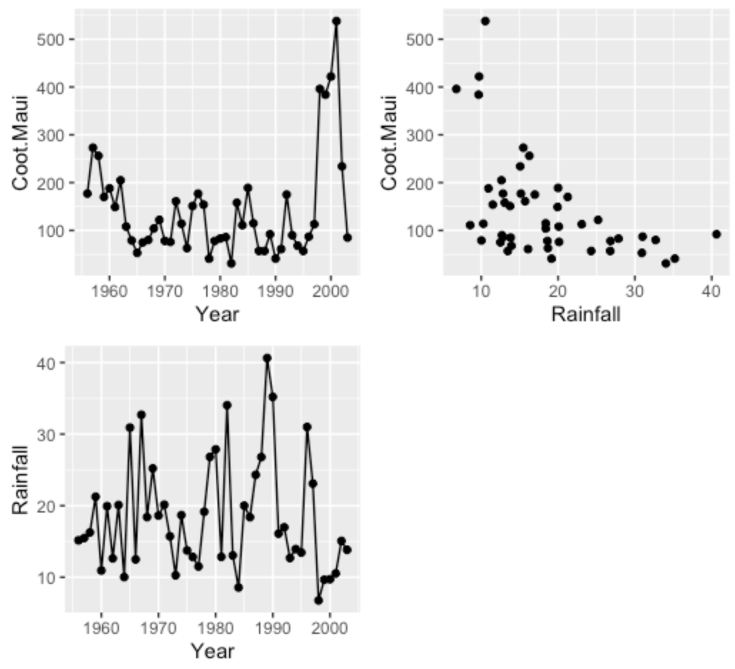
Lecture 24. Generalized Least Squares

Temporal autocorrelation

We've already thought a bit about data that has spatial or temporal correlation, but now we're going to address it more directly, starting with time. The analysis of time series is a large and complex field, and there are many different approaches used for different tasks. Autoregressive models can be useful for prediction, but maybe less for explaining patterns; fourier decomposition and wavelets are useful for detecting periodic fluctuation; mechanistic models of temporal dynamics can be fit to ecological time series using state space models; etc. We don't have time to dig into these matters in this course, so I will focus on how time series can be analyzed with the kind regression models we've been learning.

This will be a simple example to illustrate some basic principles. Here is time series from Maui of counts of the Hawaiian Coot:

```
p1 = ggplot(birds, aes(Year, Coot.Maui)) + geom_point() + geom_line()
p2 = ggplot(birds, aes(Year, Rainfall)) + geom_point() + geom_line()
p3 = ggplot(birds, aes(Rainfall, Coot.Maui)) + geom_point()
plot_grid(p1, p3, p2, ncol = 2)
```



On the top left is coots over time. There is clearly fluctuation, and it looks like samples in adjacent years are similar. The bottom left plot shows rainfall over time, which also fluctuates considerably. The third plot compares coot abundance to rainfall, and it look like there is a negative relationship.

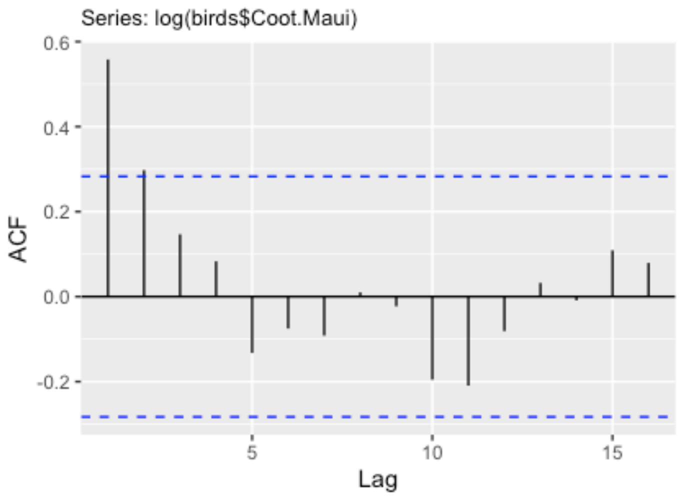
Our simple task here is to make a model for the relationship between rainfall and coot abundance. Although the data are counts, I'm going to start with by assuming log(coots) are normally distributed, in order to illustrate how to analyze time series on normally distributed data. Because #coots never gets close to zero, the lognormal approximation will probably work well.

