Fitting univariate state-space models

This lab shows you how to fit some basic univariate state-space models using MARSS(). This will also introduce you to the idea of writing AR models in state-space form. Before running the code, load the required R packages:

```r
library(MARSS)
library(stats)
library(forecast)
```

1.1 Fitting a state-space model with MARSS

The `MARSS` package fits multivariate auto-regressive models of this form:

\[
\begin{align*}
    \mathbf{x}_t &= \mathbf{Bx}_{t-1} + \mathbf{u} + \mathbf{w}_t \quad \text{where } \mathbf{w}_t &\sim \mathcal{N}(0, \mathbf{Q}) \\
    \mathbf{y}_t &= \mathbf{Zx}_t + \mathbf{a} + \mathbf{v}_t \quad \text{where } \mathbf{v}_t &\sim \mathcal{N}(0, \mathbf{R}) \\
    \mathbf{x}_0 &= \mu
\end{align*}
\] (1.1)

To fit your time-series model with `MARSS` you need to put your model in that form. The \( \mathbf{B}, \mathbf{Z}, \mathbf{u}, \mathbf{a}, \mathbf{Q}, \mathbf{R} \) and \( \mu \) are parameters that are (potentially) estimated. The \( \mathbf{y} \) are your data. The \( \mathbf{x} \) are the hidden state(s). Everything in bold is a matrix; if it is a small bolded letter, it is a matrix with 1 column.

**Important:** in a state-space model, \( \mathbf{y} \) is always the data and \( \mathbf{x} \) is something estimated from the data.

A basic `MARSS()` call looks like `fit=MARSS(y, model=list(...)`. The `model` argument tells `MARSS` what the parameters look like. The list has the elements with the names: \( \mathbf{B}, \mathbf{u}, \mathbf{Q}, \mathbf{Z}, \mathbf{a}, \mathbf{R}, \mathbf{x0} (\mu) \). The names correspond to the parameters with the same names in Equation (1.1) except that \( \mu \) is called \( \mathbf{x0} \). `tinitx` indicates whether the initial \( \mathbf{x} \) is specified at \( t = 0 \) so \( \mathbf{x}_0 \) or \( t = 1 \) so \( \mathbf{x}_1 \).

Here’s an example. Let’s say we want to fit a univariate AR-1 model observed with error. Here is that model:
1 Univariate state-space models

\[ x_t = b x_{t-1} + w_t \text{ where } w_t \sim N(0,q) \]
\[ y_t = x_t + v_t \text{ where } v_t \sim N(0,r) \]
\[ x_0 = \mu \] (1.2)

To fit this with `MARSS()`, we need to write Equation 1.2 as Equation 1.1. Equation 1.1 is in MATRIX form. The `MARSS()` model list wants the parameters written EXACTLY like they would be written for Equation 1.1. For example, 1 is the number 1 in R. It is not a matrix:

```r
class(1)
[1] "numeric"
```

If you need a 1 (or 0) in your model, you need to pass in the parameter as a 1 × 1 matrix: `matrix(1)`.

With that mind, our model list for Equation 1.2 is:

```r
mod.list=list(  
  B=matrix(1), U=matrix(0), Q=matrix("q"),  
  Z=matrix(1), A=matrix(0), R=matrix("r"),  
  x0=matrix("mu"), tinitx=0 )
```

We can simulate some AR-1 plus error data like so

```r
q=0.1; r=0.1; n=100
y=cumsum(rnorm(n,0,sqrt(q)))+rnorm(n,0,sqrt(r))
```

And then fit with `MARSS()` using our `mod.list` above:

```r
fit=MARSS(y,model=mod.list)
```

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

**MARSS fit is**

Estimation method: kem
Convergence test: conv.test.slope.tol = 0.5, abstol = 0.001
Estimation converged in 17 iterations.
Log-likelihood: -84.82259
AIC: 175.6452  AICc: 175.8952

<table>
<thead>
<tr>
<th>Estimate</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>R.r</td>
<td>0.120</td>
</tr>
<tr>
<td>Q.q</td>
<td>0.125</td>
</tr>
<tr>
<td>x0.mu</td>
<td>0.296</td>
</tr>
</tbody>
</table>

Initial states (x0) defined at t=0

Standard errors have not been calculated. Use `MARSSparamCIs` to compute CIs and bias estimates.
1.2 Examples using the Nile river data

We will use the data from the Nile River (Figure 1.1). We will fit different flow models to the data and compare the models with AIC.

```
library(datasets)
dat=as.vector(Nile)
```

Fig. 1.1. The Nile River flow volume 1871 to 1970 (included dataset in R).

