA & 2 & NA \\ NA & NA & 3 \end{bmatrix} \quad \texttt{or} \quad \begin{bmatrix} ``north" & NA & NA \\ NA & ``middle" & NA \\ NA & NA & ``south" \end{bmatrix}$$

- **Q** has one value on the diagonal and another one on the off-diagonals. There are no fixed values in **Q**.

$$\texttt{free\$Q} = \begin{bmatrix} 1 & 2 & 2 \\ 2 & 1 & 2 \\ 2 & 2 & 1 \end{bmatrix} \quad \texttt{or} \quad \begin{bmatrix} ``a" & ``b" & ``b" \\ ``b" & ``a" & ``b" \\ ``b" & ``b" & ``a" \end{bmatrix}$$

- **Q** is unconstrained. There are no fixed values in **Q** in this case. Note that since, **Q** is a variance-covariance matrix, it must be symmetric across the diagonal.

$$\texttt{free\$Q} = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 3 & 5 & 6 \end{bmatrix}$$

4.2 Limits on the model forms that can be fit by MARSS 1.0

MARSS version 1.0 will allow any combination of fixed and shared values in **a** and **u**, **R**, **Q**, **B**, and **Z** there are limits to what forms these matrices can take. These limitations have to do with the way the EM algorithm is coded for version 1.0. Version 2.0 will remove many of these restrictions.

- **R** and **Q** can be fixed, unconstrained, diagonal with any pattern of fixed and shared values on the diagonal, or a matrix with one estimated value on the diagonal and another estimated value on the off-diagonals (an "equal var-cov" matrix).
- If there are missing values in the data, **R** must be diagonal (or fixed).
- **B** cannot be estimated. It must be fixed but it need not be diagonal. However, all its eigen values must fall on the unit circle[1].
- **Z** cannot be estimated. It must be fixed. Although in all our examples, **Z** is a design matrix (meaning only 0s and 1s and all row sums equal to 1), **Z** is not required to be a design matrix.

The other limitation is that one must specify a model that has only one solution. The core MARSS functions will allow you to attempt to fit an improper model (one with multiple solutions). If you do this accidentally, it may or may not be obvious that you have a problem. The MARSS estimation functions may chug along happily and return a solution. Careful thought about your model structure and the structure of the estimated parameter matrices will help you determine if your model is under-constrained and unsolvable. Basically, take care when using MARSS core functions directly and remember that it will not prevent you from fitting an under-constrained model. This is not a problem when using the function `MARSS()`. The `MARSS()` function includes error-checks that are designed to prevent users from specifying an under-constrained models (for example it only allows **a** to be fixed or act like a scaling factor.

[1] This means that the absolute value of all the eigen values must be less than or equal to 1 (`all(abs(eigen(B)$values)<=1)`).

5

Algorithms used in the MARSS package

5.1 Kalman filter and smoother

The MARSS model is a linear dynamical system with discrete time and Gaussian errors. 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: $E(\mathbf{X}_t|\mathbf{y}_1^t)$. 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: $E(\mathbf{X}_t|\mathbf{y}_1^T)$. 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 reference and adopt their notation (for the most part).

The `MARSSkf` function provides the following Kalman filter and smoother 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 from time 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`.

`J` This is similar to the Kalman gain but is part of the Kalman smoother. See Equation 6.49 in Shumway and Stoffer (2006).

`Innov` This has the innovations at time t, defined as $\boldsymbol{\epsilon}_t \equiv \mathbf{y}_t$-$E(\mathbf{Y}_t)$. These are the residuals, the difference between the data and their predicted values. See Equation 6.24 in Shumway and Stoffer (2006).

`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.

`logLik` The log likelihood of the data conditioned on the model parameters. See the section below on the likelihood calculation.

5.2 The exact likelihood

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

$$\log L(\Theta|data) = -\frac{N}{2\log(2\pi)} - \frac{1}{2}\left(\sum_{t=1}^{T}\log|\Sigma_t| + \sum_{t=1}^{T}(\boldsymbol{\epsilon}_t)^\top \Sigma_t^{-1}\boldsymbol{\epsilon}_t\right) \tag{5.1}$$

where N is the total number of data points and $|\Sigma_t|$ is the determinant of the innovations variance-covariance matrix. Reference equation 6.62 in Shumway and Stoffer (2006). However there are a few differences between the log likelihood output by `MARSSkf` 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. 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 $\boldsymbol{\epsilon}_t$ and Σ_t in the Equation 5.1. Suppose the value $y_{i,t}$ is missing. First, the corresponding i-th value of $\boldsymbol{\epsilon}_t$ 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.

5.3 Maximum-likelihood parameter estimation

5.3.1 Kalman-EM algorithm

The MARSS package provides a maximum-likelihood algorithm which uses an Expectation-Maximization (EM) algorithm (function `MARSSkem`) with the Kalman smoother. 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, $\hat{\Theta}$, 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 Θ that maximizes the expected value of the likelihood over the distribution of the states ($\mathbf{X}$) conditioned on $\hat{\Theta}_1$"

$$\hat{\Theta}_2 = \arg\max_{\Theta} \quad \mathrm{E}_{\mathbf{X}|\hat{\Theta}_1}[\log L(\Theta|\mathbf{Y}_1^T = \mathbf{y}_1^T, \mathbf{X})] \tag{5.2}$$

Then using $\hat{\Theta}_2$ in place of $\hat{\Theta}_1$ in Equation (5.2), 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 actually fairly 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.
3. M step: put those $\mathrm{E}(\mathbf{X}|\mathbf{Y}_1^T = \mathbf{y}_1^T, \hat{\Theta}_1)$ and $\mathrm{E}(g(\mathbf{X})|\mathbf{Y}_1^T = \mathbf{y}_1^T, \hat{\Theta}_1)$ 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

[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.

The EM equations in our algorithm, which we term the Kalman-EM algorithm, are extensions of those in Shumway and Stoffer (1982) and Ghahramani and Hinton (1996). Our Kalman-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, ...) and where there are fixed values within the parameter matrices. Holmes (2010) gives the full derivation of our EM algorithm.

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. Quasi-Newton methods find the absolute maximum much faster, but they can be sensitive to initial conditions. We have found that with MARSS models, quasi-Newton methods (at least using `optim`) will sometimes converge far from the maximum even when started close to the known maximum. For this reason, the Monte Carlo initial condition search that works for EM algorithms may not work for Newton algorithms.

5.4 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 does both parametric and innovations bootstrapping of MARSS models.

The innovations bootstrap essentially bootstraps the residuals after model-fitting (called innovations) and uses the algorithm by Stoffer and Wall (1991). 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` also allows one to construct approximate confidence intervals by bootstrapping from a numerically estimated Hessian matrix.

5.5 Simulation and forecasting

The `MARSSsimulate` function simulates from a MARSS model using a list of parameter matrices. It use the `mvrnorm` function to produce draws of the process and observation errors from multivariate normal distributions for each time step.

5.6 Model selection

The package provides a `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-parameteric 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 bootstraped maximum-likelihood parameter estimates and the log likelihood of the data under the original maximum-likelihood parameter estimate:

$$AICb = -2 \log L(\hat{\Theta}|\mathbf{y}) + 2\left(\frac{1}{N_b}\sum_{i=1}^{N_b} - \log \frac{L(\hat{\Theta}^*(i)|\mathbf{y})}{L(\hat{\Theta}|\mathbf{y})}\right) \tag{5.3}$$

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 $L(\hat{\Theta}^*|\mathbf{y})$ not $L(\hat{\Theta}^*|\mathbf{y}^*)$. In other words, we are taking the average of the likelihood of the original data given the bootstrapped parameter sets.

6
Examples

Here we show a series of quick examples with little explanatory text. Chapters 8–11 give case studies which walk through detailed multi-level analyses. The examples in this chapter use the Washington harbor seal dataset (?harborSealWA), which has five observation time series. First set up the data:

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

6.1 Fit different MARSS models to a dataset

6.1.1 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 $\mathbf{u}$ are all different (meaning, they are not forced to be the same). Mathematically, the MARSS model being fit is:

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

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

To fit this model, use `MARSS()`. The function will output the basic model structure and the time to convergence.

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

```
abstol reached in 28 iterations and parameters appear converged.
Alert: with less than 100 iterations, the convergence diagnostics will be uncertain.

MARSS fit is
Estimation method:  kem
Estimation converged in 28 iterations.
Log-likelihood: 21.84028
AIC: -11.68055   AICc: -1.606480

     Estimate
Q.1   0.03353
Q.2   0.01175
Q.3   0.00765
Q.4   0.00554
Q.5   0.06303
R.1   0.00948
U.1   0.06839
U.2   0.07164
U.3   0.04182
U.4   0.05241
U.5  -0.00271
x0.1  6.05542
x0.2  6.79635
x0.3  6.70269
x0.4  5.88259
x0.5  6.60160

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

The default method is the Kalman-EM algorithm (`method="kem"`). You can use a quasi-Newton method (BFGS) by setting `method="BFGS"`. The quasi-Newton method can be a bit fragile. In fact, the BFGS method will generate numerical errors for this example.

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

```
Stopped with errors: No parameter estimates returned. See $errors in output for details.
MARSSkf returned errors.  Sometimes better initial conditions will solve this.
 Error in optim(pars, neglogLik, MLEobj = tmp.MLEobj, method = MLEobj$method,  :
  objective function in optim evaluates to length 0 not 1
```

If you wanted to use the Kalman-EM fit as the initial conditions, pass in the `inits` argument.

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

Output not shown, but it also generates numerical errors.

6.1.2 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:

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

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

To fit, use `MARSS()` with the constraint argument set. The output is not shown. Type `print(kemfit)` to see the fit.

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

6.1.3 Five equally correlated hidden state processes

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

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

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

To fit, use

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

To see the parameter estimates, use

```
print(kemfit)
```

```
MARSS fit is
Estimation method:  kem
Estimation converged in 98 iterations.
Log-likelihood: 31.31512
AIC: -36.63025   AICc: -30.24428

     Estimate
Q.2   0.00880
Q.1   0.00847
R.1   0.01735
U.1   0.06942
U.2   0.07860
U.3   0.04061
U.4   0.05074
U.5  -0.00909
x0.1  6.08013
x0.2  6.75781
x0.3  6.70508
x0.4  5.79460
x0.5  6.61629

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

6.1.4 Five hidden state processes with a "north" and a "south" u parameter

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 parameters, while hidden trajectories 4-5 share theirs. This is the model:

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

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

To fit use:

```
regions=factor(c("N","N","N","S","S"))
kemfit = MARSS(dat, constraint=list(U=regions, Q=regions))
```

```
algorithm run for 15 iterations, abstol was reached, and parameters appear converged.
Alert: with less than 100 iterations, the convergence diagnostics will be uncertain.

MARSS fit is
Estimation method:  kem
Estimation converged in 15 iterations.
Log-likelihood: 16.36329
AIC: -12.72658   AICc: -9.059918

     Estimate
Q.N    0.0120
Q.S    0.0253
R.1    0.0143
U.N    0.0609
U.S    0.0243
x0.1   6.1171
x0.2   6.8247
x0.3   6.6811
x0.4   5.8626
x0.5   6.5990

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

6.1.5 Fixed observation error variance

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

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

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

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

To fit this model:

```
regions=factor(c("N","N","N","S","S"))
kemfit = MARSS(dat, constraint=list(U=regions, Q=regions,
          R=diag(0.01,5)))
```

```
algorithm run for 15 iterations, abstol was reached, and parameters appear converged.
Alert: with less than 100 iterations, the convergence diagnostics will be uncertain.

MARSS fit is
Estimation method:  kem
Estimation converged in 15 iterations.
Log-likelihood: 16.86671
AIC: -15.73341   AICc: -12.78259

     Estimate
Q.N    0.0151
Q.S    0.0332
U.N    0.0607
U.S    0.0247
x0.1   6.0845
x0.2   6.7944
x0.3   6.6602
x0.4   5.8439
x0.5   6.5983
```

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

6.1.6 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:

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

$$\begin{bmatrix} y_{1,t} \\ y_{2,t} \\ y_{3,t} \\ y_{4,t} \\ y_{5,t} \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} x_t + \begin{bmatrix} 0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} + \begin{bmatrix} w_{1,t} \\ w_{2,t} \\ w_{3,t} \\ w_{4,t} \\ w_{5,t} \end{bmatrix},\ \mathbf{w}_t \sim \mathrm{MVN}\left(0, \begin{bmatrix} r\,0\,0\,0\,0 \\ 0\,r\,0\,0\,0 \\ 0\,0\,r\,0\,0 \\ 0\,0\,0\,r\,0 \\ 0\,0\,0\,0\,r \end{bmatrix}\right)$$

Note the default constraint for R is "diagonal and equal" so we can leave this off when specifying the `constraint` argument. This model is fit with the control argument `minit` to force the algorithm to run for 100 iteration; this allows good convergence diagnostics to be computed.