Before we fit a model to the data, let's explore the temporal structure of the data. The most common way to do this is by plotting the autocorrelation function. The autocorrelation function is just the correlation between the time series and itself, at different lags. We can write it like this:

$$r_k = \frac{\sum_{t=1}^{N-k} (Y_t - \bar{Y})(Y_{t+k} - \bar{Y})}{\sum_{t=1}^{N} (Y_t - \bar{Y})^2}$$

r_k is the correlation at lag k, N is the length of the time series, Y_t is the observation at time t. The top of this fraction is just the equation for the covariance between observations at time t and observations at time $t+k$; the bottom is the sum of squares of the data, which makes the correlation range between 0 and 1. We can plot it with ggAcf() in the package 'forecast':

```
ggAcf(log(birds$Coot.Maui))
```



This function plots the estimated autocorrelation at each lag, plus a dotted line that approximates the threshold at which the correlation is significantly different from zero. The correlation at lag 0 is 1, which is plotted for reference. For the coots the correlation at lag 1 is about 0.6, and at lag 2 is about 0.3. So, observations within 1 or 2 years of each other are correlated, and after that there is not much signal of correlation. We can think of various reasons why observations within 1-2 years of each other might be similar. Coots live for more than one year, so if there is a good year for coot reproduction/survival, the population size could remain high for multiple years. If coots move around, but we sample at one location, then if the location tends to be good/bad for multiple years in a row, that could produce a similar signal.

Autoregressive and moving average models

To incorporate this kind of temporal structure into a regression model, we need a simple model for time series dynamics. In particular, we need a model that relates the observations in a time series to each other, to try and characterize the autocorrelation pattern. One very simple model is called an *autoregressive* model. An autoregressive model of order 1 looks like this:

$$Y_t = c + \varphi Y_{t-1} + \varepsilon_t$$

Here c is a constant that determines the mean of the Y's, φ is effect of Y_{t-1} on Y_t, and ε_i is random, independently drawn noise. The idea here is that the value of Y at time t depends on its previous value to some extent, quantified by φ, and otherwise it is randomly distributed according to the epsilons. For our applications φ will be assumed to be between -1 and 1, and thus models the lag-1 autocorrelation. This is a first-order autoregressive model because what happens only depends on the previous time step. The number of lagged observations can be any number; here's what a second-order model looks like:

$$Y_t = c + \varphi_1 Y_{t-1} + \varphi_2 Y_{t-2} + \varepsilon_t$$

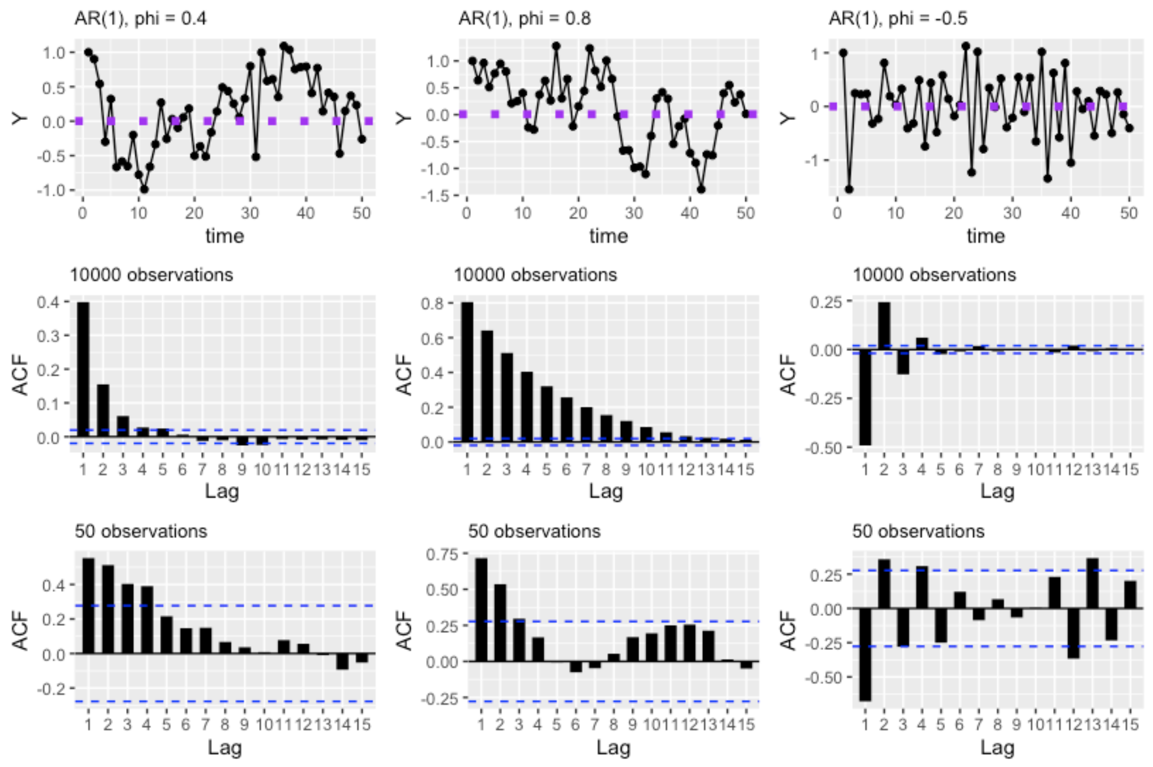
Now what happens at time t depends on both previous time steps. These models are typically abbreviated as AR(1) and AR(2), etc.

To visualize what these models mean, we can simulate.

```
#AR1
phis = c(0.4, 0.8, -0.5)
tseries = acf.long = acf.short = vector(mode = 'list', length = 3)
par(mfrow = c(3,3), mar = c(4,4,2,1))
for (j in 1:3) {
Y = 1
tmax = 10000
SD = 0.5
phi = phis[j]
for (t in 2:tmax) {
  Y[t] = phi*Y[t-1] + rnorm(1, mean = 0, sd = SD)
}
tsteps = 1:tmax
simseries = data.frame(tsteps, Y)
tseries[[j]] = ggplot(subset(simseries, tsteps <= 50), aes(tsteps, Y))
+ geom_point() + geom_line() + labs(title = paste('AR(1), phi =', phi))
 + xlab('time') + ylab('Y') + geom_hline(yintercept = mean(Y), col = 'p
urple', lty = 3, size = 2)
acf.long[[j]] = plot(ggAcf(Y, lwd = 3, lag.max = 15) + labs(title = '10
000 observations'))
acf.short[[j]] = plot(ggAcf(Y[1:50], lwd = 3, lag.max = 15) + labs(titl
e = '50 observations'))
}

plot_grid(plotlist = c(tseries, acf.long, acf.short), nrow = 3)
```

Here I've simulated three examples of an AR1 model, with phi = 0.4, 0.8, or -0.5. I've assumed that c = 0, so the overall mean is zero. And I've assumed that the residual noise has SD = 0.5. I simulated a time series of 10,000 steps (e.g years), but I've plotted the first 50, because that's the kind of data we have for populations.



The column on the left shows the time series. The column in the middle is the autocorrelation plot for the whole 10,000 steps. The column on the right is the autocorrelation plot for the first 50 steps. The middle column shows you what the expected pattern should look like, and the plot on the right is an example of what it will look like with limited data.

With phi = 0.4, adjacent years are often similar when looking at the time series, and the autocorrelation plot shows that the signal decays exponentially over time. We can understand why this is, if we start at the beginning and substitute values:

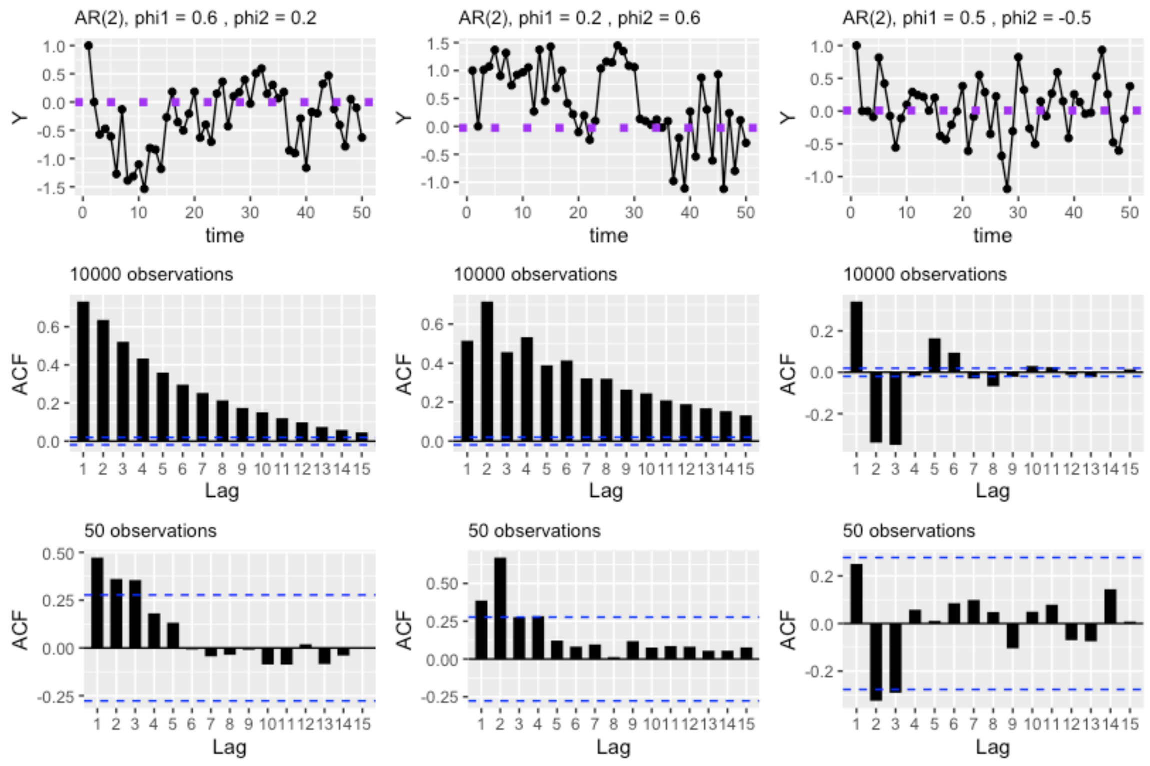
$$Y_2 = \varphi Y_1 + \varepsilon_2$$
$$Y_3 = \varphi Y_2 + \varepsilon_3 = \varphi(\varphi Y_1 + \varepsilon_2) + \varepsilon_3 = \varphi^2 Y_1 + \varphi\varepsilon_2 + \varepsilon_3$$

If phi = 0.4, then adjacent values should have a correlation of 0.4, and values separated by 2 years will have a correlation of 0.4*0.4 = 0.16, and values separated by 3 years will have a correlation of 0.4^3 = 0.064, etc.

