Time Series Models for EDNA

MARSS Model

Reading in the Data

Here is an example of how you can set up this model, and I imagine have a similar structure to your data using phytoplankton from lake Washington, such that we are looking at the response of blue-green algae to temperatures. For your datasets "bluegreens" will be your salmon abundance and "temperature" will be your eDNA data. Rather than monthly data for multiple years, it will be daily.

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

```
Registered S3 method overwritten by 'quantmod':
  method            from
  as.zoo.data.frame zoo
```

```
library(datasets)
data(lakeWAplankton, package = "MARSS")

fulldat <- lakeWAplanktonTrans
years <- fulldat[, "Year"] >= 1967 & fulldat[, "Year"] < 1974
dat <- t(fulldat[years, c("Bluegreens")]) #to run data in MARSS, your data needs to be "trans

covariates <- t(fulldat[years, c("Temp")])
```

Before you can run the model, both the response and predictor should be z-scored such that they both have a mean of 0 and sd of 1. You can do it this way (which can make it easier to back transform your data later) or you can use the scale() function.

```
# z-score the response variables
the.mean <- apply(dat, 1, mean, na.rm = TRUE)
the.sigma <- sqrt(apply(dat, 1, var, na.rm = TRUE))
dat <- (dat - the.mean) * (1/the.sigma)

the.meanC <- apply(covariates, 1, mean, na.rm = TRUE)
the.sigmaC <- sqrt(apply(covariates, 1, var, na.rm = TRUE))
covariates <- (covariates - the.meanC) * (1/the.sigmaC)
```

Once your data is prepared, you can now assign values to the matrices needed for MARSS. MARSS models are all just matrix math, and how to set these up can be confusing at first. Your model should have a very simple set up since you are fitting one TS at a time. This is a "observation error only" model, it sounds like your data structure should have both process and observation error, so consider this a fallback option! (its also a nice starting place to make sure everything is set up correctly)

```
Q <- "zero" #this is your process error - its 0 which means we will only fit
#observation error

U <- "unconstrained"#this is a matrix that relates your "state" to obs error

x0 <- "zero" #this is your starting value, it shouldn't matter too much for
#your models (basically where the model starts looking for a good "fit")

B <- "identity" #since we are doing process - error only this will be an identity too

Z <- "identity" #this is an identity matrix, since your model is fairly simple
#single population, it will be an "identity"...as you get into fancier
#multivariate stuff it will change

d <- covariates #this is where you put your covariates, right now they are influencing
#your observation error - in reality you will want it to be your process error covariate,
#so when we add that into the model we will move the covariates into the "c" matrices

A <- "zero" #We set A="zero" because the data and covariates have been demeaned

D <- "unconstrained" #this is the coefficient for your covariates.

y <- dat  # your data (salmon abundance)
model.list <- list(B = B, U = U, Q = Q, Z = Z, A = A, D = D,
    d = d, x0 = x0)
obs_only<- MARSS(y, model = model.list)
```

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

MARSS fit is
Estimation method: kem
Convergence test: conv.test.slope.tol = 0.5, abstol = 0.001
Algorithm ran 15 (=minit) iterations and convergence was reached.
Log-likelihood: -107.5872
AIC: 221.1745   AICc: 221.4745

    Estimate
R.R  0.75860
U.U -0.00709
D.D  0.32725
Initial states (x0) defined at t=0

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

```r
R <- A <- U <- "zero" # no observation error and demeaned data...so these are all 0!
B <- "unconstrained"
Z <- "identity"
Q <- "equalvarcov" #this is how your variance-covariance matrix is set up -
#you only have 1 TS!
C <- "unconstrained" #coefficient for your covariates! We have moved them to
#the c matrix so they influence the process rather than observations
c = covariates
model.list <- list(B = B, U = U, Q = Q, Z = Z, A = A, R = R,
    C = C, c = c)
process_only<- MARSS(dat, model = model.list)
```

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

MARSS fit is
Estimation method: kem
Convergence test: conv.test.slope.tol = 0.5, abstol = 0.001
Algorithm ran 15 (=minit) iterations and convergence was reached.
Log-likelihood: -68.75691
AIC: 145.5138   AICc: 146.0201
```

```
      Estimate
B.B      0.790
Q.Q      0.301
x0.x0    0.837
C.C      0.162
Initial states (x0) defined at t=0