```
 kemfit =
    MARSS(dat, constraint=list(Z=factor(c(1,1,1,1,1))),
     control=list(minit=100))
```

```
Success! abstol reached at 101 iterations and parameters converged.

MARSS fit is
Estimation method:  kem
Estimation converged in 101 iterations.
Log-likelihood: 3.837584
AIC: 8.324832   AICc: 10.64741

     Estimate
A.2    0.7987
A.3    0.2796
A.4   -0.5509
A.5   -0.6296
Q.1    0.0048
R.1    0.0455
U.1    0.0475
x0.1   6.4394

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

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

Mathematically, this model is:

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

$$\begin{bmatrix} y_{1,t} \\ y_{2,t} \\ y_{3,t} \\ y_{4,t} \\ y_{5,t} \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} x_t + \begin{bmatrix} 0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} + \begin{bmatrix} w_{1,t} \\ w_{2,t} \\ w_{3,t} \\ w_{4,t} \\ w_{5,t} \end{bmatrix},\ \mathbf{w}_t \sim \mathrm{MVN}\left(0, \begin{bmatrix} r_1 & 0 & 0 & 0 & 0 \\ 0 & r_2 & 0 & 0 & 0 \\ 0 & 0 & r_3 & 0 & 0 \\ 0 & 0 & 0 & r_4 & 0 \\ 0 & 0 & 0 & 0 & r_5 \end{bmatrix}\right)$$

To fit this model:

```
kemfit =
   MARSS(dat, constraint=list(Z=factor(c(1,1,1,1,1)),
   R="diagonal and unequal"),control=list(minit=100))
```

```
Success! algorithm run for 100 iterations, abstol reached and parameters converged.

MARSS fit is
Estimation method:  kem
Estimation converged in 100 iterations.
Log-likelihood: 17.14767
AIC: -10.29534   AICc: -4.916028

     Estimate
A.2   0.79875
A.3   0.27858
A.4  -0.53217
A.5  -0.60336
Q.1   0.00672
R.1   0.03225
R.2   0.03543
R.3   0.01338
R.4   0.01098
R.5   0.19630
U.1   0.05263
x0.1  6.31277

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

6.1.8 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 process independent 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:

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

$$\begin{bmatrix} y_{1,t} \\ y_{2,t} \\ y_{3,t} \\ y_{4,t} \\ y_{5,t} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_{n,t} \\ x_{s,t} \end{bmatrix} + \begin{bmatrix} 0 \\ a_1 \\ a_2 \\ 0 \\ a_3 \end{bmatrix} + \begin{bmatrix} w_{1,t} \\ w_{2,t} \\ w_{3,t} \\ w_{4,t} \\ w_{5,t} \end{bmatrix}, \ \mathbf{w}_t \sim \text{MVN}\left(0, \begin{bmatrix} r & 0 & 0 & 0 & 0 \\ 0 & r & 0 & 0 & 0 \\ 0 & 0 & r & 0 & 0 \\ 0 & 0 & 0 & r & 0 \\ 0 & 0 & 0 & 0 & r \end{bmatrix}\right)$$

To fit the model:

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

```
abstol reached in 16 iterations and parameters appear converged.
Alert: with less than 100 iterations, the convergence diagnostics will be uncertain.

MARSS fit is
Estimation method:  kem
Estimation converged in 16 iterations.
Log-likelihood: 8.446231
AIC: -0.892463   AICc: 1.430118

     Estimate
A.2   0.79399
A.3   0.27323
A.5  -0.06798
Q.1   0.00927
R.1   0.03430
U.1   0.04309
x0.1  6.20472
x0.2  6.24855

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

6.2 Show a summary of the model being fit

```
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 "g#", meaning "group #". Each shared element is a group. For example, a model with a diagonal **Q** matrix with one variance would be printed as a diagonal matrix with "g1" on all the diagonal elements.

6.3 Compute confidence intervals on a fitted model

The function `MARSSparamCIs()` is used to compute confidence intervals. 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.

6.3.1 Approximate confidence intervals from an estimated Hessian matrix

```
#default uses an est 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
Estimation converged in 16 iterations.
Log-likelihood: 8.446231
AIC: -0.892463   AICc: 1.430118

       ML.Est Std.Err   low.CI  up.CI
A.2   0.79399 0.06174  0.67299 0.9150
A.3   0.27323 0.06277  0.15021 0.3963
A.5  -0.06798 0.08924 -0.24289 0.1069
Q.1   0.00927 0.00533 -0.00118 0.0197
R.1   0.03430 0.00668  0.02119 0.0474
U.1   0.04309 0.01567  0.01237 0.0738
x0.1  6.20472 0.11398  5.98132 6.4281
x0.2  6.24855 0.12839  5.99691 6.5002

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

6.3.2 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
Estimation converged in 16 iterations.
Log-likelihood: 8.446231
AIC: -0.892463   AICc: 1.430118

       ML.Est Std.Err  low.CI  up.CI  Est.Bias Unbias.Est
A.2   0.79399 0.05420  0.7024 0.8570  0.013072    0.80706
A.3   0.27323 0.06472  0.2133 0.4035 -0.022195    0.25103
A.5  -0.06798 0.10126 -0.2496 0.0376  0.020571   -0.04740
Q.1   0.00927 0.00531  0.0009 0.0157  0.000636    0.00991
R.1   0.03430 0.00632  0.0205 0.0389  0.003940    0.03824
U.1   0.04309 0.01660  0.0199 0.0656 -0.000401    0.04268
x0.1  6.20472 0.19944  5.9232 6.4940 -0.012260    6.19246
x0.2  6.24855 0.17061  6.0241 6.5384 -0.109624    6.13893

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

6.4 Work with 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.

```
parvec=MARSSvectorizeparam(kemfit)
parvec
```

```
        A.2          A.3          A.5          Q.1
0.793991591  0.273229103 -0.067975309  0.009274083
        R.1          U.1         x0.1         x0.2
0.034295919  0.043085650  6.204717299  6.248551859
```

If you want to replace the estimated parameter values with different values, you can use the same function:

```
parvec.new=parvec
parvec.new["Q.1"]=0.02; parvec.new["U.1"]=0.05
kem.new=MARSSvectorizeparam(kemfit, parvec.new)
```

Then you might want to find out the likelihood of the data using those new parameter values. You compute that with the Kalman filter function `MARSSkf()`, sending it the data and the parameters as a list.

```
kf=MARSSkf(dat, kem.new$par, miss.value=-99)
kf$logLik
```

```
         [,1]
[1,] 6.851116
```

6.5 Determine which variance elements have not converged

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

```
dat.short = dat[,1:10]
kem.degen = MARSS(dat.short, silent=2)
```

```
abstol reached at 48 iterations but some parameters have not converged.
```

This will print a short warning that one of the variance elements has not converged (silent=2 means brief messages). Type `cat(kem.degen$errors)` to see the warnings. It may be that your tolerance is too high and if you just ran a few more iterations thenthe the variances will converge. So first try setting `control$minit` high and see if the warning goes away. Setting `minit` forces a minimum number of iterations. 200 should be enough.

```
kem.200 = MARSS(dat.short, control=list(minit=200), silent=2)
```

```
abstol reached at 200 iterations but some parameters have not converged.
```

The problem still does not go away. Type `cat(kem.200$errors)` to see which parameter(s) is not converging or you can use `find.degenerate()` to plot the log parameter value against the log iteration number to figure out which parameter is not converged. Use `kem.200` because this plot needs at least 100 iterations to look good. We can try a few solutions. First perhaps we are just trying to estimate too many variances. We can try using only one **Q** variance and one **U** parameter:

```
kem.small=MARSS(dat.short,constraint=list(Q="diagonal and equal",
   U="equal"),control=list(minit=200),silent=2)
```

```
abstol reached at 200 iterations but some parameters have not converged.
```

```
find.degenerate(kem.200)
```

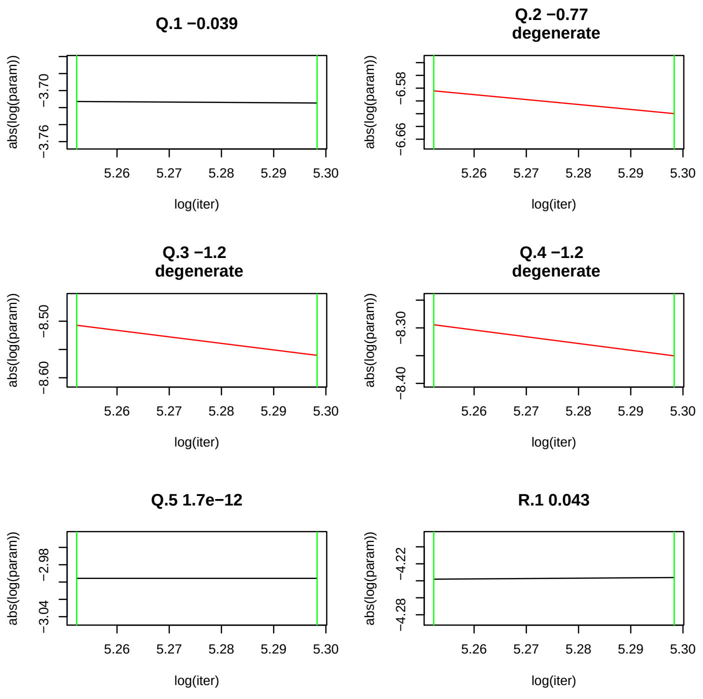

Fig. 6.1. A diagnostic plot showing which diagonal variance element is not converging. This is a log-log plot of iteration versus the log of the variance estimate. It should be flat. Diagonal parameter elements that are not flat are shown in red.

No, we still get the degenerate warning. There are simply not enough data to estimate both process and observation variances. For the EM algorithm, you cannot set **Q** to a fixed small value because then **U** would converge very, very slowly simply because of the nature of the **U** update equations. However, you can try a quasi-Newton method with **Q** fixed small. Here we use one **Q** variance and fix it small.

```
kem.bfgs.degen=MARSS(dat.short,constraint=list(
   Q=diag(1E-12,5),U="equal"),method="BFGS")
```

```
Success! Converged in 33 interations.

MARSS fit is
Estimation method:  BFGS
Estimation converged in 33 iterations.
Log-likelihood: 11.90747
```

```
AIC: -9.814944   AICc: -1.199559

     Estimate
R.1    0.0239
U.1    0.1025
x0.1   6.1650
x0.2   6.8738
x0.3   6.6347
x0.4   5.8492
x0.5   6.5958

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

We could also just set the degenerate variances, the 2nd, 3rd and 4th elements on the diagonal of **Q**, to a small value and estimate the 1st and 5th elements on the diagonal:

```
 kem.bfgs.degen=MARSS(dat.short,constraint=list(
    Q=diag(c(NA,1E-12,1E-12,1E-12,NA),5),U="equal"),method="BFGS")
```

```
Success! Converged in 31 interations.

MARSS fit is
Estimation method:  BFGS
Estimation converged in 31 iterations.
Log-likelihood: 12.73636
AIC: -7.472718   AICc: 8.890919

     Estimate
Q.1    0.0220
Q.25   0.0500
R.1    0.0162
U.1    0.1024
x0.1   6.0625
x0.2   6.8746
x0.3   6.6354
x0.4   5.8496
x0.5   6.5958

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

Note that this gets us close to the correct likelihood, but is not quite right since really the degenerate diagonal elements should equal 0, but you cannot set them to zero because the likelihood calculation in `MARSSkf` would throw an error.

6.6 Generate bootstrap parameter estimates

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

```
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 estimates from a Hessian matrix (`sim="hessian"`) or a non-parametric bootstrap (`sim="innovations"`).

6.7 Simulate data

6.7.1 Simulated data from a fitted MARSS model

You can easily simulate data from a fitted model using `MARSSsimulate()`.

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

```
MARSSkf(sim.data[,,2], kem.sim.1$par, miss.value = -99)$logLik
```

```
          [,1]
[1,] -30.26865
```

There are no missing values in our simulated data, but we still need to pass `miss.value` into `MARSSkf()`.

6.7.2 Simulated data from a user-built MARSS model

This shows you how to build up a model from scratch and simulate from that using `MARSSsimulate()`.