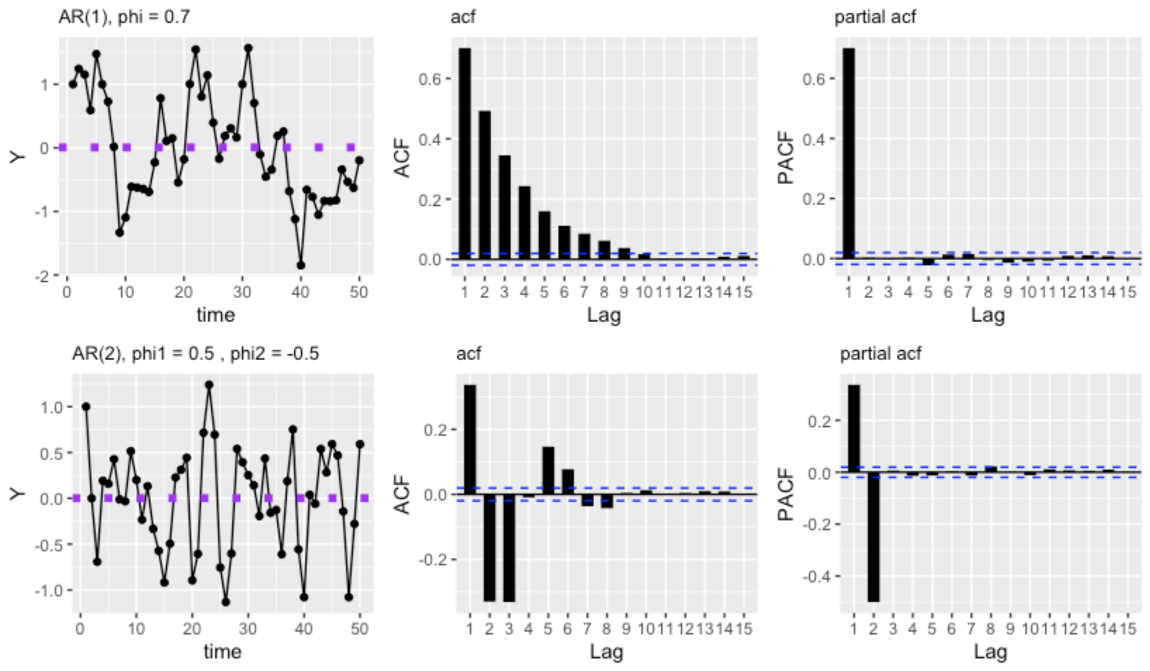
With phi = 0.8, the time series will have longer series of fluctuations away from zero, because adjacent values are highly correlated, and so the time series has a long 'memory' of previous events. We can imagine that yearly samples of a very long-live species (e.g. redwood trees) could show a pattern like this, while yearly samples of a species with a shorter lifespan might be more like phi = 0.4.

Finally, when phi = -0.5, we get a different kind of pattern, because the negative phi means the time series tends to oscillate back and forth around the mean. This also causes there to be a (weaker) positive correlation at lag-2, because values separate by two years will tend to have a correlation of (-0.5)*(-0.5) = 0.25. This kind of pattern might arise due to predatory-prey population cycles for short-lived species.

When we go from AR(1) to AR(2), the temporal patterns can get more complex, because now we're essentially superimposing two autocorrelation patterns on top of each other. Here are some examples:



To help distinguish the *order* of the autocorrelation process, i.e. how many previous observations affect the current observations, we can plot a *partial autocorrelation function*. This calculates the correlation at lag *k*, *while substracting any contribution from shorter lags*. The partial autocorrelation removes the pattern of exponential decay, because the long tail of autocorrelation is just an echo of autocorrelation at shorter lags. This can be done with ggAcf(mydata, type = “partial”).



Here I’ve plotted an AR(1) process on the top, and an AR(2) process on the bottom. The plots on the right show the partial acf. The AR(1) only shows autocorrelation at lag 1, and the AR(2) only shows autocorrelation at the first two lags. So this kind of plot can be useful for determining what kind of AR model is appropriate for the data.

A second kind of model that is often combined with the autoregressive model is called a *moving average* model. It looks like this:

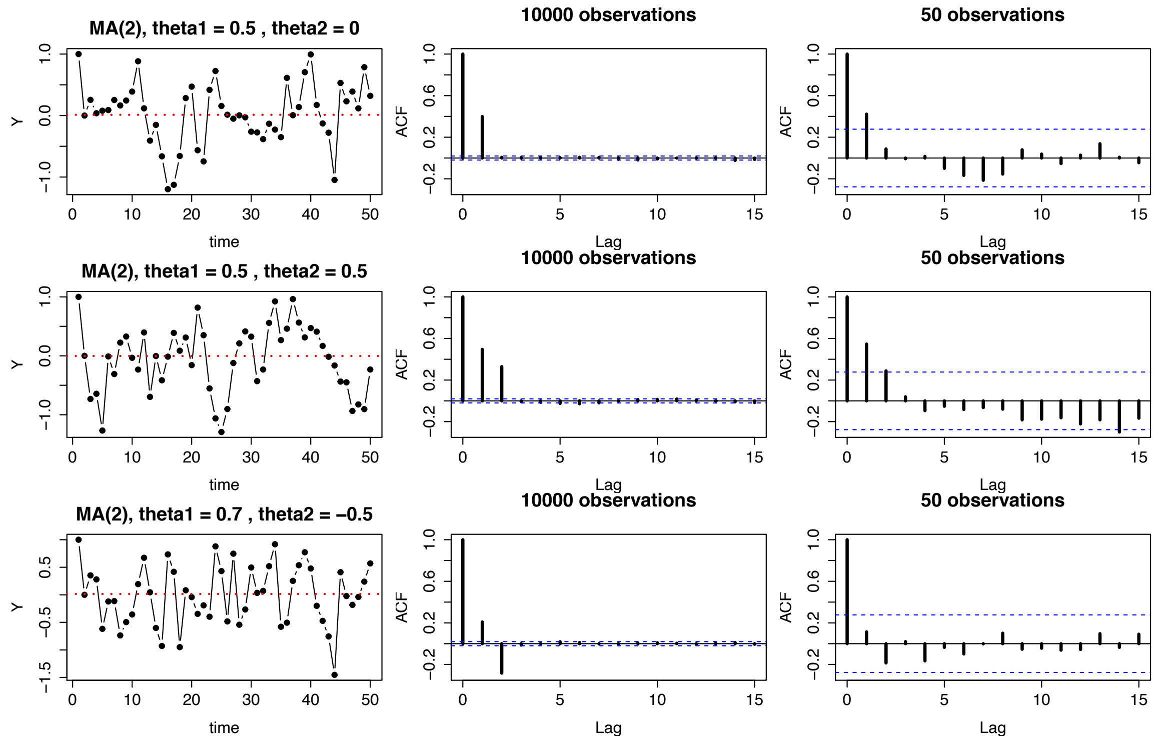
$$Y_t = \mu + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q}$$

where μ is a constant for the mean, and the θ_i are parameters that determine how much the current observation Y_t is affected by the random ‘shocks’ from prior time steps. With *q* parameters, this is called an order-q moving average model, or MA(q). The autoregressive and moving average models are superficially similar, but they have somewhat different effects. The autoregressive model says “the observation right now depends on the values of previous observations”. This is the kind of thing that would happen in a population because the current generation was created by the previous generation. A moving average model says that the process at time t is affected by random influences from multiple previous years. This is a little harder to relate to population dynamics (our example), but this kind of model is good at representing short-term autocorrelation that does not have a long-term “memory”. Let’s simulate from a moving average model that is affected by the two previous years, i.e. MA(2).

```
#MA
theta1s = c(0.5, 0.5, 0.7)
theta2s = c(0, 0.5, -0.5)
par(mfrow = c(3,3), mar = c(4,4,3,1))
for (j in 1:3) {
theta1 = theta1s[j]
theta2 = theta2s[j]
tmax = 10000
SD = 0.5

eps = rnorm(1, mean = 0, sd = SD)
eps[2] = rnorm(1, mean = 0, sd = SD)
for (t in 3:tmax) {
  eps[t] = rnorm(1, mean = 0, sd = SD)
  Y[t] = eps[t] + theta1*eps[t-1] + theta2*eps[t-2]
}

plot(c(1:50), Y[1:50], type = 'b', pch = 19, main = paste('MA(2), theta
1 =', theta1, ', theta2 =', theta2), xlab = 'time', ylab = 'Y')
abline(h = mean(Y), col = 'red', lty = 3, lwd = 2)
acf(Y, lwd = 3, lag.max = 15, ylim = c(-0.3, 1), main = '10000 observat
ions')
acf(Y[1:50], lwd = 3, lag.max = 15, ylim = c(-0.3, 1), main = '50 obser
vations')
}
```



The first row is actually an MA(1) model, because theta2 = 0. The second row is an MA(2) model with two positive coefficients, and the third row is an MA(2) model with a positive and negative coefficient. When we make the acf plot using 10,000 samples, we can see that there is zero correlation beyond lag-1 for MA(1), and zero correlation beyond lag-2 for the MA(2) models. That's because in the MA models the current observation does not depend on the previous observations *per se*, rather the current observation depends on residual 'shocks' received by the previous observations. So this lack of an exponential decay can be used as an indication of an MA process instead of an AR process in the data. But with limited data it is often difficult to make this distinction, and we rely on comparing the fits of alternative models with AIC instead.