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

If it converges with both above...lets try both types of error! If your R does not converge...that means you don't hav enough data to estimate observation error

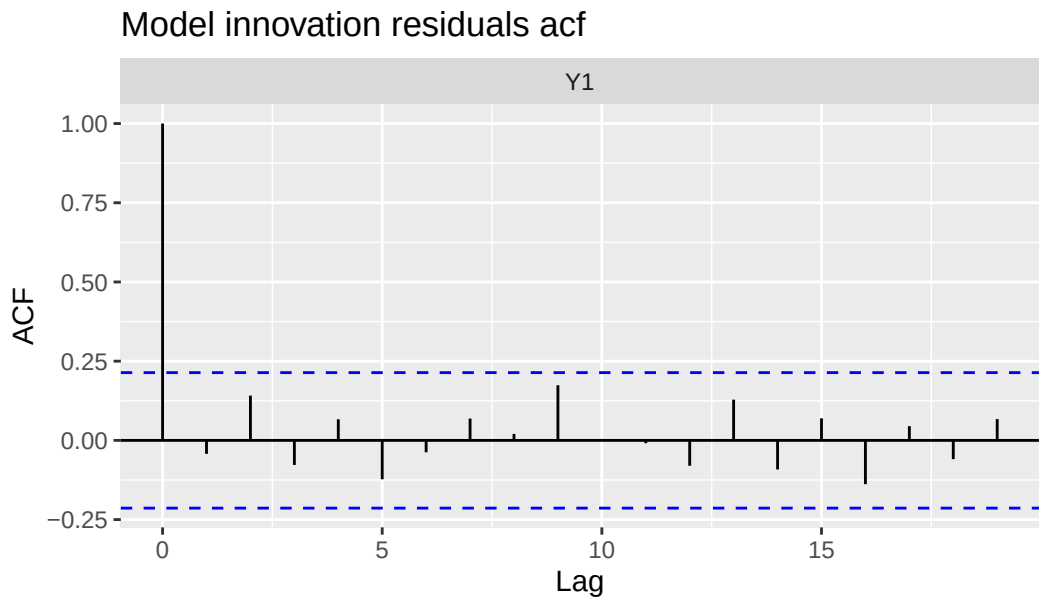
```
D <- d <- A <- U <- "zero" # you do not have covs in the d matrix
Z <- "identity"
B <- "unconstrained"
Q <- "equalvarcov"
C <- "unconstrained"
c <- covariates
R <- "diagonal and equal"
x0 <- "unequal"
tinitx <- 1
model.list <- list(B = B, U = U, Q = Q, Z = Z, A = A, R = R,
    D = D, d = d, C = C, c = c, x0 = x0, tinitx = tinitx)
kem <- MARSS(dat, model = model.list)
```

```
Success! abstol and log-log tests passed at 45 iterations.
Alert: conv.test.slope.tol is 0.5.
Test with smaller values (<0.1) to ensure convergence.
Alert: Numerical warnings were generated. Print the $errors element of output to see the warn

MARSS fit is
Estimation method: kem
Convergence test: conv.test.slope.tol = 0.5, abstol = 0.001
Estimation converged in 45 iterations.
Log-likelihood: -66.99111
AIC: 143.9822   AICc: 144.7515

      Estimate
R.R     0.0557
B.B     0.8308
```

```
Q.Q     0.2087
x0.x0   0.5599
C.C     0.2085
Initial states (x0) defined at t=1

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

MARSSkem warnings. Type MARSSinfo() for help.
 MARSSkem: The solution became unstable and logLik DROPPED. Try MARSSinfo('LLdropped') for i

Use control$trace=1 to generate a more detailed error report.
```