```
nsim = 20              # number of simulations
burn = 10              # length of burn in period
tSteps = 25
m=3                    #number of hidden state trajectories
B = diag(1,m);
A = array(0, dim=c(m,1))
Z = diag(1,m)
U = array(0.01, dim=c(m,1))
Q = diag(0.3, m)       #independent process errors
R = diag(0.01, m)      #independent obs errors
x0 = array(10, dim=c(m,1)) #initial conditions, really x_1
V0 = array(0,dim=c(m,m))   #leave this 0
the.par.list =
    list(Z=Z, A=A, R=R, B=B, U=U, Q=Q, x0=x0, V0=V0 )
# simulate data
sim = MARSSsimulate(the.par.list,nsim=nsim,tSteps=burn+tSteps)
# take off the burn
obs = sim$sim.data[,(burn+1):(burn+tSteps),] #m x T x nsim
```

6.7.3 Correlation between estimated parameters

We can use a `for` loop along with `MARSSvectorizeparam()` to estimate parameters for each of the `nsim` simulated datasets in `obs`, and then assemble the estimates into a matrix.

```
for(i in 1:nsim){
dat=obs[,,i]
kem.sim=MARSS(dat, silent=TRUE)
if(i==1) par.sim=MARSSvectorizeparam(kem.sim)
else par.sim=rbind(par.sim, MARSSvectorizeparam(kem.sim))
}
```

Then we use `pairs()` to get a quick visual look at how the parameters are correlated (or not) and how variable they are (Figure 6.2).

We could also use `MARSSboot()` to do this. However, `MARSSboot()` is designed to take a marssMLE object (such as would be returned from `MARSSkem`). To use `MARSSboot`, we need to make a marssMLE object which has information on initial conditions and maximization options. An easy way to do this is to use `MARSS` with `fit=FALSE`. This will return a correctly formed marssMLE object with no estimated parameters. The only difference between this method and the above is that `MARSSboot` uses a single start condition for the maximization search while the approach above will compute a different start for each simulated data set. They should give the same answers (assuming they are not finding local minima) but the computation speeds and convergence times may differ.

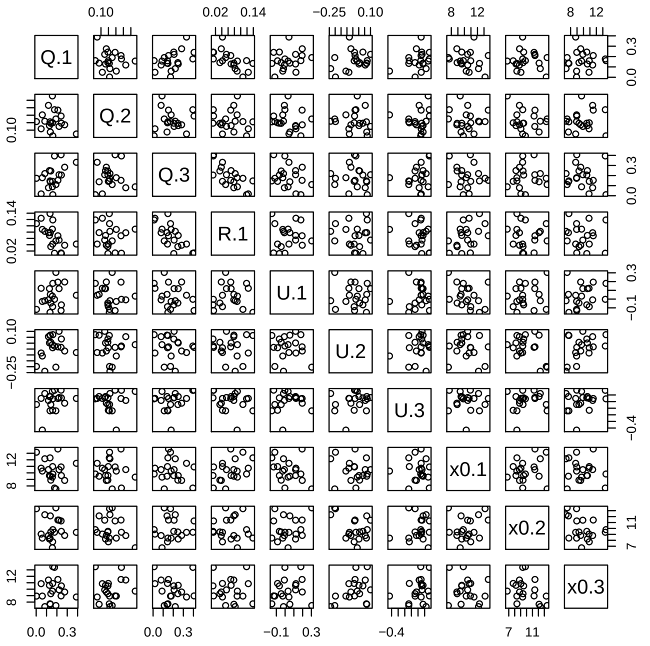

Fig. 6.2. The parameter pairs plotted against each other using the command `pairs(par.sim)`.

```
#generate some data just to fit with MARSS to make marssMLE obj
tmp = MARSSsimulate(the.par.list, nsim=1, tSteps=tSteps)$sim.data
#dim of sim.data is n x T x nsim
tmp = tmp[,,1]
#use fit FALSE to just return an marssMLE obj
sim.marssMLE = MARSS(tmp, silent=TRUE, fit=FALSE)
sim.marssMLE$par = the.par.list
par.sim = MARSSboot(sim.marssMLE,
     nboot=20, output="parameters", sim="parametric")$boot.params
#boot.paras is #params x nboot; change the dim order
par.sim=t(par.sim)
```

We could also build the marssMLE object from scratch, by making a list with elements `par`, `start`, and `control`. The first is `the.par.list` above and the latter are used by the maximization routine (see `?marssMLE`). If you want a non-parameteric bootstrap, use `sim="innovations"`.

6.8 Compute bootstrap AIC for a model

This computes a parametric bootstrap AIC for the model `kemfit` using the function `MARSSaic()`. Use `output="AICbb"` to produce a non-parameter 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
Estimation converged in 16 iterations.
Log-likelihood: 8.446231
AIC: -0.892463   AICc: 1.430118   AICbp(param): 11.64734

     Estimate
A.2   0.79399
A.3   0.27323
A.5  -0.06798
Q.1   0.00927
R.1   0.03430
U.1   0.04309
x0.1  6.20472
x0.2  6.24855

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

7

Case studies: 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. Type from the command line: (Windows) `install.packages("MARSS.zip", repos = NULL)` or (Mac/Unix) `install.packages("MARSS.tar.gz", repos = NULL)`. Mac users need Xtools installed for this to work. 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. Now you should be ready.
- Each case study comes with an associated script file. To open up a copy of the case study script with the code you need to do the exercises, type `show.doc(MARSS, Case_study_#.R)` (with # 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 (5000). If it says your model variances did not converge, try running with `control=list(minit=...)` set higher (to say 100 or 200).

- 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 `constraint$Z` must be m, the number of x hidden state trajectories in your model.
- If you are fitting to population counts, your data must be logged (base e) before being passed in. The default missing value indicator is -99. You can change that by passing in `miss.value=...`.
- 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.

8

Case Study 1: Count-based PVA for data with observation error

8.1 The Problem

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 use the Kalman filter to fit a univariate (meaning one time series) state-space model to count data for a population. 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 σ^2, the process error variance. However this difference has a large effect on our risk estimates, as you will see.

In this case study, we use a density-independent model, which is the same as the Gompertz model (3.1) with $\mathbf{B} = 1$. Density-independence is often a reasonable assumption when doing a PVA 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 PVA, 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:

$$x_t = x_{t-1} + u + v_t \qquad \text{where } v_t \sim \mathrm{N}(0, \sigma^2) \tag{8.1}$$

$$y_t = x_t + w_t \qquad \text{where } w_t \sim \mathrm{N}(0, \eta^2) \tag{8.2}$$

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 $\mathbf{Q}$ and $\mathbf{R}$ for the respective variance-covariance matrices.

8.2 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'll used a decline of 5% per year, process variability of 0.01 (typical for big mammals), 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's the code:

First, set things up:

```
sim.u = -0.05          # growth rate
sim.Q = 0.01           # process error variance
sim.R = 0.05           # non-process error variance
nYr= 30                # 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 8.1:

```
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 8.2 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]=-99
```

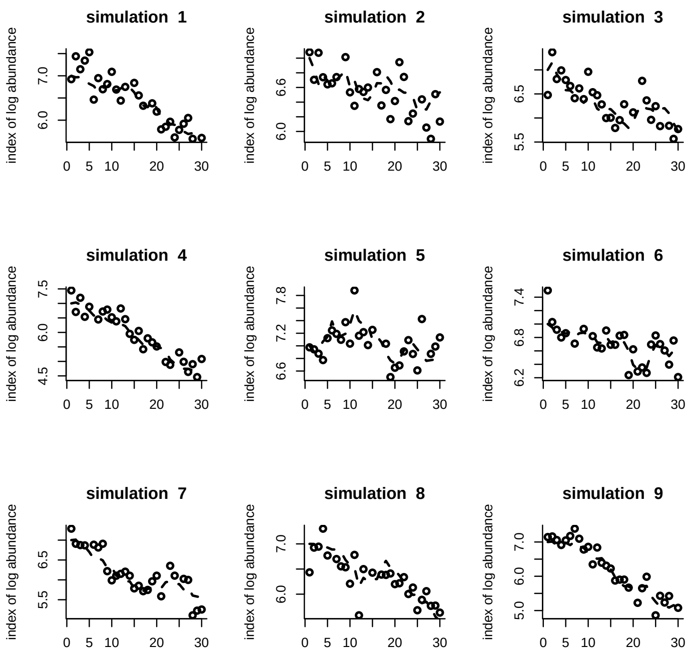

Fig. 8.1. Plot of nine simulated population time series with process and observation error. Circles are observation and the dashed line is the true population size.

Now let's look at the simulated data. Stochastic population trajectories show much variation, so it is best to look at a few at once. In Figure 8.1, nine simulations from the identical parameters (above) are shown.

Example 8.1 (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 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 8.1 code

Type `show.doc(MARSS, Case_study_1.R)` to open a file with all the example code.

```
par(mfrow=c(3,3))
sim.u = -0.05
sim.Q = 0.01
sim.R = 0.05
nYr= 30
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]=-99
  plot(years[y!=-99], y[y!=-99],
    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))
```

8.3 Maximum-likelihood parameter estimation

8.3.1 Model with process and observation error

We put the simulated data through the Kalman-EM algorithm in order to estimate the parameters, u, σ^2, and η^2, and 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 -99. After this call, the maximum-likelihood parameter estimates are `kem$par$U`, `kem$par$Q` and `kem$par$R`. There are numerous other outputs from the `MARSS()` function. To get a list of the outputs type in `names(kem)`. 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 8.1 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.params
```

```
                kem.U        kem.Q      kem.R
sim 1    -0.059757850 0.0009018543 0.05809171
sim 2    -0.024360242 0.0005989792 0.05125810
sim 3    -0.038994131 0.0185161657 0.05194617
sim 4    -0.087505238 0.0026163643 0.07185865
sim 5     0.002736377 0.0118516150 0.04981834
sim 6    -0.025048078 0.0056833688 0.03511823
sim 7    -0.067319467 0.0338319436 0.02899006
sim 8    -0.038528569 0.0015180227 0.06939992
sim 9    -0.074161836 0.0094017006 0.04898656
mean sim -0.045882115 0.0094355571 0.05171864
true     -0.050000000 0.0100000000 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.

The Kalman-EM algorithm also gives an estimate of the true population size with observation error removed. This is in `kem$states`. Figure 8.2 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!

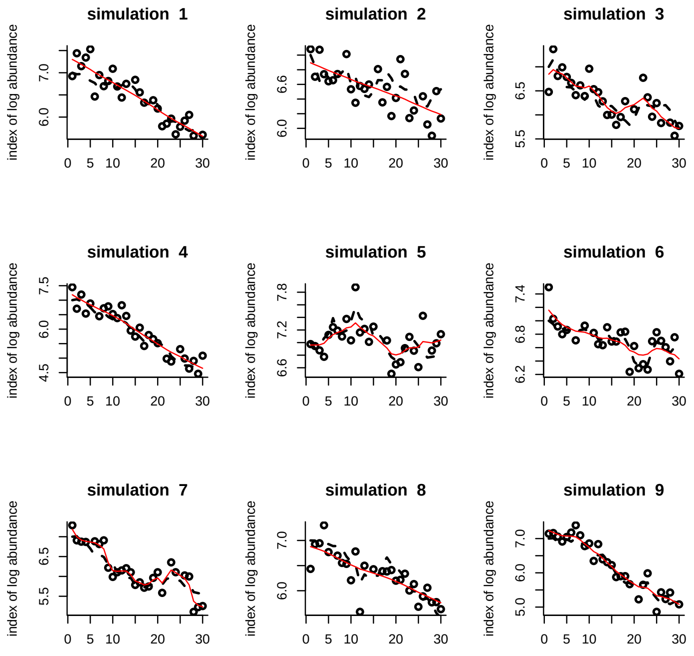

Fig. 8.2. The circles are the observed population sizes with error. The dashed lines are the true population sizes. The solid thin lines are the estimates of the true population size from the Kalman-EM algorithm

8.3.2 Model with no observation error

We used the Kalman-EM algorithm 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; 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[y!=-99] # the non missing years
den.y = y[y!=-99] # 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 8.3. 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))
```

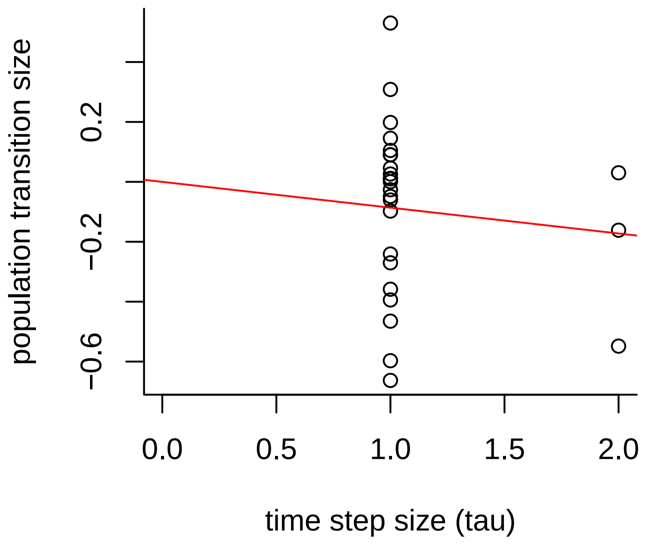

Fig. 8.3. The regression of $log(N_{t+\tau}) - log(N_t)$ against τ. The slope is the estimate of u and the variance of the residuals is the estimate of Q.

Here are the parameter values for the data in Figure 8.2 using the process-error only model:

```
den91.params
```

```
             den91.U    den91.Q
sim 1    -0.033086712 0.11983545
sim 2    -0.007719434 0.10845528
sim 3    -0.006090313 0.13184606
sim 4    -0.083201346 0.16542512
sim 5    -0.022558950 0.12813995
sim 6    -0.038119044 0.07062119
sim 7    -0.038037401 0.10246642
sim 8    -0.028013635 0.15826053
sim 9    -0.113624962 0.11851342
mean sim -0.041161311 0.12261816
true     -0.050000000 0.01000000
```

Notice that the u estimates are similar to those from the Kalman-EM algorithm, 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 $2 \times$ the observation variance).

Example 8.2 (The variability in parameter estimates)

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

Example 8.2 code

Type `show.doc(MARSS, Case_study_1.R)` to open a file with all the example code.