1.2.1 Flat level model

We will start by modeling these data as a simple average river flow with variability around some level $\mu$. 

```r
If we wanted to fix $q = 0.1$, then $Q = [0.1]$ (a $1 \times 1$ matrix with 0.1). We just change `mod.list$Q` and re-fit:

```r
mod.list$Q = matrix(0.1)
fit = MARSS(y, model=mod.list)
```
Univariate state-space models

\[ y_t = \mu + v_t \text{ where } v_t \sim \mathcal{N}(0, r) \] (1.3)

where \( y_t \) is the river flow volume at year \( t \).

We can write this model as a univariate state-space model as follows. We use \( x_t \) to model the average flow level. \( y_t \) is just an observation of this flat \( x_t \). Work through \( x_1, x_2, \ldots \) starting from \( x_0 \) to convince yourself that \( x_t \) will always equal \( \mu \).

\[ x_t = 1 \times x_{t-1} + 0 + w_t \text{ where } w_t \sim \mathcal{N}(0, 0) \]
\[ y_t = 1 \times x_t + 0 + v_t \text{ where } v_t \sim \mathcal{N}(0, r) \] (1.4)

The model is specified as a list as follows:

```r
mod.nile.0 = list(
  B=matrix(1), U=matrix(0), Q=matrix(0),
  Z=matrix(1), A=matrix(0), R=matrix("r"),
  x0=matrix("mu"), tinitx=0 )
```

We then fit the model with `MARSS()`.

```r
kem.0 = MARSS(dat, model=mod.nile.0)
```

Output not shown, but here are the estimates and AICc.

```r
c(coef(kem.0, type="vector"), LL=kem.0$logLik, AICc=kem.0$AICc)
```

| R.r x0.mu LL AICc |
|-----------------|-----------------|-----------------|-----------------|
| 28351.5675      | 919.3500        | -654.5157       | 1313.1552       |

1.2.2 Linear trend in flow model

Figure 1.2 shows the fit for the flat average river flow model. Looking at the data, we might expect that a declining average river flow would be better. In MARSS form, that model would be:

\[ x_t = 1 \times x_{t-1} + u + w_t \text{ where } w_t \sim \mathcal{N}(0, 0) \]
\[ y_t = 1 \times x_t + 0 + v_t \text{ where } v_t \sim \mathcal{N}(0, r) \] (1.5)

where \( u \) is now the average per-year decline in river flow volume. The model is specified as follows:

```r
mod.nile.1 = list(
  B=matrix(1), U=matrix("u"), Q=matrix(0),
  Z=matrix(1), A=matrix(0), R=matrix("r"),
  x0=matrix("mu"), tinitx=0 )
```

We then fit the model with `MARSS()`:
1.2 Examples using the Nile river data

\[ \text{kem.1} = \text{MARSS(dat, model=mod.nile.1)} \]

Here are the estimates, log-likelihood and AICc:

\[
\begin{array}{cccc}
\text{R.r} & \text{U.u} & \text{x0.mu} & \text{LL} \\
22213.595453 & -2.692106 & 1054.935067 & -642.315910 \\
\text{AICc} & 1290.881821 \\
\end{array}
\]

Figure 1.2 shows the fits for the two models with deterministic models (flat and declining) for mean river flow along with their AICc values (smaller AICc is better). The AICc for the model with a declining river flow is lower by over 20 (which is a lot).

1.2.3 Stochastic level model

Looking at the flow levels, we might suspect that a model that allows the average flow to change would model the data better and we might suspect that there have been sudden, and anomalous, changes in the river flow level. We will now model the average river flow at year \( t \) as a random walk, specifically an autoregressive process which means that average river flow is year \( t \) is a function of average river flow in year \( t-1 \).

\[
\begin{align*}
\text{x}_t &= \text{x}_{t-1} + \text{w}_t \quad \text{where} \quad \text{w}_t \sim \text{N}(0, q) \\
\text{y}_t &= \text{x}_t + \text{v}_t \quad \text{where} \quad \text{v}_t \sim \text{N}(0, r) \\
\text{x}_0 &= \mu
\end{align*}
\]

As before, \( y_t \) is the river flow volume at year \( t \). \( x_t \) is the mean level. The model is specified as:

\[
\text{mod.nile.2} = \text{list(} \\
\text{B=matrix(1), U=matrix(0), Q=matrix("q"),} \\
\text{Z=matrix(1), A=matrix(0), R=matrix("r"),} \\
\text{x0=matrix("mu"), tinitx=0 })
\]

We could also use the text shortcuts to specify the model. Because \( \text{R} \) and \( \text{Q} \) are \( 1 \times 1 \) matrices, “unconstrained”, “diagonal and unequal”, “diagonal and equal” and “equalvarcov” will all lead to a \( 1 \times 1 \) matrix with one estimated element. For \( a \) and \( u \), the following shortcut could be used:

\[
\text{A=U="zero"}
\]

Because \( x_0 \) is \( 1 \times 1 \), it could be specified as “unequal”, “equal” or “unconstrained”.

\[
\text{kem.2} = \text{MARSS(dat, model=mod.nile.2)}
\]

Here are the estimates, log-likelihood and AICc:
\begin{verbatim}
c(coef(kem.2, type="vector"), LL=kem.2$logLik, AICc=kem.2$AICc)
R.r Q.q x0.mu LL AICc
15065.6121 1425.0030 1111.6338 -637.7631 1281.7762
\end{verbatim}

1.2.4 Stochastic level model with drift

We can add a drift to term to our random walk; the \( u \) in the process model \((x)\) is the drift term. This causes the random walk to tend to trend up or down.

\[
\begin{align*}
x_t &= x_{t-1} + u + w_t \quad \text{where } w_t \sim N(0, q) \\
y_t &= x_t + v_t \quad \text{where } v_t \sim N(0, r) \\
x_0 &= \mu
\end{align*}
\]

(1.7)

The model is then specified by changing \( U \) to indicate that a \( u \) is estimated:

\begin{verbatim}
mod.nile.3 = list(  
  B=matrix(1), U=matrix("u"), Q=matrix("q"),  
  Z=matrix(1), A=matrix(0), R=matrix("r"),  
  x0=matrix("mu"), tinitx=0)
\end{verbatim}

\begin{verbatim}
kem.3 = MARSS(dat, model=mod.nile.3)
\end{verbatim}

Here are the estimates, log-likelihood and AICc:

\begin{verbatim}
c(coef(kem.3, type="vector"), LL=kem.3$logLik, AICc=kem.3$AICc)
R.r U.u Q.q x0.mu LL AICc
15585.278194 -3.248793 1088.987455 1124.044484
-637.302692 1283.026436
\end{verbatim}

Figure 1.2 shows all the models along with their AICc values.

1.3 The StructTS function

The \texttt{StructTS} function in R will also fit the stochastic level model:

\begin{verbatim}
fit.sts = StructTS(dat, type="level")
fit.sts
\end{verbatim}

Call:
\texttt{StructTS(x = dat, type = "level")}

Variances:
\begin{verbatim}
  level  epsilon
1469  15099
\end{verbatim}
The estimates from `StructTS()` will be different (though similar) from `MARSS()` because `StructTS()` uses \( x_1 = y_1 \), that is the hidden state at \( t = 1 \) is fixed to be the data at \( t = 1 \). That is fine if you have a long data set, but would be disastrous for the short data sets typical in fisheries and ecology.

`StructTS()` is much, much faster for long time series. The example in `?StructTS` is pretty much instantaneous with `StructTS()` but takes minutes with the EM algorithm that is the default in `MARSS()`. With the BFGS algorithm, it is much closer to `StructTS()`:

```r
trees <- window(treering, start = 0)
fitts = StructTS(trees, type = "level")
fitem = MARSS(as.vector(trees),mod.nile.2)
fitbf = MARSS(as.vector(trees),mod.nile.2, method="BFGS")
```

Note that `mod.nile.2` specifies a univariate stochastic level model so we can use it just fine with other univariate data sets.

In addition, `fitted(fit.sts)` where `fit.sts` is a fit from `StructTS()` is very different than `fit.marss$states` from `MARSS()`.