In practice, for the regression models we're considering, people tend to start with AR models, and add in MA coefficients if it looks like there is extra complexity in the time series that needs to be accounted for. So this is basically a nonparametric-esque approach, because interpreting the parameters is not very helpful.

Generalized least squares - theory

To get back to the data, we're going to use an autoregressive model for the coots on Maui. In fact we want to 1) model the effect of rainfall, while 2) also modeling any additional temporal structure in the data. So we're actually going to model the *temporal correlation of the the residuals*, while estimating the effects of any predictors (in this case rainfall). To do this we can use a technique called *generalized least squares*. This is really confusing terminology, because 'generalized linear models' are linear-ish model for *non-normal* data, while 'generalized least squares' is a linear model for *normally* distributed data that includes a residual correlation structure (the residuals are the 'generalized' part here). Apparently this is the fault of the guy who coined the GLM name.

Generalized least squares is linear modeling with an added component for how the residual variation has some extra structure (temporal, spatial, etc.). We've already used random effects to model certain kinds of structured variability, but GLS can handle some other kinds of situations. To start with, we can't just think of each observation like this:

$$Y_t = \beta_0 + \beta_1 X_t + \varepsilon_t$$

where each observation has some predicted value ($\beta_0 + \beta_1 X_t$) and some independently drawn residual noise ε_t. Instead, we're going to think about all the residuals in the whole dataset in a single vector, like this:

$$\begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \\ \varepsilon_4 \end{bmatrix} = \vec{\varepsilon} \sim MultivariateNormal(\Sigma)$$

where the residual vector $\vec{\varepsilon}$ is drawn from a multivariate normal distribution with variance-covariance matrix Σ. Here my residual vector only has four elements to save space, but imagine that it includes all observations in the dataset. For a time series, these would be residuals for year 1, 2, 3, and 4. By treating the residuals as a vector drawn from a multivariate distribution, *we can explicitly model any correlation among the residuals*. This allows us to 1) learn things about the residual correlation structure, and 2) allow a regression model of a time series (or spatial dataset, etc.) to have assumptions that match the nature of the data.

In this context, if we want to model the residuals as an AR(1) process, we just specify a variance-covariance matrix for the residuals that looks like this:

$$\sigma^2 \begin{bmatrix} 1 & \varphi & \varphi^2 & \varphi^3 \\ \varphi & 1 & \varphi & \varphi^2 \\ \varphi^2 & \varphi & 1 & \varphi \\ \varphi^3 & \varphi^2 & \varphi & 1 \end{bmatrix}$$

Here the residual variance, σ^2, is pulled outside the matrix, so that what's inside the matrix is just the expected correlation among the four residuals. The correlation between the residual at time 1 and the residual at time 2 is φ, because we're assuming an AR(1) model for the residuals. The correlation between the residual at time 1 and the residual at time 3 is φ^2, because they are separated by two years;

and the correlation between the residual at time 1 and the residual at time 4 is φ^3. So if $\varphi = 0.5$, then $\varphi^2 = 0.25$ and $\varphi^3 = 0.125$. This means samples that are one year apart will have a correlation of 0.5 (on average), and samples two years apart will have a correlation of 0.25 on average, and samples three years apart will have a correlation of 0.125 on average. That's how an autoregressive model works. This matrix is specifying that when the residual vector is drawn, the residuals should have this correlation structure (on average).

Also not that the overall variance in the residuals, sigma, is assumed to be a constant. *This means we're assuming the variability in the data does not change over time*. This is usually stated more generally by saying that we assume the residuals have a *stationary* distribution, which just means that the distribution from which the residuals are drawn does not change over time. Depending on the data, this may or may not be a problematic assumption.