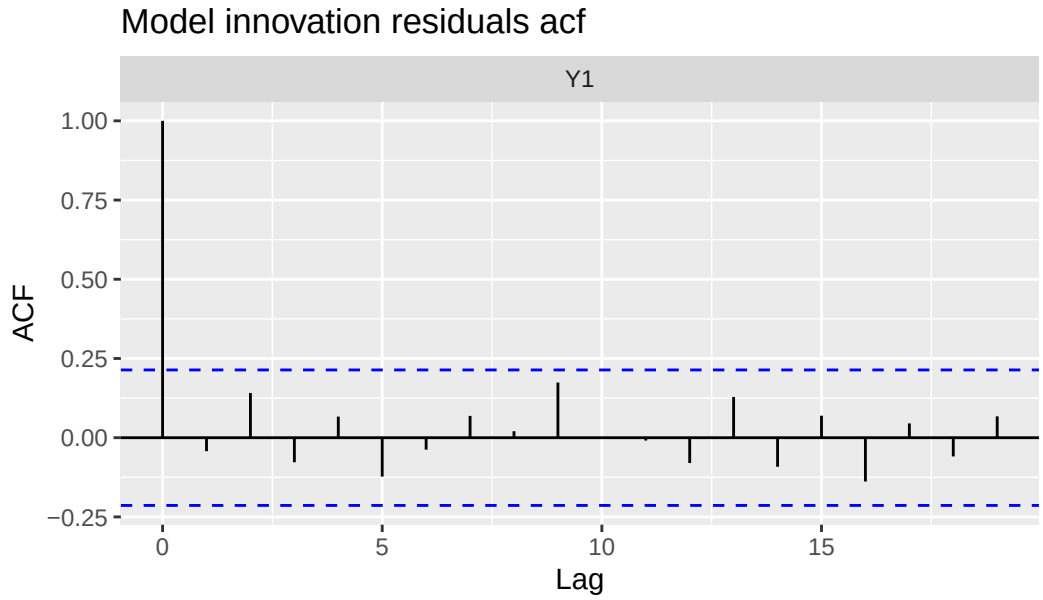
Lets take a look at the residuals - running acf on the residuals will tell us if the error is autocorrelated, they are not! Where a vertical line exceeds the dashed line tells you where the autocorrelation ends. For this model, we are good! The last vertical line to excede the dashed line is how many time steps you want to lag your data

```
resids <- residuals(kem)
autoplot(resids, plot.type = "acf")
```



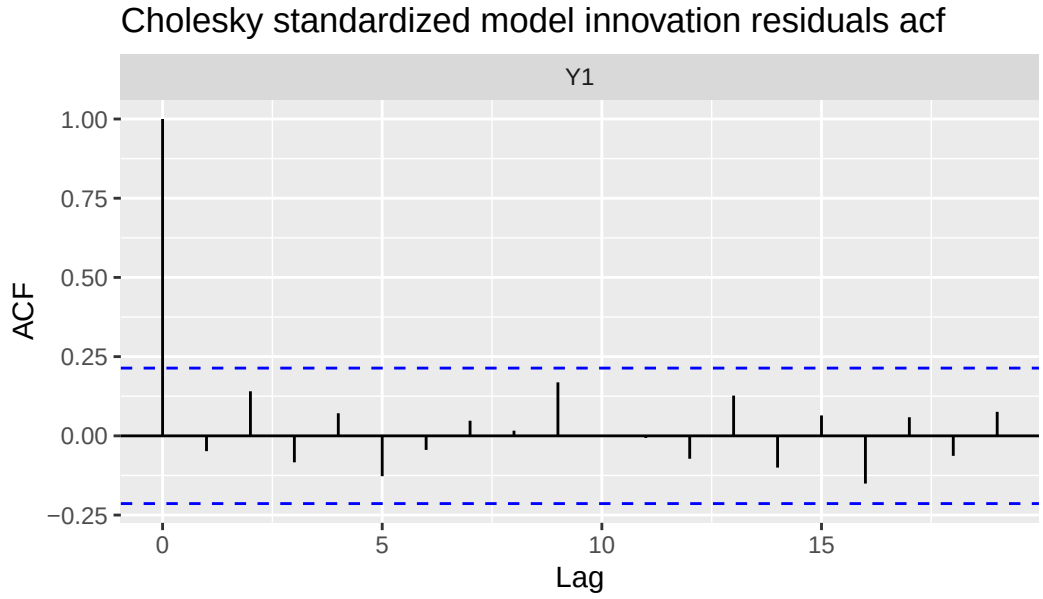
Model innovation (ytt1) residuals. These residuals should be temporally uncorrelated.

```
plot.type = acf.model.resids.ytt1
```