```
sim.u = -0.05   # growth rate
sim.Q = 0.01    # process error variance
sim.R = 0.05    # non-process error variance
nYr= 30         # 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=11, ncol=5,
  dimnames=list(c(paste("sim",1:9),"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]=-99

  #Kalman-EM estimates
  kem = MARSS(y.ts[i,], silent=TRUE)
  params[i,c(1,3,4)] = c(kem$par$U,kem$par$Q,kem$par$R)

  #Dennis et al 1991 estimates
  den.years = years[y.ts[i,]!=-99]  # the non missing years
  den.yts = y.ts[i,y.ts[i,]!=-99]   # 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     -0.0288 -0.0271 0.001127 0.0699  0.1398
sim 2     -0.0621 -0.0587 0.017859 0.0618  0.1476
sim 3     -0.0472 -0.0394 0.019210 0.0217  0.0644
sim 4     -0.0573 -0.0778 0.001121 0.0726  0.1688
sim 5     -0.0485 -0.0716 0.001163 0.0581  0.1151
sim 6     -0.0687 -0.0999 0.001715 0.0533  0.1260
sim 7     -0.0289 -0.0406 0.004226 0.0590  0.1334
sim 8     -0.0366 -0.0341 0.002307 0.0646  0.1160
sim 9     -0.0673 -0.0475 0.000503 0.0448  0.1107
mean sim -0.0495 -0.0552 0.005470 0.0562  0.1246
true      -0.0500 -0.0500 0.010000 0.0500  0.0100
```

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 cut and paste the code from the pdf file into an R script file or on to the R command line.*

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 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).*

8.4 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 of 'a p_d 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; 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 below the population size at the last census.

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:

$$\Pi(x_d, t_e) = \pi(u) \times \Phi\left(\frac{-x_d + |u|t_e}{\sqrt{\sigma^2 t_e}}\right) + \exp(2x_d|u|/\sigma^2)\Phi\left(\frac{-x_d - |u|t_e}{\sqrt{\sigma^2 t_e}}\right) \quad (8.3)$$

where x_e is the threshold and is defined as $x_e = \log(N_0/N_e)$, where 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 8.4 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 Kalman-EM 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 30-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 Kalman-EM 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 and are 100-yr better or worse. In Exercise 3, you'll remake this with different u. You'll discover from that forecasts are more certain for populations that are declining faster.

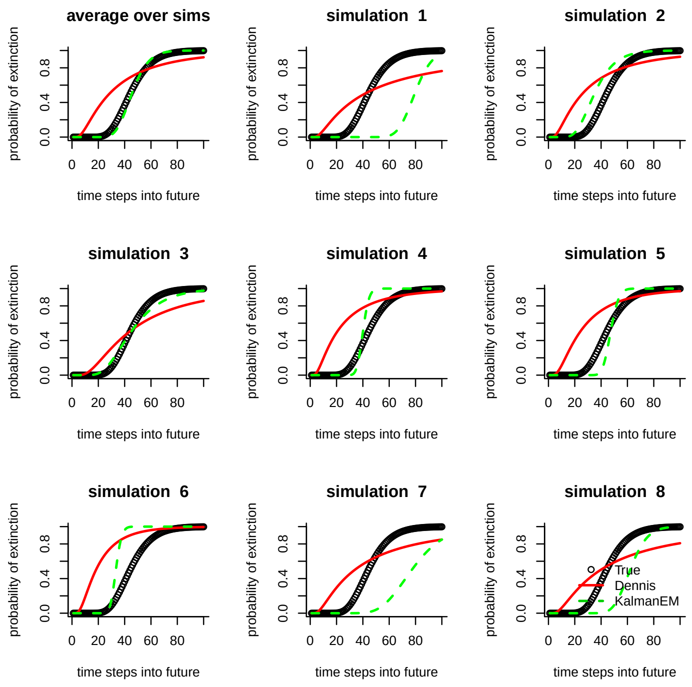

Fig. 8.4. Plot of the true and estimated probability of declining 90% in different time horizons for nine simulated population time series with observation error.

Example 8.3 (The effect of parameter values on risk estimates)

In this example, you will recreate Figure 8.4 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 8.2 code before running the Example 8.3 code.

Example 8.3 code

Type show.doc(MARSS, Case_study_1.R) 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)

  #Kalman-EM parameter estimates
  u=params[j,1];   Q=params[j,3]
  p.ever = ifelse(u<=0,1,exp(-2*u*xd/Q))
  for (i in 1:100){
    kal.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

  #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 8.2 code. Then run the Example 8.3 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 assumes that the data contain 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 8.2 code.)*

4. *How does changing the rate of decline (*`sim.u`*) change the estimates of risk? Rerun the Example 8.2 code using a lower u; this will create a new matrix of parameter estimates. Then run the Example 8.3 code. Do the estimates seem better of worse for rapidly declining populations?*

5. *Rerun the Example 8.2 code using fewer number of years (*`nYr` *smaller) and increase* `fracmissing`*. Then run the Example 8.3 code. The graphs will start to look peculiar. Why do you think it is doing that? Hint: look at the estimated parameters.*

8.5 Certain and uncertain regions

From Example 8.3, 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. Using only the point estimates of the probability of 90% decline by themselves in a PVA should not be done. 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 8.3, 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 8.5 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.

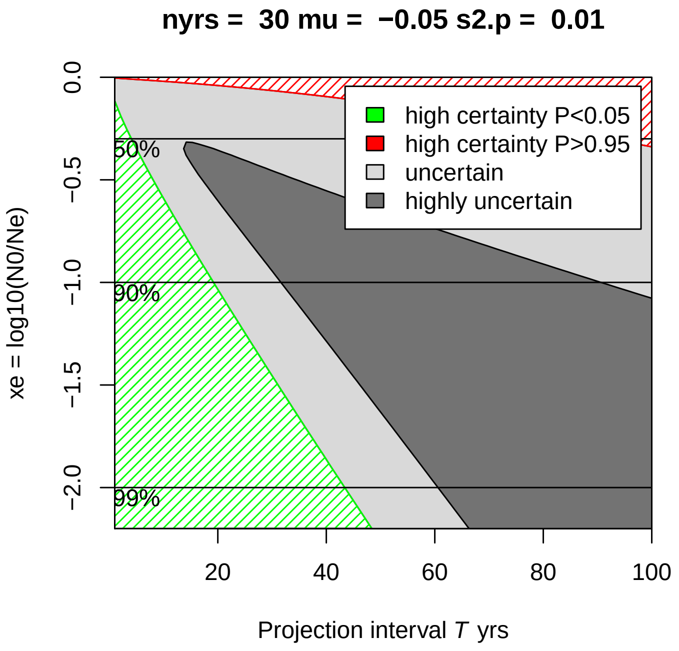

Fig. 8.5. This figure shows your region of high uncertainty (dark grey). In this region, the minimum 95% confidence intervals (meaning if you had no observation error) span 80% of the 0 to 1 probability. That is, you are uncertain if the probability of a specified decline is close to 0 or close to 1. The green (dots) 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 red (dots) 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 grey is between these two certain/uncertain extremes.

Example 8.4 (Uncertain and certain regions)

Use the Example 8.4 code to re-create Figure 8.5 and get a feel for when (what parameter ranges) risk estimates are more certain and when they are less certain.

Exercise 8.4 code

Type `show.doc(MARSS, Case_study_1.R)` to open a file with all the example code.

```
par(mfrow = c(1, 1))
CSEGtmufigure(N = 30, u = -0.05, s2p = 0.01)
```

`N` *are the number of years of data,* `u` *is the mean population growth rate, and* `s2p` *is the process variance.*

8.6 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 PVA 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 8.6 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 column 1 = years and column 2 = population count (logged). `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.

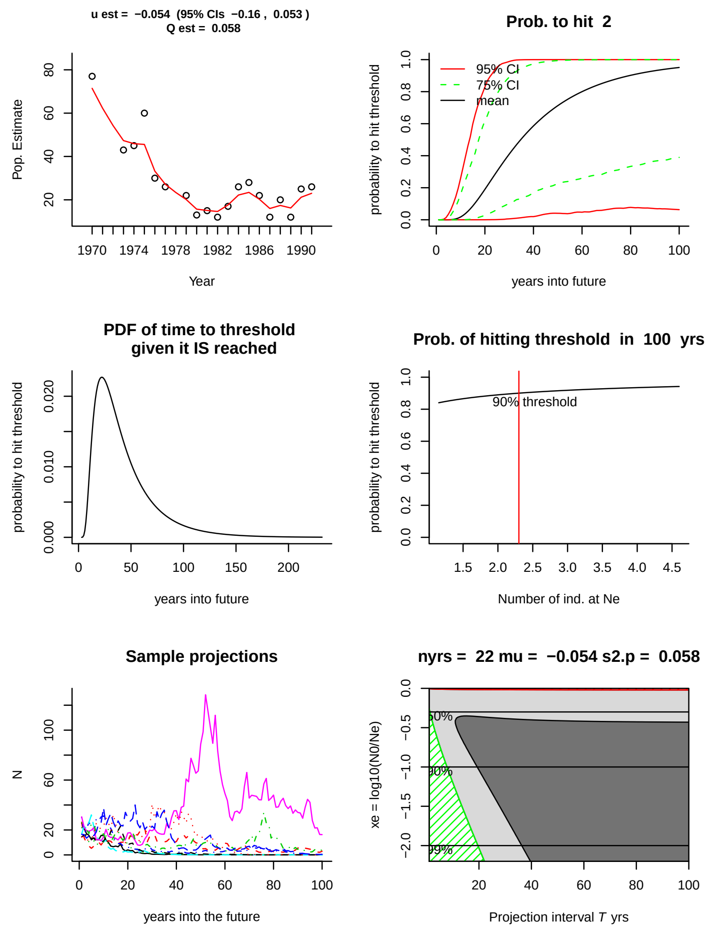

Fig. 8.6. Risk figure using data for the critically endangered African Wild Dog (data from Ginsberg et al. 1995). This population went extinct after 1992.

Example 8.5 (Risk figures for different species)

Use the Example 8.5 code to re-create Figure 8.6. 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 `show.doc(MARSS, Case_study_1.R)` *to open a file in* R*with all 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)
```

8.7 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 and σ^2 and η^2
- Then you simulate time series using those estimates and Equations 8.1 and 8.2
- 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 8.3 and code from Example 8.3)
- 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.

8.8 Comments

Data with cycles, from age-structure or predator-prey interactions, are difficult to analyze and the Kalman-EM algorithm used in the MARSS package will give poor estimates for this type of data. The slope method (Holmes, 2001) is 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 PVA 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).