```r
t=10
fitted(fit.sts)[t]
```

```
[1] 1162.904
```

is the expected value of \( y_{t+1} \) (in this case \( y_{11} \) since we set \( t = 10 \)) given the data up to \( y_t \) (in this case, up to \( y_{10} \)). It is called the one-step ahead prediction. We are not going to use the one-step ahead predictions unless we are forecasting or doing cross-validation.

Typically, when we analyze fisheries and ecological data, we want to know the estimate of the state, the \( x_t \), given ALL the data. For example, we might need an estimate of the population size in year 1990 given a time series of counts from 1930 to 2015. We don’t want to use only the data up to 1989; we want to use all the information. `fit.marss$states` from `MARSS()` is the expected value of \( x_t \) given all the data. For the stochastic level model, that is equal to the expected value of \( y_t \) given all the data except \( y_t \).

If you needed the one-step predictions from `MARSS`, you can get them from the Kalman filter output:

```r
kf=print(kem.2, what="kfs")
kf$ktt1[1,t]
```

`kfs` means Kalman filter/smoothers and passing in `what="kfs"` is saying to return the Kalman filter/smoothers output. The expected value of \( x_t \) conditioned on \( y_1 \) to \( y_t-1 \) is in `ktt1` of that output. The expected value of \( x_t \) conditioned on all the data is in `kttT`. 
1.4 Comparing models with AIC and model weights

To get the AIC or AICc values for a model fit from a MARSS fit, use `fit$AIC` or `fit$AICc`. The log-likelihood is in `fit$logLik` and the number of estimated parameters in `fit$num.params`. For fits from other functions, try `AIC(fit)` or look at the function documentation.

Let’s put the AICc values 3 Nile models together:

```r
nile.aic = c(kem.0$AICc, kem.1$AICc, kem.2$AICc, kem.3$AICc)
```
Then we calculate the AICc minus the minus AICc in our model set and compute the model weights. ΔAIC is the AIC values minus the minimum AIC value in your model set.

\[
\Delta AIC = \text{nile.aic} - \text{min(nile.aic)} \\
\text{relLik} = \exp(-0.5*\Delta AIC) \\
\text{aicweight} = \frac{\text{relLik}}{\text{sum(relLik)}}
\]

And this leads to our model weights table:

```
aic.table=data.frame( 
AICc=nile.aic, 
\Delta AIC=\Delta AIC, 
relLik=relLik, 
weight=aicweight) 
rownames(aic.table)=c("flat level","linear trend", "stoc level", "stoc level w drift")
```

Here the table is printed using `round()` to limit the number of digits shown.

```
round(aic.table, digits=3)
```

<table>
<thead>
<tr>
<th></th>
<th>AICc</th>
<th>\Delta AIC</th>
<th>relLik</th>
<th>weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>flat level</td>
<td>1313.155</td>
<td>31.379</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>linear trend</td>
<td>1290.882</td>
<td>9.106</td>
<td>0.011</td>
<td>0.007</td>
</tr>
<tr>
<td>stoc level</td>
<td>1281.776</td>
<td>0.000</td>
<td>1.000</td>
<td>0.647</td>
</tr>
<tr>
<td>stoc level w drift</td>
<td>1283.026</td>
<td>1.250</td>
<td>0.535</td>
<td>0.346</td>
</tr>
</tbody>
</table>

One thing to keep in mind when comparing models within a set of models is that the model set needs to include at least one model that can fit the data reasonably well. ‘Reasonably well’ means the model can put a fitted line through the data. Can’t all models do that? Definitely, not. For example, the flat-level model cannot put a fitted line through the Nile River data. It is simply impossible. The straight trend model also cannot put a fitted line through the flow data. So if our model set only included flat-level and straight trend, then we might have said that the straight trend model is ‘best’ even though it is just the better of two bad models.

### 1.5 Basic diagnostics

The first diagnostic that you do with any statistical analysis is check that your residuals correspond to your assumed error structure. We have two types of errors in a univariate state-space model: process errors, the $w_t$, and observation errors, the $v_t$. They should not have a temporal trend. To get the residuals from most types of fits in R, you can use `residuals(fit)`. MARSS calls the $v_t$, model residuals, and the $w_t$, state residuals. We can plot these using the following code (Figure 1.3).
The residuals should also not be autocorrelated in time. We can check the autocorrelation with \texttt{acf}. We won’t do this for the state residuals for the flat level or linear trends since for those models $w_t = 0$. The acf’s are shown in Figure 1.4. The stochastic level model looks the best in that its model residuals (the $v_t$) are fine but the state model still has problems. Clearly the state is not a simple random walk. This is not surprising. The Aswan Low Dam was completed in 1902 and changed the mean flow. The Aswan High Dam was completed in 1970 and also affected the flow. You can see these perturbations in Figure 1.1.
par(mfrow=c(2,2))
resids=residuals(kem.0)
acf(resids$model.residuals[1,], main="flat level v(t)")
resids=residuals(kem.1)
acf(resids$model.residuals[1,], main="linear trend v(t)")
resids=residuals(kem.2)
acf(resids$model.residuals[1,], main="stoc level v(t)")
acf(resids$state.residuals[1,], main="stoc level w(t)")

Fig. 1.4. The model and state residual acfs for the 3 models.
1.6 Fitting a univariate AR-1 state-space model with JAGS*

Here we show how to fit model 3, Equation 1.7, with JAGS. First load the required packages:

```r
library(coda)
library(R2jags)
```

Next write the model for JAGS to a file (filename in `model.loc`):

```r
cmp = "ss_model.txt"
jagsscript = cat("model {
  # priors on parameters
  mu ~ dnorm(Y1, 1/(Y1*100)); # normal mean = 0, sd = 1/sqrt(0.01)
  tau.q ~ dgamma(0.001,0.001); # This is inverse gamma
  sd.q <- 1/sqrt(tau.q); # sd is treated as derived parameter
  tau.r ~ dgamma(0.001,0.001); # This is inverse gamma
  sd.r <- 1/sqrt(tau.r); # sd is treated as derived parameter
  u ~ dnorm(0, 0.01);

  # Because init X is specified at t=0
  X0 <- mu
  X[1] ~ dnorm(X0+u,tau.q);
  Y[1] ~ dnorm(X[1], tau.r);

  for(i in 2:N) {
    predX[i] <- X[i-1]+u;
    X[i] ~ dnorm(predX[i],tau.q); # Process variation
    Y[i] ~ dnorm(X[i], tau.r); # Observation variation
  }
},file=model.loc"
```

Next we specify the data (and any other input) that the JAGS code needs. In this case, we need to pass in `dat` and the number of time steps since that is used in the for loop. We also specify the parameters that we want to monitor. We need to specify at least one, but we will monitor all of them so we can plot them after fitting. Note, that the hidden state is a parameter in the Bayesian context (but not in the maximum likelihood context).

```r
jags.data = list("Y"=dat, "N"=length(dat), Y1=dat[1])
jags.params=c("sd.q","sd.r","X","mu", "u")
```

Now we can fit the model:

```r
mod_ss = jags(jags.data, parameters.to.save=jags.params, model.file=model.loc, n.chains = 3, n.burnin=5000, n.thin=1, n.iter=10000, DIC=TRUE)
```
We can then show the posteriors with the MLEs from MARSS on top:

```r
attach.jags(mod_ss)
par(mfrow=c(2,2))
hist(mu)
abline(v=coef(kem.3)$x0, col="red")
hist(u)
abline(v=coef(kem.3)$U, col="red")
hist(log(sd.q^2))
abline(v=log(coef(kem.3)$Q), col="red")
hist(log(sd.r^2))
abline(v=log(coef(kem.3)$R), col="red")
detach.jags()
```

**Fig. 1.5.** The posteriors for model 3 with MLE estimates from MARSS shown in red.

To plot the estimated states, we write a helper function:

```r
plotModelOutput = function(jagsmodel, Y) {
  attach.jags(jagsmodel)
  x = seq(1,length(Y))
  ```
Fig. 1.6. The estimated states from the Bayesian fit along with 95% credible intervals (black and grey) with the MLE states and 95% confidence intervals in red.
1.7 A simple random walk model of animal movement

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

\[ x_{1,t} = x_{1,t-1} + u_1 + w_{1,t}, \quad w_{1,t} \sim N(0, \sigma_1^2) \]  

\[ x_{2,t} = x_{2,t-1} + u_2 + w_{2,t}, \quad w_{2,t} \sim N(0, \sigma_2^2) \]  

where \( x_{1,t} \) is the location at time \( t \) along one axis (here, longitude) and \( x_{2,t} \) is for another, generally orthogonal, axis (in here, latitude). The parameter \( u_1 \) is the rate of longitudinal movement and \( u_2 \) is the rate of latitudinal movement.

We add errors to our observations of location:

\[ y_{1,t} = x_{1,t} + v_{1,t}, \quad v_{1,t} \sim N(0, \eta_1^2) \]  

\[ y_{2,t} = x_{2,t} + v_{2,t}, \quad v_{2,t} \sim N(0, \eta_2^2) \]

This model is comprised of two separate univariate state-space models. Note that \( y_1 \) depends only on \( x_1 \) and \( y_2 \) depends only on \( x_2 \). There are no actual interactions between these two univariate models. However, we can write the model down in the form of a multivariate model using diagonal variance-covariance matrices and a diagonal design (Z) matrix. Because the variance-covariance matrices and Z are diagonal, the \( x_1:y_1 \) and \( x_2:y_2 \) processes will be independent as intended. Here are Equations 1.9 and 1.11 written as a MARSS model (in matrix form):

\[
\begin{bmatrix}
    x_{1,t} \\
    x_{2,t}
\end{bmatrix} = \begin{bmatrix}
    x_{1,t-1} \\
    x_{2,t-1}
\end{bmatrix} + \begin{bmatrix}
    u_1 \\
    u_2
\end{bmatrix} + \begin{bmatrix}
    w_{1,t} \\
    w_{2,t}
\end{bmatrix}, \quad w_t \sim MVN\left(0, \begin{bmatrix}
    \sigma_1^2 & 0 \\
    0 & \sigma_2^2
\end{bmatrix}\right) \tag{1.12}
\]

\[
\begin{bmatrix}
    y_{1,t} \\
    y_{2,t}
\end{bmatrix} = \begin{bmatrix}
    1 & 0 \\
    0 & 1
\end{bmatrix} \begin{bmatrix}
    x_{1,t} \\
    x_{2,t}
\end{bmatrix} + \begin{bmatrix}
    v_{1,t} \\
    v_{2,t}
\end{bmatrix}, \quad v_t \sim MVN\left(0, \begin{bmatrix}
    \eta_1^2 & 0 \\
    0 & \eta_2^2
\end{bmatrix}\right) \tag{1.13}
\]

The variance-covariance matrix for \( \mathbf{w} \) is a diagonal matrix with unequal variances, \( \sigma_1^2 \) and \( \sigma_2^2 \). The variance-covariance matrix for \( \mathbf{v} \) is a diagonal matrix with unequal variances, \( \eta_1^2 \) and \( \eta_2^2 \). We can write this succinctly as

\[
\mathbf{x}_t = \mathbf{x}_{t-1} + \mathbf{u} + \mathbf{w}_t, \quad \mathbf{w}_t \sim MVN(0, \mathbf{Q}) \tag{1.14}
\]

\[
\mathbf{y}_t = \mathbf{x}_t + \mathbf{v}_t, \quad \mathbf{v}_t \sim MVN(0, \mathbf{R}). \tag{1.15}
\]
Problems

1.1 Write the equations for each of these models: ARIMA(0,0,0), ARIMA(0,1,0), ARIMA(1,0,0), ARIMA(0,0,1), ARIMA(1,0,1). Read the help file for Arima (forecast package) if you are fuzzy on the arima notation.

1.2 The MARSS package includes a data set of sharp-tailed grouse in Washington. Load the data to use as follows:

library(MARSS)
dat=log(grouse[,2])

Consider these two models for the data:
Model 1 random walk with no drift observed with no error
Model 2 random walk with drift observed with no error

Written as a univariate state-space model, model 1 is

\[ x_t = x_{t-1} + w_t \] where \( w_t \sim N(0, q) \)
\[ x_0 = a \]
\[ y_t = x_t \] (1.16)

Model 2 is almost identical except with \( u \) added

\[ x_t = x_{t-1} + u + w_t \] where \( w_t \sim N(0, q) \)
\[ x_0 = a \]
\[ y_t = x_t \] (1.17)

\( y \) is the log grouse count in year \( t \).

a) Plot the data. The year is in column 1 of grouse.
b) Fit each model using MARSS().
c) Which one appears better supported given AICc?
d) Load the forecast package. Use \?auto.arima to learn what it does.

Then use auto.arima(dat) to fit the data. Next run auto.arima on the data with trace=TRUE to see all the ARIMA models it compared.

Note, ARIMA(0,1,0) is a random walk with \( b=1 \). ARIMA(0,1,0) with drift would be a random walk (\( b=1 \)) with drift (with \( u \)).
e) Is the difference in the AICc values between a random walk with and without drift comparable between MARSS() and auto.arima()? 

Note when using auto.arima, it will fit AR-1 models of the following form (notice the \( b \)): \( x_t = bx_{t-1} + w_t \). auto.arima refers this model \( x_t = x_{t-1} + w_t \), which is also AR-1 but with \( b=1 \), as ARIMA(0,1,0). This says that the first difference of the data (that’s the 1 in the middle) is a ARMA(0,0) process (the 0s in the 1st and 3rd spots). So ARIMA(0,1,0) means this: \( x_t - x_{t-1} = w_t \). 

1.3 Create a random walk with drift time series using the following command:
dat=cumsum(rnorm(100,0.1,1))
Use ?rnorm to make sure you know what the arguments to rnorm are.

a) Write out the equation for that random walk as a univariate state-space model.

b) What is the order of this random walk written as ARIMA(p, d, q)?
   "what is the order" means "what is p, d, and q. Model "order" is how
   arima and Arima specify arima models.

c) Fit that model using Arima() in the forecast package. You’ll need
to specify the order and include.drift term. Use ?Arima to review
what that function does if needed.

d) Fit that model with MARSS().

e) How are the two estimates different?

Now fit the first-differenced data:

diff.dat=diff(dat)
Use ?diff to check what the diff function does.

f) If \( x_t \) denotes a time series. What is the first difference of \( x \)? What is
the second difference?

g) What is the \( x \) model for diff.dat? Look at your answer to part (a)
and the answer to part (e).

h) Fit diff.dat using Arima(). You’ll need to change order and include.mean.
i) Fit that model with MARSS().

If you’ve done this right, the estimated parameters using Arima and MARSS
will now be the same.

This question should clue you into the fact that Arima is not exactly fitting
Equation 1.1. It’s very similar, but not quite written that way. By the way,
Equation 1.1 is how structural time series observed with error are written
(state-space models). To recover the estimates that a function like arima
or Arima returns, you need to write your state-space model in a specific
way (as seen above).

1.4 Arima() will also fit what it calls an ‘AR-1 with drift’. An AR-1 with drift
is NOT this model:

\[
x_t = bx_{t-1} + u + w_t \quad \text{where} \quad w_t \sim N(0, q)
\]  \hspace{1cm} (1.18)

The population dynamics literature, this equation is called the Gompertz
model and is a type of density-dependent population model.

a) Write R code to simulate Equation 1.18. Make \( b \) less than 1 and greater
than 0. Set \( u \) and \( x_0 \) to whatever you want. You can use a for loop.

b) Plot the trajectories and show that this model does not “drift” upward
or downward. It fluctuates about a mean value.

c) Hold \( b \) constant and change \( u \). How do the trajectories change?

d) Hold \( u \) constant and change \( b \). Make sure to use a \( b \) close to 1 and
another close to 0. How do the trajectories change?
18 1 Univariate state-space models

e) Do 2 simulations each with the same \( w_t \). In one simulation, set \( u = 1 \)
and in the other \( u = 2 \). For both simulations, set \( x_1 = u/(1-b) \). You can
set \( b \) to whatever you want as long as \( 0 < b < 1 \). Plot the 2 trajectories
on the same plot. What is different?

We will fit what \texttt{arima} calls “AR-1 with drift” models in the chapter on
MARSS models with covariates.

1.5 The MARSS package includes a data set of gray whales. Load the data to
use as follows:

```r
library(MARSS)
dat=log(graywhales[,2])
```

Fit a random walk with drift model observed with error to the data:

\[
\begin{align*}
x_t &= x_{t-1} + u + w_t \quad \text{where} \quad w_t \sim N(0,q) \\
y_t &= x_t + v_t \quad \text{where} \quad v_t \sim N(0,r)
\end{align*}
\]

(1.19) \( x_0 = a \)

\( y \) is the whale count in year \( t \). \( x \) is interpreted as the 'true' unknown
population size that we are trying to estimate.

a) Fit this model with \texttt{MARSS()}.
b) Plot the estimated \( x \) as a line with the actual counts added as points.
   \( x \) is in \texttt{fit$states}. It is a matrix, which \texttt{plot} will not like so you will
   need to change it to a vector using \texttt{as.vector()} or \texttt{fit$states[1,]}
c) Simulate 1000 sample trajectories using the estimated \( u \) and \( q \) starting
   at the estimated \( x \) in 1997. You can do this with a couple \texttt{for}
   loops or write something terse with \texttt{cumsum} and \texttt{apply}.
d) Using these simulated trajectories, what is your estimated probability
   of reaching 50,000 graywhales in 2007.
e) What kind of uncertainty does that estimate NOT include?

1.6 Fit the following models to the graywhales data using MARSS(). Assume
\( b = 1 \).

Model 1 Process error only model with drift

Model 2 Process error only model without drift

Model 3 Process error with drift and observation error with observation
error variance fixed = 0.05.

Model 4 Process error with drift and observation error with observation
error variance estimated.

a) Compute the AICc’s for each model and likelihood or deviance \((-2 * \log\ \text{likelihood})\). Where to find these? Try \texttt{names(fit)}. \texttt{logLik()} is the
standard R function to return log-likelihood from fits.
b) Calculate a table of \( \Delta \text{AICc} \) values and AICc weights.
c) Show the acf of the model and state residuals for the best model. You
will need a vector of the residuals to do this. If \texttt{fit} is the fit from a
fit call like \texttt{fit = MARSS(dat)}, you get the residuals using this code:
do the acf's suggest any problems?

1.7 Evaluate the predictive accuracy of forecasts using the `forecast` package using the 'airmiles' dataset. Load the data to use as follows:

```r
library(forecast)
dat = log(airmiles)
n = length(dat)
training.dat = dat[1:(n-3)]
test.dat = dat[(n-2):n]
```

This will prepare the training data and set aside the last 3 data points for validation.

a) Fit the following four models using `Arima()`:
   - ARIMA(0,0,0)
   - ARIMA(1,0,0)
   - ARIMA(0,0,1)
   - ARIMA(1,0,1).

b) Use `forecast()` to make 3 step ahead forecasts from each.

c) Calculate the MASE statistic for each using the `accuracy` function in the forecast package. Type `?accuracy` to learn how to use this function.

d) Present the results in a table.
e) Which model is best supported based on the MASE statistic?

1.8 The WhaleNet Archive of STOP Data has movement data on loggerhead turtles on the east coast of the US from ARGOS tags. The MARSS package `loggerheadNoisy` dataset is lat/lot data on eight individuals, however we have corrupted this data severely by adding random errors in order to create a “bad tag” problem (very noisy). Use `head(loggerheadNoisy)` to get an idea of the data. Then load the data on one turtle, MaryLee. MARSS needs time across the columns to you need to transpose the data (as shown).

```r
turtlename = "MaryLee"
dat = loggerheadNoisy[which(loggerheadNoisy$turtle==turtlename),5:6]
dat = t(dat)
```

a) Plot MaryLee's locations (as a line not dots). Put the latitude locations on the y-axis and the longitude on the y-axis. You can use `rownames(dat)` to see which is in which row. You can just use `plot()` for the homework. But if you want, you can look at the MARSS Manual chapter on animal movement to see how to plot the turtle locations on a map using the `maps` package.

b) Analyze the data with a state-space model (movement observed with error) using

```r
fit0 = MARSS(dat)
```

Look at the output from the above MARSS call. What are each of the parameters output from MARSS?
c) What assumption did the default MARSS model make about observation error and process error?
d) Does MaryLee move faster in the latitude direction versus longitude direction?
e) Add MaryLee’s estimated “true” positions to your plot of her locations. You can use `lines(x, y, col="red")` (with x and y replaced with your x and y data). The true position is the “state”. This is in the states element of an output from MARSS `fit0$states`.
f) Compare the following models for these data. Movement in the lat/lon direction is (1) independent but the variance is the same, (2) is correlated and lat/lon variances are different, and (3) is correlated and the lat/lon variances are the same. You only need to change Q specification. Your MARSS call will now look like with ... replaced with your Q specification.
```
fit1 = MARSS(dat, list(Q=...))
```
g) Plot your state residuals (true location residuals). What are the problems? Discuss in reference to your plot of the location data. Here is how to get state residuals from MARSS:
```
resids = residuals(fit0)$state.residuals
```
The lon residuals are in row 1 and lat residuals are in row 2 (same order as the data).