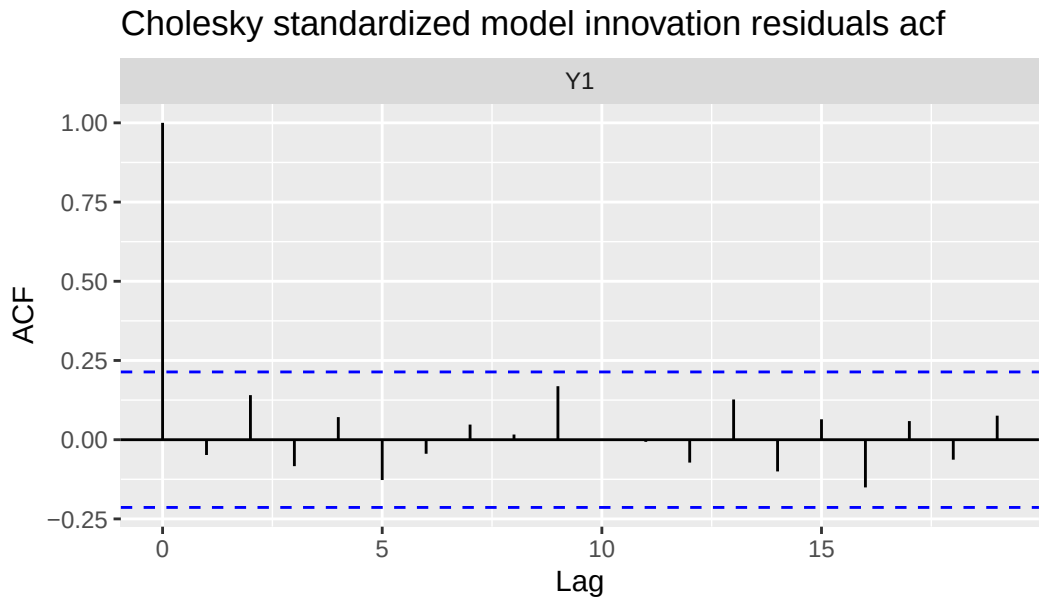
Model innovation (ytt1) residuals. These residuals should be temporally uncorrelated.

```
Hit <Return> to see next plot (q to exit):
```



Cholesky standardized model innovation (ytt1) residuals. These residuals should be temporally uncorrelated.

```
plot.type = acf.std.model.resids.ytt1
```

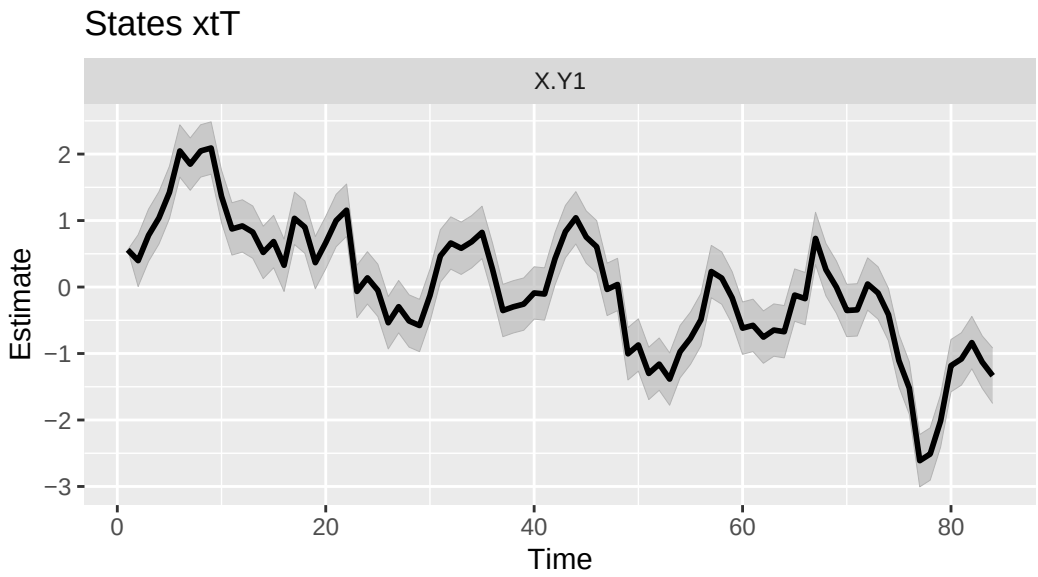


Cholesky standardized model innovation (ytt1) residuals. These residuals should be temporally uncorrelated.

```
Finished plots.
```

Lets look at our "states". THis is our estimated model of bluegreen algae based as a function of the predictor temperature

```
ggplot2::autoplot(kem, plot.type = "xtT")
```



Now, if you are not sure how to set up your matrix structures your can always try different ones and compare with AIC!