9

Case study 2: Combining multi-site and subpopulation data to estimate trends and trajectories

9.1 The problem

In this case study, we will use multivariate state-space models to combine surveys from multiple 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 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 9.1 shows the data. The numbers on each line denote the different regions:

```
1 SJF
2 SJI
3 EBays
4 PSnd
5 HC
```

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

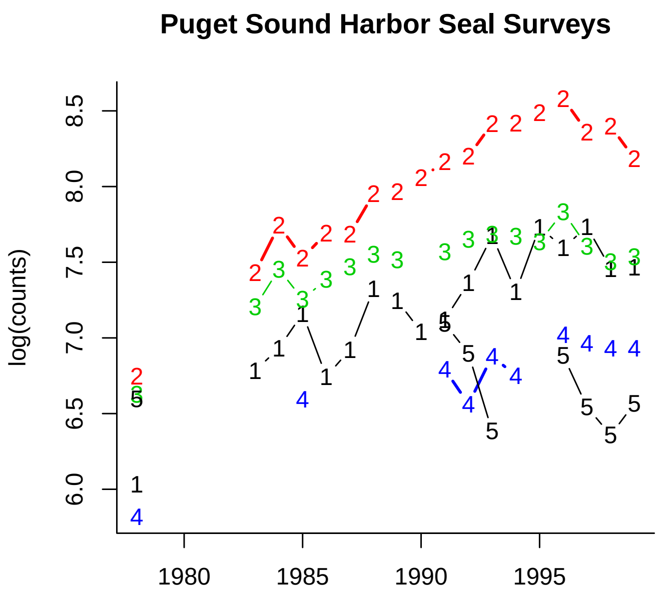

Fig. 9.1. Plot of the of the count data from the five harbor seal regions (Jeffries et al. 2003). Each region is an index of the total harbor seal population, but the bias (the difference between the index and the true population size) for each region is unknown.

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. You are going to 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 your assumptions change your estimates.

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 (`years`) are in row 1 of `dat` and the logged data are in the rest of the rows. 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)
```

```
     Years    SJF    SJI  EBays   PSnd    HC
[1,]  1978   6.03   6.75   6.63   5.82   6.6
[2,]  1979 -99.00 -99.00 -99.00 -99.00 -99.0
[3,]  1980 -99.00 -99.00 -99.00 -99.00 -99.0
[4,]  1981 -99.00 -99.00 -99.00 -99.00 -99.0
[5,]  1982 -99.00 -99.00 -99.00 -99.00 -99.0
[6,]  1983   6.78   7.43   7.21 -99.00 -99.0
[7,]  1984   6.93   7.74   7.45 -99.00 -99.0
[8,]  1985   7.16   7.53   7.26   6.60 -99.0
```

The `-99`'s in the data are missing values. The algorithm will ignore those values.

9.2 Analyze assuming a single total Puget Sound population

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

$$\mathbf{x}_t = \mathbf{B}\mathbf{x}_{t-1} + \mathbf{u} + \mathbf{v}_t, \text{ where } \mathbf{v}_t \sim \text{MVN}(0, \mathbf{Q}) \tag{9.1}$$
$$\mathbf{y}_t = \mathbf{Z}\mathbf{x}_t + \mathbf{a} + \mathbf{w}_t, \text{ where } \mathbf{w}_t \sim \text{MVN}(0, \mathbf{R}) \tag{9.2}$$

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

9.2.1 A single population process, x

When we are looking at trends over a large geographic region, we might make the assumption that the different census sites 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 population abundance as:

$$n_t = \exp(u + v_t) n_{t-1}, \tag{9.3}$$

where n_t is the total count in year t, u is the mean population growth rate, and v_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:

$$x_t = x_{t-1} + u + v_t. \tag{9.4}$$

$x_t = \log n_t$. 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 (σ^2). Thus $\mathbf{u}$ and $\mathbf{Q}$ are 1×1 matrices.

9.2.2 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, $\mathbf{y}_t = \mathbf{Z}\mathbf{x}_t + \mathbf{a} + \mathbf{w}_t$, is written as:

$$\begin{bmatrix} y_{1,t} \\ y_{2,t} \\ y_{3,t} \\ y_{4,t} \\ y_{5,t} \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} x_t + \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix} + \begin{bmatrix} w_{1,t} \\ w_{2,t} \\ w_{3,t} \\ w_{4,t} \\ w_{5,t} \end{bmatrix} \tag{9.5}$$

Each y_i is the time series for a different region (the names for the numbered regions are given on page 2). The a's are the bias between the regional sample and the total population. The a's are scaling (or intercept-like) parameters that are not important for trend estimation[2]. We will ignore them[3]. 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.[4]

[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

[3] 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 th total population.

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

We specify independent observation errors with unique variances by $\mathbf{w}_t \sim \mathrm{MVN}(0, \mathbf{R})$, where

$$\mathbf{R} = \begin{bmatrix} \eta_{1,t} & 0 & 0 & 0 & 0 \\ 0 & \eta_{2,t} & 0 & 0 & 0 \\ 0 & 0 & \eta_{3,t} & 0 & 0 \\ 0 & 0 & 0 & \eta_{4,t} & 0 \\ 0 & 0 & 0 & 0 & \eta_{5,t} \end{bmatrix} \tag{9.6}$$

$\mathbf{Z}$ is specifying 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 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 $n \times 1$:

$$\mathbf{Z} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \tag{9.7}$$

9.2.3 Set the constraints 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 `constraint` to `MARSS()`. `constraint` is a list which specifies any constraints on $\mathbf{Z}$, $\mathbf{u}$, $\mathbf{Q}$, etc. The function call will now look like:

```
kem = MARSS(dat, constraint=list(Z=Z.constraint, U=U.constraint,
      Q=Q.constraint, R=R.constraint) )
```

First we set the $\mathbf{Z}$ constraint. We need to tell the `MARSS` function that $\mathbf{Z}$ is a column vector of 1s (as in Equation 9.5). We do this using a $1 \times n$ vector *as an object of class factor.* The i-th element specifies which population trajectory the i-th observation time series belongs to. Since there is only one population trajectory in analysis 1, we will have a vector of five 1's: every observation time series is measuring the first, and only, population trajectory. In later analyses, you will see how to specify the constraint on Z when we have multiple populations.

```
Z.constraint = 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 **R** variance-covariance matrix only has terms on the diagonal (the variances) with the off-diagonal terms (the covariances) equal to zero[5]:

```
R.constraint = "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 constraints on **Z** and **R**. Since there is only one population, there is only one **u** and **Q** (they are scalars), so there are no constraints to set on them.

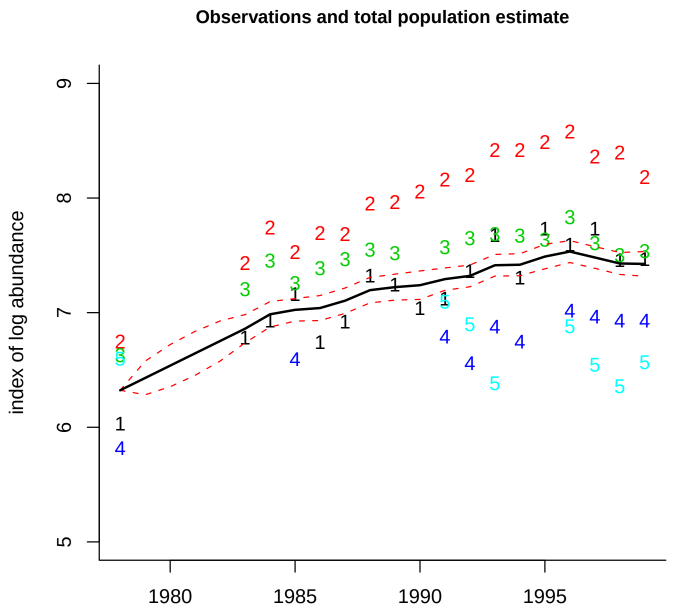

Fig. 9.2. Plot of the estimate of "log total harbor seals in Puget Sound" (minus the unknown bias for time series 1) against the data. The estimate of the total seal count has been scaled relative to the first time series. The 95% confidence intervals on the population estimates are the dashed lines. These are not the confidence intervals on the observations, and the observations (the numbers) will not fall between the confidence interval lines.

[5] For the EM function in the MARSS 1.0 package, the measurement errors must be uncorrelated if there are missing values in the data.

9.2.4 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 9.2) 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 9.2 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 a list: `kem1$par`. To get the element `U` of that list, which is the estimated long term population growth rate, type in `kem1$par$U`. Multiply by 100 to get the percent increase per year. The estimated process variance is given by `kem1$par$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 9.1 (Fit the single population model)

Analyze the harbor seal data using the single population model (Equations 9.4 and 9.5). The code for Example 9.1 shows you how to input data and send it to the function `MARSS()`*. When you run* `MARSS()`*, it will print information on the structure of the model it is fitting and how many iterations it took to run. 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 9.1 code

Type `show.doc(MARSS, Case_study_2.R)` 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.constraint = factor(c(1,1,1,1,1))
R.constraint = "diagonal and unequal"
kem1 = MARSS(dat, constraint=
  list(Z=Z.constraint, R=R.constraint))
#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$states,type="l",lwd=2)
title("Observations and total population estimate",cex.main=.9)
#show params
kem1$par
kem1$logLik
kem1$AIC
```

9.3 Changing the assumption about the observation variances

The variable `kem1$par$R` contains the estimates of the observation error variances. It is a matrix. Here is **R** from Example 9.1:

```
print(kem1$par$R, digits=3)
```

```
        SJF:1  SJI:2 EBays:3 PSnd:4  HC:5
SJF:1   0.0325 0.0000  0.0000 0.0000 0.000
SJI:2   0.0000 0.0355  0.0000 0.0000 0.000
EBays:3 0.0000 0.0000  0.0131 0.0000 0.000
PSnd:4  0.0000 0.0000  0.0000 0.0113 0.000
HC:5    0.0000 0.0000  0.0000 0.0000 0.195
```

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 same survey methodology[6].

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

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

This tells `MARSS` that all the η^2's along the diagonal in **R** are the same. To fit this model to the data, call `MARSS()` as:

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

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] -10.231173   8.384659
```

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. Go ahead and type in the *R*code. Then add the parameter estimates to the table at the back.

One of the key diagnostics when you are comparing fits from multiple models, it to examine whether the model is flexible enough to fit the data. You do this by looking for temporal trends in the the residuals between the estimated population states (e.g. `kem2$states`) and the data. In Figure 9.3, 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[7].

Example 9.2 (Fit a model with shared observation variances)

[6] 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.

[7] 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.

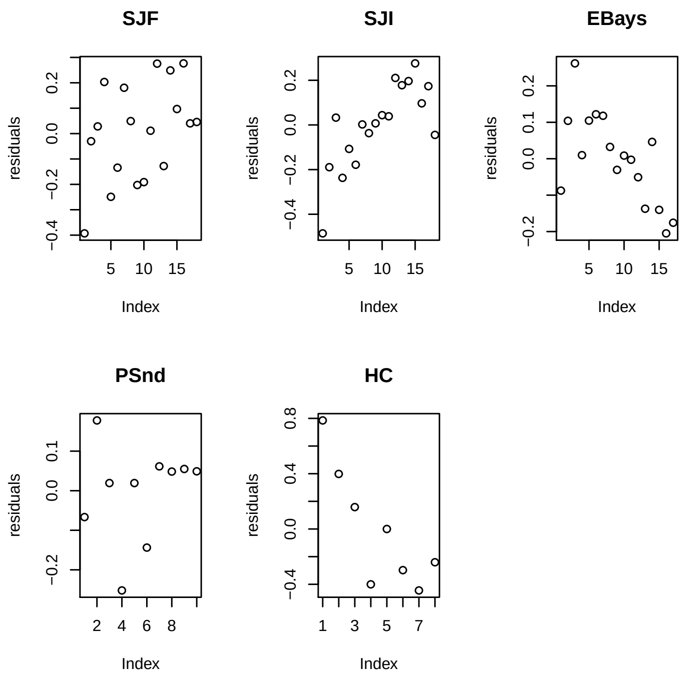

Fig. 9.3. Residuals for the model with a single population. The plots of the residuals should not have trends with time, but they do... This is an indication that the single population model is inconsistent with the data. The code to make this plot is given in the script file for this case study.

Analyze the data using the same population model as in Example 9.1, but constrain the **R** *matrix so that all sites have the same observation variance. The Example 9.2 code shows you how to do this. It also shows you how to make the diagnostics figure (Figure 9.3).*

Example 9.2 code

Type `show.doc(MARSS, Case_study_2.R)` to open a file with all the example code.

```
#fit model
Z.constraint = factor(c(1,1,1,1,1))
R.constraint = "diagonal and equal"
kem2 = MARSS(dat, constraint=
  list(Z=Z.constraint, R=R.constraint))
#show parameters
kem2$par$U       #population growth rate
kem2$par$Q       #process variance
kem2$par$R[1,1]  #observation variance
kem2$logLik #log likelihood
c(kem1$AIC,kem2$AIC)
#plot residuals
plotdat = t(dat); plotdat[plotdat == -99] = NA;
matrix.of.biases = matrix(kem2$par$A,
  nrow=nrow(plotdat),ncol=ncol(plotdat),byrow=T)
xs = matrix(kem2$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))
```

9.4 Analyze the data assuming North and South subpopulations

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, σ^2, 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:

$$\begin{bmatrix} x_{n,t} \\ x_{s,t} \end{bmatrix} = \begin{bmatrix} x_{n,t-1} \\ x_{s,t-1} \end{bmatrix} + \begin{bmatrix} u \\ u \end{bmatrix} + \begin{bmatrix} v_{n,t} \\ v_{s,t} \end{bmatrix} \tag{9.8}$$

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:

$$\begin{bmatrix} v_{n,t} \\ v_{s,t} \end{bmatrix} \sim MVN\left(\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \Sigma = \begin{bmatrix} \sigma^2 & 0 \\ 0 & \sigma^2 \end{bmatrix}\right) \tag{9.9}$$

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

$$\begin{bmatrix} y_{1,t} \\ y_{2,t} \\ y_{3,t} \\ y_{4,t} \\ y_{5,t} \end{bmatrix} = \begin{bmatrix} A_1 \\ A_2 \\ A_3 \\ A_4 \\ A_5 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_{n,t} \\ x_{s,t} \end{bmatrix} + \begin{bmatrix} w_{1,t} \\ w_{2,t} \\ w_{3,t} \\ w_{4,t} \\ w_{5,t} \end{bmatrix} \tag{9.10}$$

9.4.1 Specifying the `MARSS()` arguments

We need to change the **Z** constraint 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:

```
Z.constraint = factor(c(1,1,2,2,2))
```

We want to specify that the u's are the same for each subpopulation and that **Q** is diagonal with equal σ^2's. To do this, we set

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

This says that there is one u and one σ^2 parameter and both subpopulations share it (if we wanted the u's to be different, we would use `U.constraint="unequal"` or leave off the **u** constraint since the default behavior is `U.constraint="unequal"`).

Now we fit this model to the data and pass in the new constraints. The output tells us the structure of the model that was fit to the data and how long it took to fit the model:

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

```
abstol reached in 25 iterations and parameters appear converged.
Alert: with less than 100 iterations, the convergence diagnostics will be uncertain.

MARSS fit is
Estimation method:  kem
Estimation converged in 25 iterations.
Log-likelihood: 12.22753
AIC: -8.455068   AICc: -6.132488

     Estimate
A.2   0.79875
A.4  -0.77149
A.5  -0.83642
Q.1   0.00878
R.1   0.02950
U.1   0.05025
x0.1  6.10811
x0.2  6.90050

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

We estimated two initial x's, one process variance, one u, three a's, and one observation variance. So $K = 8$ parameters. The Kalman filter requires an initial condition ($t = 1$) for each x time series. When $m < n$, the number of a's estimated is $n - m$ since one of the a's for each state process will be set to 0. The AIC is `2*8 - 2*kem3$logLik`.

Figure 9.4 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 9.3 (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,* u *and* σ^2*. The Example 9.3 code shows how to set the `MARSS()` arguments for this case.*

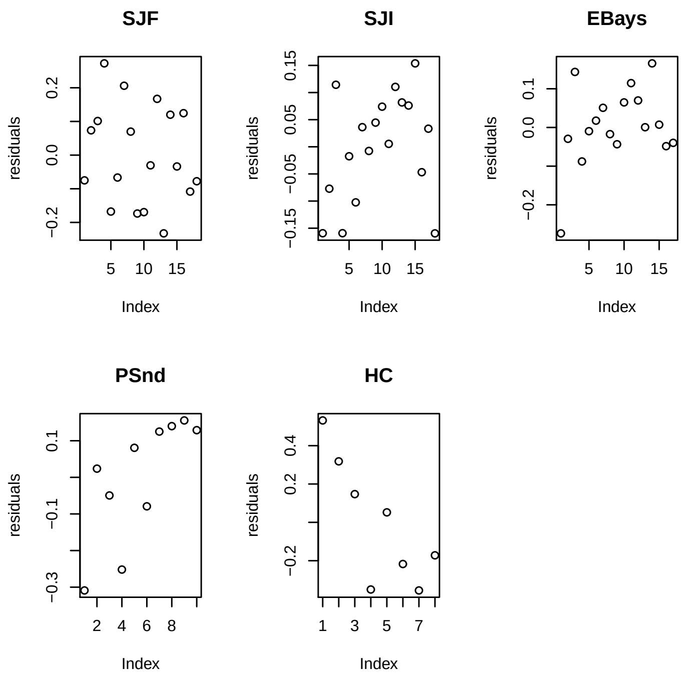

Fig. 9.4. The residuals for the analysis with a North and South subpopulation. The plots of the residuals should not have trends with time. Compare with the residuals for the analysis with one subpopulation.

Example 9.3 code

Type `show.doc(MARSS, Case_study_2.R)` to open a file with all the example code.

```
#fit model
Z.constraint = factor(c(1,1,2,2,2))
U.constraint = "equal"
Q.constraint = "diagonal and equal"
R.constraint = "diagonal and equal"
kem3 = MARSS(dat, constraint=list(Z=Z.constraint,
  R=R.constraint, U=U.constraint, Q=Q.constraint))
#plot residuals
plotdat = t(dat); plotdat[plotdat == -99] = NA;
matrix.of.biases = matrix(kem3$par$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))
```

9.5 Using MARSS() to fit 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 9.4 (Five subpopulations)

Analyze the data using a model with five subpopulations, where each site 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* σ^2*. The Example 9.4 code shows how to set the* `MARSS()` *arguments for this case. You can change* `R.constraint="diagonal and equal"` *to make all the observation variances equal.*

Example 9.4 code

Type `show.doc(MARSS, Case_study_2.R)` *to open a file with all the example code.*

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

Example 9.5 (Two subpopulations but different divisions)

Analyze the data using a model that assumes that the Strait of Juan de Fuca and San Juan Islands sites represent a northern Puget Sound subpopulation, while the other three sites represent a southern Puget Sound subpopulation. This time assume that each population trajectory (north and south) has different population parameters, u *and* σ^2 *and that each of the five sampling sites 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* `show.doc(MARSS, Case_study_2.R)` *at the R command line.*

Example 9.6 (Hood Canal treated separately but covaries with others)

Analyze the data using a model with two subpopulations with the divisions being Hood Canal versus everywhere else. Set

```
Q.constraint = "equalvarcov"
```

to make all the subpopulations covary in time but with equal covariances and variances.

Example 9.7 (Three subpopulations with shared parameter values)

Analyze the data using a model with three subpopulations as follows: north (sites 1 and 2), south (sites 3 and 4), Hood Canal (site 5). You can specify that some subpopulations share parameters while others do not. You do this by using a vector of factors for the constraints:

```
Q.constraint=factor(c("coastal","interior","interior"))
U.constraint=factor(c("puget sound","puget sound","hood canal"))
R.constraint=factor(c("boat","boat","plane","plane","plane"))
```

When `Q.constraint` *and* `U.constraint` *are vectors (passed in as a factor), as above, they specify which x's share parameter values. The factors must be a vector of length m, where m is the number of x's. The i-th factor corresponds to the i-th x. In the example above, we specified that x_1 has its own process variance (which we named "coastal") and x_2 and x_3 share a process variance value (which we named "interior"). For the long-term trends, we specified 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").*

When `R.constraint` *is vector of factors, it specifies which y's have the same observation variance. We need a 1×5 vector here because we need to specify a value for each observation time series (there are 5). Here we imagine that observation time series 1 and 2 are boat surveys while the others are plane surveys and we want to allow the variances to differ based on methodology.*

9.6 Discussion

Case Study 2 shows you how to combine multiple datasets that are measuring the same underlying process and fit those data using a multivariate state-space framework. This allows you to combine data sets and use all the available data. You can also combine data that are discontinuous; that is data that do not overlap in time. For example, if you have data from one type of monitoring program in one set of years and then data from a different program starting in some later years, you can still easily estimate the population dynamics parameters using both sets of data.

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 is 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. The current EM code will not handle covariance in $\mathbf{R}$ when there are missing data, but even it did, 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 change your data in a way that will make the observation errors independent—for example, using data from different months or defining your "regions"
- The `MARSS()` argument `control` specifies the options for the EM algorithm. We left the default tolerance, `abstol=0.01`. You would want to set this lower, e.g. `abstol=0.0001`, 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 ($\mathbf{u}$, $\mathbf{Q}$, $\mathbf{R}$).

Results table

Ex.		pop. growth rate `kem$par$U`	process variance `kem$par$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 + σ^2's identical					
5	N+S subpops unique obs vars u's + σ^2's identical					
6	PS + HC subpops unique obs vars u's + σ^2's unique					
7	N + S + HC subpops unique obs vars u's + σ^2's unique					

For AIC, 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 measurement error variances are all identical versus different affect your estimate of the trend? You may want to rerun examples 3-7 with the `R.constraint` changed. `R.constraint="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 trend? 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 estimate process variance was very small, `kem1$par$Q`. Why do you think σ^2 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.]

10

Case Study 3: Using MARSS models to identify spatial population structure and covariance

10.1 The problem

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 10.1). 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 10.1). 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? Type `show.doc(MARSS, Case_study_3.R)` to open a file in *R*with all *R*code to get you started on the analyses in this chapter.

10.2 Question 1: 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.

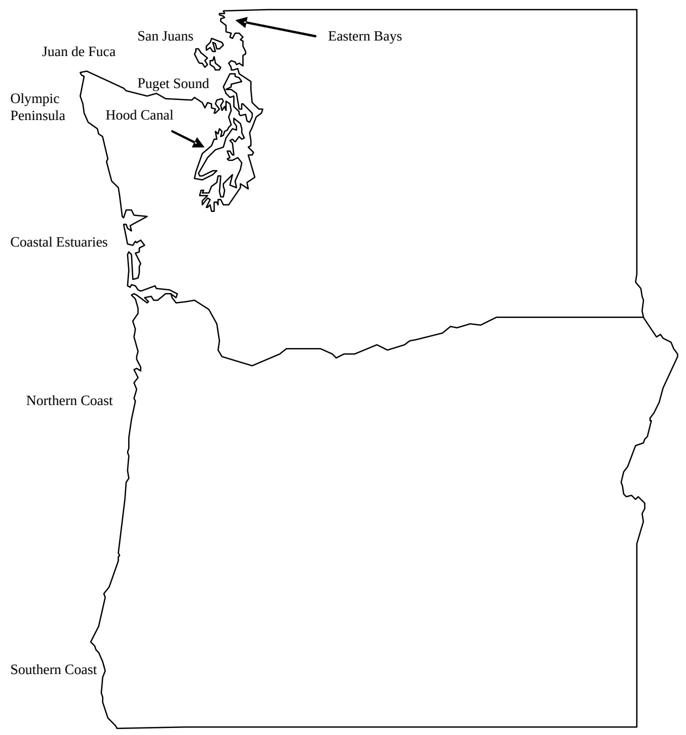

Fig. 10.1. Map of spatial distribution of 9 harbor seal sites in Washington and Oregon.

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.

10.2.1 Specify the design, Z, matrices

Write down the **Z** matrices for the hypotheses. Hint: Hypothesis 1 and 2 have the same **Z** matrix, Hypothesis 3 and 4 have the same **Z** matrix and Hypothesis 5 is a column of 1s.

$$\begin{array}{r|cc|cc|c|}
 & \multicolumn{2}{c}{\text{H 1 and 2}} & \multicolumn{2}{c}{\text{H 3 and 4}} & \text{H 5}\\
 & \multicolumn{2}{c}{\mathbf{Z}} & \multicolumn{2}{c}{\mathbf{Z}} & \mathbf{Z}\\
 & \text{subpop} & \text{subpop} & \text{subpop} & \text{subpop} & \text{subpop}\\
 & 1 & 2 & 1 & 2 & 1\\
\texttt{Coastal Estuaries} & & & & & \\
\texttt{Olympic Peninsula} & & & & & \\
\texttt{Str. Juan de Fuca} & & & & & \\
\texttt{San Juan Islands} & & & & & \\
\texttt{Eastern Bays} & & & & & \\
\texttt{Puget Sound} & & & & & \\
\texttt{Hood Canal} & & & & & \\
\texttt{OR North Coast} & & & & & \\
\texttt{OR South Coast} & & & & &
\end{array}$$

Next you need to specify the constraints argument so that `MARSS` knows the structure of your $\mathbf{Z}$'s. The $\mathbf{Z}$ constraint will be a vector of factors, i.e. it will have the form `factor(c(....))`.

- Hypothesis 1 and 2: `Z.constraint=`
- Hypothesis 3 and 4: `Z.constraint=`
- Hypothesis 5: `Z.constraint=`

10.2.2 Specify the grouping arguments

For this case study, we will assume that subpopulations share the same growth rate. What should `U.constraint` be for each hypothesis? To specify shared u parameters, `U.constraint` is set as a length m vector of factors and specifies which subpopulations share their u parameter. Written in R it takes the form `factor(c(#,#,...))`

- Hypothesis 1-4: `U.constraint=`
- Hypothesis 5: `U.constraint=`

What about `Q.constraint`? To specify a diagonal $\mathbf{Q}$ matrix with shared values along the diagonal, `Q.constraint` is set as a length m vector of factors. The vector specifies which x_i's share their process variance parameter. Look at each hypothesis (above) and write down the corresponding `Q.constraint`.

- Hypothesis 1: `Q.constraint=`
- Hypothesis 2: `Q.constraint=`
- Hypothesis 3: `Q.constraint=`
- Hypothesis 4: `Q.constraint=`
- Hypothesis 5: `Q.constraint=`

Lastly, specify `R.constraint`. As we mentioned above, we will assume that the observation errors are independent and the observation variance is the same across sites. You can specify this constraint either as a text string or as a n length vector of factors.

- Hypothesis 1-5: `R.constraint=`

10.2.3 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, constraint=list(Z = Z.constraint,
  Q = Q.constraint, R = R.constraint, U = U.constraint))
```

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.

H	pop. growth rate `kem$par$U`	process variance `kem$par$Q`	obs. variance `kem$par$R`	K `kem$num.params`	log-like. `kem$logLik`	AIC `kem$AIC`
1						
2						
3						
4						
5						

10.2.4 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.constraint`). You can also try the analysis with temporally co-varying subpopulations (good and bad years correlated) by setting `Q.constraint="unconstrained"` or `Q.constraint="equalvarcov"`.

10.3 Question 2: Is Hood Canal separate?

The Hood Canal site may represent a distinct population, and has recently been subjected to 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

10.3.1 Specify the Z matrix and `Z.constraint`

The **Z** matrix for each hypothesis is the same. The coastal subpopulation consists of 4 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 4 sites. Thus $m = 3$ and **Z** is a 9×3 matrix:

$$\begin{array}{r c}
 & \begin{array}{ccc} \text{subpop 1} & \text{subpop 2} & \text{subpop 3} \end{array} \\
\begin{array}{r}
\texttt{Coastal Estuaries} \\
\texttt{Olympic Peninsula} \\
\texttt{Str. Juan de Fuca} \\
\texttt{San Juan Islands} \\
\texttt{Eastern Bays} \\
\texttt{Puget Sound} \\
\texttt{Hood Canal} \\
\texttt{OR North Coast} \\
\texttt{OR South Coast}
\end{array} &
\left[\begin{array}{ccc}
 & & \\
 & & \\
 & & \\
 & & \\
 & & \\
 & & \\
 & & \\
 & & \\
 & &
\end{array}\right]
\end{array}$$

Then write down `Z.constraint` for this **Z**: `factor(c(...))`

10.3.2 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.constraint=`
- Hypothesis 2: `U.constraint=`
- Hypothesis 3: `U.constraint=`
- Hypothesis 4: `U.constraint=`

`U.constraint` will be a length m vector of factors. Once you have more than two subpopulations, it can get hard to keep straight which `U.constraint` 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.constraint` correspond one-to-one with the columns of **Z**:

`U.constraint=factor(c(col 1 Z, col 2 Z, col 3 Z, ..))`.

Specify `Q.groups` showing which subpopulations share their process variance parameter.

- Hypothesis 1: `Q.constraint=`
- Hypothesis 2: `Q.constraint=`
- Hypothesis 3: `Q.constraint=`
- Hypothesis 4: `Q.constraint=`

`Q.constraint` will be a length m vector of factors. `R.constraint` is the same as for Question 1; the observation variances are the same for each site.

10.3.3 Fit the 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, constraint=list(Z = Z.constraint,
   Q = Q.constraint, R = R.constraint, U = U.constraint))
```

H	pop. growth rate `kem$par$U`	proc. variance `kem$par$Q`	obs. variance `kem$par$R`	K `kem$num.params`	log-like `kem$logLik`	AIC `kem$AIC`
1						
2						
3						
4						

10.3.4 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.constraint[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 `show.doc(MARSS, Case_study_3.R)` to open a file in R with all the example code.

11

Case Study 5: Using state-space models to analyze noisy animal tracking data

11.1 A simple random walk model of animal movement

A simple random walk model of movement with drift but no correlation is

$$x_{1,t} = x_{1,t-1} + u_1 + v_{1,t}, \;\; v_{1,t} \sim \mathrm{N}(0, \sigma_1^2) \tag{11.1}$$
$$x_{2,t} = x_{2,t-1} + u_2 + v_{2,t}, \;\; v_{2,t} \sim \mathrm{N}(0, \sigma_2^2) \tag{11.2}$$

where $x_{1,t}$ is the location at time t along one axis (in our case study, longitude) and $x_{2,t}$ is for another, generally orthogonal, axis (in our case study, latitude). We add errors to our observations of location:

$$y_{1,t} = x_{1,t} + a_1 + w_{1,t}, \;\; w_{1,t} \sim \mathrm{N}(0, \eta_1^2) \tag{11.3}$$
$$y_{2,t} = x_{2,t} + a_2 + w_{2,t}, \;\; w_{2,t} \sim \mathrm{N}(0, \eta_2^2), \tag{11.4}$$

Together Equations 11.2 and 11.4 describe a MARSS model (now written in matrix form):

$$\mathbf{x}_t = \mathbf{x}_{t-1} + \mathbf{u} + \mathbf{v}_t, \;\; \mathbf{v}_t \sim \mathrm{MVN}(0, \mathbf{Q}) \tag{11.5}$$
$$\mathbf{y}_t = \mathbf{x}_t + \mathbf{a} + \mathbf{w}_t, \;\; \mathbf{w}_t \sim \mathrm{MVN}(0, \mathbf{R}). \tag{11.6}$$

11.2 The problem

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 11.1) and it would appear that our sea turtles are becoming land turtles (at least part of the time).

For this case study, we will use the `MARSS()` function to estimate true positions and speeds from the corrupted data. We will use a mapping package to plot the results: the `maps` package. If you have not already, install this package by selecting the 'Packages' menu and then 'Install packages' and then select `maps`. If you are on a Mac, remember to select "binaries" for the package type. Type `show.doc(MARSS, Case_study_5.R)` to open a file in *R*with all *R*code to get you started on the analyses in this chapter.

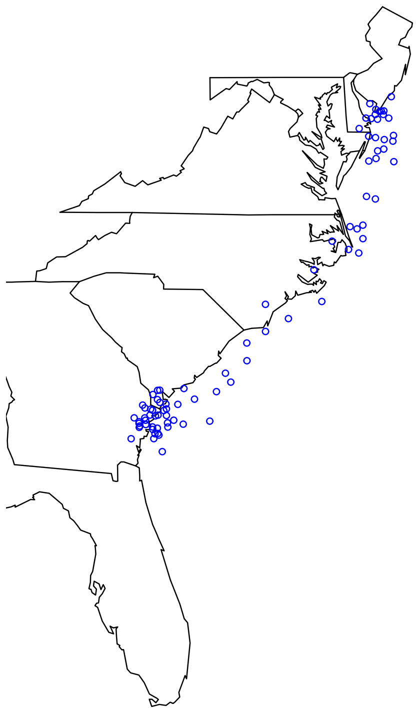

Fig. 11.1. Plot of the tag data from the turtle Big Mama. Errors in the location data make it seem that Big Mama has been moving overland.

11.3 Estimate locations from bad tag data

11.3.1 Read in the data and load `maps` package

Our noisy data are in `loggerheadNoisy`. They consist of daily readings of location (longitude/latitude). The data are recorded daily and `MARSS()` requires a data entry for each day. If data are missing for a day, then the entries for lat and lon for that day should be -99. However, to make this case study run quickly, we have interpolated all missing values in the original, uncorrupted, dataset (`loggerhead`). 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:

```
levels(loggerheadNoisy$turtle)
```

```
[1] "BigMama"  "Bruiser"  "Humpty"   "Isabelle" "Johanna"
[6] "MaryLee"  "TBA"      "Yoto"
```

We will first analyze the position data for "Big Mama". We put the data for "Big Mama" into variable `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 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). There is one observation time series for each so

```
Z.constraint=factor(c(1,2))
```

. We will assume that the errors are independent and that there are different drift rates (u), process variances (σ^2) and measurement variances for latitude and longitude (η^2).

```
U.constraint="unequal"
Q.constraint="diagonal and unequal"
R.constraint="diagonal and unequal"
```

Fit the model to the data:

```
kem = MARSS(dat, constraint=list(Z = Z.constraint,
      Q = Q.constraint, R = R.constraint, U = U.constraint))
```

11.3.2 Compare state estimates to the real positions

The real locations (from which `loggerheadNoisy` was produced by adding noise) are in `loggerhead`. In Figure 11.2, we compare the tracks estimated from the noisy data with the original, good, data (see the *R* script, `Case_Study_5.R` for the code to make this plot. There are only a few data points for the real data because in the real tag data, there are many missing days.

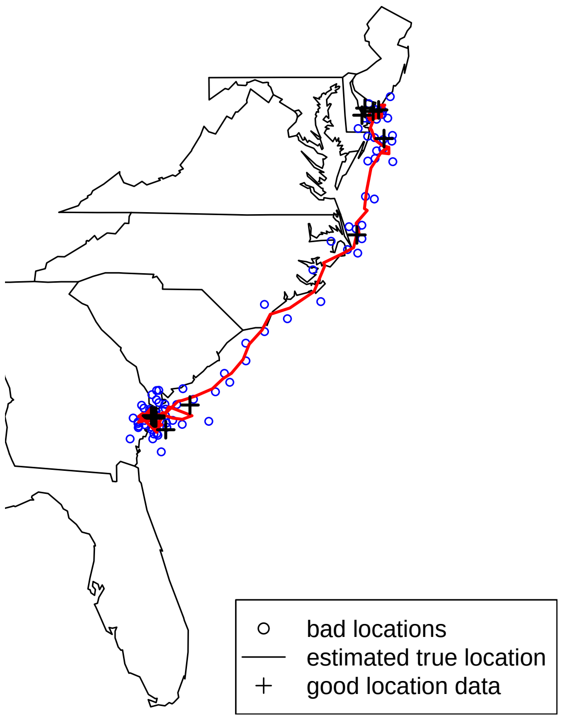

Fig. 11.2. Plot of the estimated track of the turtle Big Mama versus the good location data (before we corrupted it with noise).

11.3.3 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 at the beginning of the *R* script, `Case_Study_5.R`):

```
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(-99, 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 11.3). Repeat the analysis done for "Big Mama" for each of the other turtles and fill

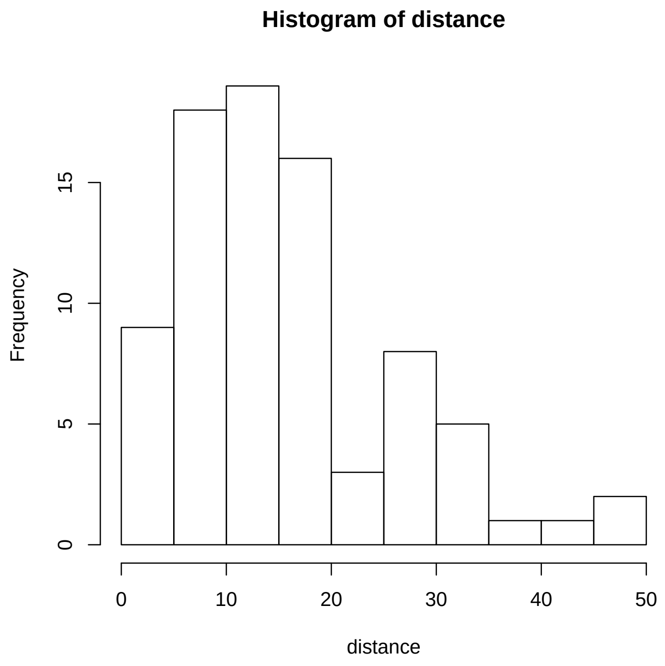

Fig. 11.3. Histogram of the miles traveled per day for Big Mama. Compare this to the estimate of miles traveled per day if you had not accounted for measurement errors. See the script file, `Case_Study_5.R`, for the code to this.

out the speed table (Table 11.3.3). If you were given the opportunity to race these turtles, would you bet on Big Mama being the fastest?

Turtle	Estimated Speed
Big Mama	
Bruiser	
Humpty	
Isabelle	
Johanna	
Mary Lee	
TBA	
Yoto	

11.4 Comparing turtle tracks to proposed fishing areas

One of the greatest threats to the long term viability of loggerhead turtles is incidental take by net/pot fisheries. Add two proposed fishing areas to your turtle plots:

```
# the proposed fishery areas
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)
```

Given that only one area can be chosen as a future fishery, what do your predicted movement trajectories for our eight turtles tell you?

11.5 Using `fields` to get density plots of locations

If you are comfortable programming in *R*, load the `fields` package. Make 3D density plots of predicted sea turtle locations. Which two areas appear to be most visited?

Include the confidence interval estimates for each location in this analysis. For this part of the exercise, we will assume that the confidence intervals are roughly the same as the probability intervals (Bayesian). We can assume that the error in latitude is independent from error in longitude. The `fields` package includes a couple different functions. One that might be useful here is `Tps()`, like in the example (`?fields`). To call `fields`, we need predictor variables (**x**), which can be random lon/lat pairs randomly drawn within the range of the data. The other requirement for `Tps()` is the response, **y**. If we think of each predicted state being a bivariate normal density, the response for each of our random pairs can be the density across all of the predicted states. There is code to help you get started in the *R* file, `Case_Study_5.R`.

11.6 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 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
# 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")
```

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, New York, New York, USA.

Harvey, A. C. 1989. Forecasting, structural time series models and the Kalman filter. Cambridge University Press, Cambridge, UK.

Durbin, J., and S. J. Koopman. 2001. Time series analysis by state space methods. Oxford University Press, Oxford.

King, R., G. Olivier, B. Morgan, and S. Brooks. 2009. Bayesian analysis for population ecology.

Giovanni, P., S. Petrone, and P. Campagnoli. 2009. Dynamic linear models in R.

Pole, A., M. West, and J. Harrison. 1994. Applied Bayesian forecasting and time series analysis.

Bolker, B. 2008. Ecological models and data in R. Princeton University Press.

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.

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.

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: Object structures

B.1 Model objects: class marssm

Objects of class 'marssm' specify Multivariate Autoregressive State Space (MARSS) models. The `model` component of an ML estimation object (class marssMLE; see below) is a marssm object. These objects have the following components:

data An optional matrix (not dataframe), in which each row is a time series (time across columns).
fixed A list with 8 matrices Z, A, R, B, U, Q, x0, V0, specifying which elements of each parameter are fixed.
free A list with 8 matrices Z, A, R, B, U, Q, x0, V0, specifying which elements of each parameter are to be estimated.
M An array of dim $n \times n \times T$ (an $n \times n$ missing values matrix for each time point). Each matrix is diagonal with 0 at the i, i value if the i-th value of $\mathbf{y}$ is missing, and 1 otherwise.
miss.value Specifies missing value representation in the data.

The matrices in `fixed` and `free` work as pairs to specify the fixed and free elements for each parameter. See Chapter 4. The dimensions for `fixed` and `free` matrices are as follows, where n is the number of observation time series and m is the number of state processes:

Z	n x m
B	m x m
U	m x 1
Q	m x m
A	n x 1
R	n x n
x0	m x 1
V0	m x m

Use `is.marssm()` to check whether an marssm object is correctly specified. The MARSS package includes an `as.marssm()` method to convert objects of class popWrap (see next section) to objects of class marssm.

B.2 Wrapper objects: class popWrap

Wrapper objects of class popWrap contain specifications and options for estimation of a MARSS model. A popWrap object has the following components:

data A matrix (not dataframe) of observations (rows) $\times$ time (columns).
m Number of hidden state processes (number of rows in $\mathbf{x}$).
constraint Either a list with 8 string elements Z, A, R, B, U, Q, x0, V0 (see below for details), or string `"use fixed/free"`.
fixed If `constraint[[elem]]="use fixed/free"`, a list with 8 matrices Z, A, R, B, U, Q, x0, V0.
free If `constraint[[elem]]="use fixed/free"`, a list with 8 matrices Z, A, R, B, U, Q, x0, V0.
inits A list with 8 matrices Z, A, R, B, U, Q, x0, V0, specifying initial values for parameters. Dimensions are given in the class 'marssm' section.
miss.value Specifies missing value representation (default is -99).
method The method used for estimation: 'kem' for Kalman-EM, 'BFGS' for quasi-Newton.
control List of estimation options. For the EM algorithm, these include the elements `minit, maxit, abstol, iter.V0, safe` and `trace`. For Monte Carlo initialization, these include the elements `MCInit`, `numInits`, `numInitSteps` and `boundsInits`. See class marssMLE section for details.

Component `constraint` is a convenient way to specify model structure for certain common cases. If `constraint="use fixed/free"`, both `fixed` and `free` must be provided. See the class marssm section for how to specify fixed and free matrices. The function `MARSS()` calls `popWrap()` to create a popWrap object, then `is.marssm()` to coerce this object to class marssm for the estimation function.

The `popWrap()` function calls `checkPopWrap()` to check user inputs from `MARSS()`. Valid constraints are below.

A May be either the string 'scaling' or the string 'zero' to specify a column vector of zeros ($\mathbf{a} = 0$).
B String 'identity' or a numeric matrix specifying a fixed $\mathbf{B}$ matrix. The string 'zero' may be used to specify a $m \times m$ matrix of zeros ($\mathbf{B} = 0$).
Q String 'unconstrained', 'diagonal and unequal', 'diagonal and equal', or 'equalvarcov'. May also be numeric or character vector of class factor specifying shared diagonal values or a numeric matrix specifying a fixed $\mathbf{Q}$ matrix.

R String ‘unconstrained’, ‘diagonal and unequal’, ‘diagonal and equal’, or ‘equalvarcov’. May also be numeric or character vector of class factor specifying shared diagonal values or a numeric matrix specifying a fixed **R** matrix.

U String ‘unconstrained’=‘unequal’, or ‘equal’. May also be numeric or character vector of class factor specifying shared **u** elements or a $m \times 1$ numeric matrix specifying a fixed **u** matrix. The string ‘zero’ may be used to specify a column vector of zeros ($\mathbf{u} = 0$).

x0 String ‘unconstrained’=‘unequal’, or ‘equal’. May also be vector of class factor specifying shared $\boldsymbol{\pi}$ $(t = 1)$ values or a $m \times 1$ numeric matrix specifying a fixed $\boldsymbol{\pi}$ $(t = 1)$ matrix. The string ‘zero’ may be used to specify a column vector of zeros ($\boldsymbol{\pi} = 0$).

Z A vector of class factor specifying which **y** time series correspond to which state time series (in **x**) or a numeric $n \times m$ matrix specifying the **Z** matrix. The string ‘identity’ can be used to specify a $n \times n$ identity matrix and string ‘ones’ may be used to specify a column vector of n ones.

B.3 ML estimation objects: class marssMLE

Objects of class marssMLE specify maximum-likelihood estimation for a MARSS model, both before and after fitting. A minimal marssMLE object contains components `model, start` and `control`, which must be present for estimation by functions like `MARSSkem()`.

model MARSS model specification (an object of class ‘marssm’).

start List with 7 matrices A, R, B, U, Q, x0, V0, specifying initial values for parameters. Dimensions are given in the class marssm section.

control A list specifying estimation options. For `method="kem"`, these are

- *minit* Minimum number of iterations in the maximization algorithm.
- *maxit* Maximum number of iterations in the maximization algorithm.
- *abstol* Optional tolerance for log-likelihood change. If log-likelihood decreases less than this amount relative to the previous iteration, the EM algorithm exits.
- *iter.V0* Maximum number of iterations for final likelihood calculation with V0 = 0.
- *trace* A positive integer. If not zero, a record will be created of each variable the maximization iterations. The information recorded depends on the maximization method.
- *safe* If TRUE, `MARSSkem()` will rerun `MARSSkf()` after each individual parameter update rather than only after all parameters are updated.
- *MCInit* Use Monte Carlo initialization?
- *numInits* Number of random initial value draws.
- *numInitSteps* Number of iterations for each initial value draw.

boundsInits Bounds on the uniform distributions from which initial values will be drawn. (Note that bounds for the covariance matrices Q and R, which require positive values, are specified in logs.)
silent Suppresses printing of progress bar and convergence information.

`MARSSkem()` appends the following components to the marssMLE' object:

method A string specifying the estimation method ('kem' for estimation by `MARSSkem()`).
par A list with 8 matrices of estimated parameter values Z, A, R, B, U, Q, x0, V0. If there are fixed elements in the matrices, the corresponding elements in `$par` are set to the fixed values.
kf A list containing Kalman filter/smoother output. See Chapter 2
numIter Number of iterations required for convergence.
convergence Convergence status.
logLik the exact Log-likelihood. See Section 5.2.
errors any error messages
iter.record record of the parameter values at each iteration (if `control$trace=1`)

Several functions append additional components to the 'marssMLE' object passed in. These include:

`MARSSaic` Appends `AIC, AICc, AICbb, AICbp`, depending on the AIC flavors requested.
`MARSShessian` Appends `Hessian, gradient, parMean` and `parSigma`.
`MARSSparamCIs` Appends `par.se, par.bias, par.upCI` and `par.lowCI`.

C

Package MARSS: The top-level MARSS functions and the base functions

Package MARSS includes functions for estimating Multivariate Autoregressive State Space models, obtaining confidence intervals for parameters, and calculating Akaike's Information Criterion (AIC) for model selection. To make the package both flexible and easy to use, it is designed in two levels. At the base level, the programmer can interact directly with the estimation functions, using two kinds of R objects: objects of the model specification class 'marssm', and objects of estimation classes such as 'marssMLE'. At the user level, the `MARSS()` function allows model estimation with just one function call, hiding the details for ease of use. Users create models in an intuitive way by specifying constraints; the `MARSS()` function then converts these constraints into the object structures required by the estimation functions, performing error checking as necessary.

The two-level package structure allows new users convenient access to the underlying functions, while maintaining flexibility to incorporate different applications and algorithms. Developers can use the base object types to write new functions for their own modeling applications.

To use `MARSS()`, the user specifies a model by supplying the constraint argument to `MARSS()`, using the method argument to specify an estimation method. Optionally, the user may provide initial values for the free parameters, and specify estimation options; for details see the `MARSS()` help file. The function returns an object containing the model, parameter values and estimation details. The user may pass the returned object to `MARSSboot()`, which generates bootstrap parameter estimates, or to `MARSSaic()`, which calculates various versions of AIC for model selection.

Figure 1 shows the underlying base level operations `MARSS()`performs. The function creates a wrapper object of class 'popWrap'. It then calls the `as.marssm( )` method for 'popWrap' to create a `marssm` model specification object from the constraints provided. This model object, initial values and control information are the minimal information required by the estimation functions, and are combined into an object of class appropriate for the estimation method. The estimation function adds to this object the estimated parameter values, estimation details, and other function-specific components, and then returns the augmented object.

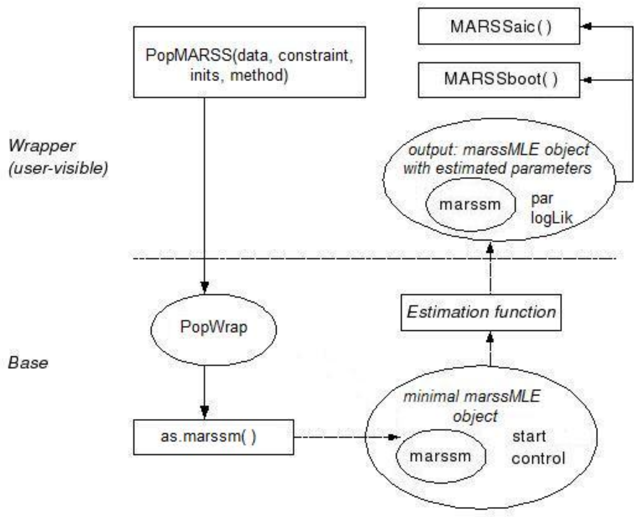

Fig. C.1. Two-level structure of the MARSS package. Rectangles represent functions; ovals represent objects.

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.

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.

HARVEY, A. C. 1989. Forecasting, structural time series models and the Kalman filter. Cambridge University Press, Cambridge, UK.

HARVEY, A. C. AND SHEPHARD, N. 1993. Structural time series models. *In* G. S. Maddala, C. R. Rao, and H. D. Vinod (eds.), Handbook of Statistics, Volume 11. 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. *In* R. S. Cantrell, C. Cosner, and S. Ruan (eds.), Spatial Ecology. 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. 2010. 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. 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.

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.

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.

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.

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.

Ward, E. J., Chirakkal, H., González-Suárez, M., Aurioles-Gamboa, D., Holmes, E. E., and Gerber, L. 2009. 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.

Index