Generalized least squares for time series

We can easily specify an AR(1) model for the residuals:

```
mod.nopredictors = gls(log(Coot.Maui) ~ 1, data = birds, correlation =
corAR1(form =~ Year))
```

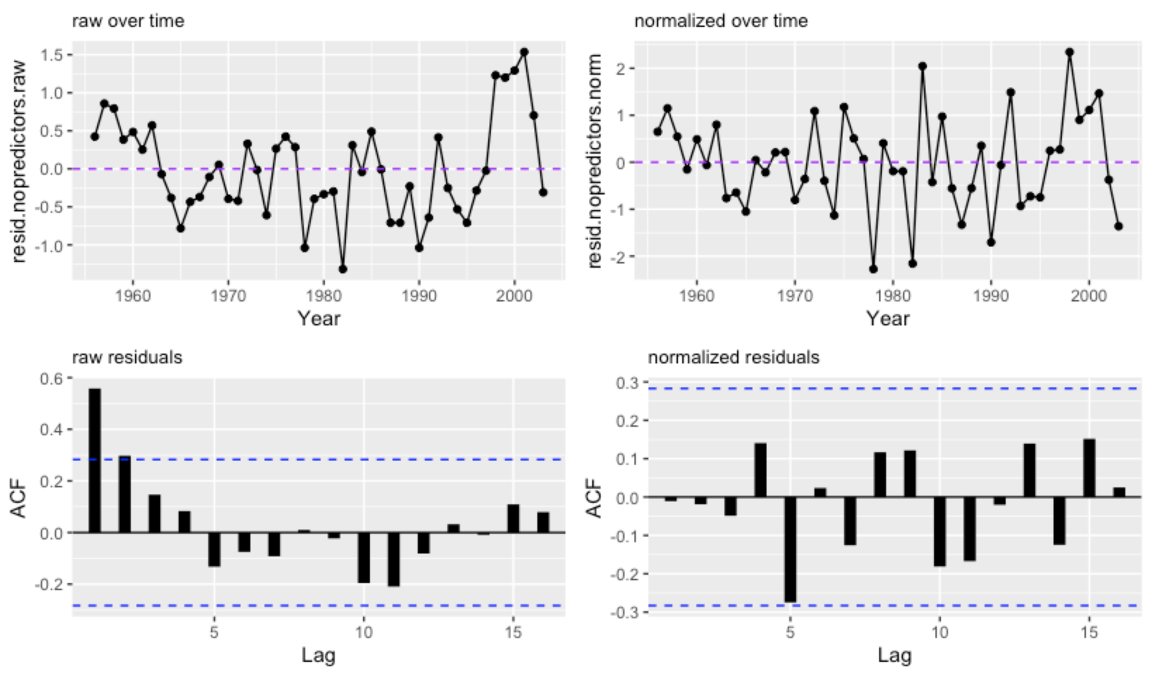
I haven't included rainfall yet, so this model will fit an intercept (the mean of the data), and it will model the correlation in the residuals around that mean. The gls() function does generalized least squares. The specification is the same as lm(), plus an extra 'correlation' argument, 'corAR1(form = ~ Year))'. The correlation argument can use a wide variety of correlation structures, which you can see by looking at ?corStruct. We will get to other options later. For now we're going to try an AR(1) model for the residuals which uses the function corAR1(). Inside this function we tell it how to order the residuals in time, with form = ~ Year. If we had time series from multiple sites, and wanted to model autocorrelation *within each site separately*, then we could modify the formula like this: form = ~ Year | Site. Let's look at the fit:

```
## Generalized least squares fit by REML
##   Model: log(Coot.Maui) ~ 1
##   Data: birds
##    AIC   BIC logLik
##   82.3 87.85 -38.15
##
## Correlation Structure: AR(1)
##  Formula: ~Year
##  Parameter estimate(s):
##   Phi
## 0.587
##
## Coefficients:
```

```
##             Value Std.Error t-value p-value
## (Intercept) 4.752      0.18    26.4       0
##
## Standardized residuals:
##       Min       Q1      Med       Q3      Max
## -2.01305 -0.61361 -0.08572  0.63491  2.34595
##
## Residual standard error: 0.6547
## Degrees of freedom: 48 total; 47 residual
```

We only have two parameters in this model, the intercept and phi. Phi is estimated to be 0.59, which is fairly strong correlation between successive years. To visualize what the AR(1) structure is doing, we can compare the raw residuals, i.e. log(Coot.Maui) – (Intercept), to the 'normalized' residuals that account for the AR(1) structure and display the 'left over' variation.

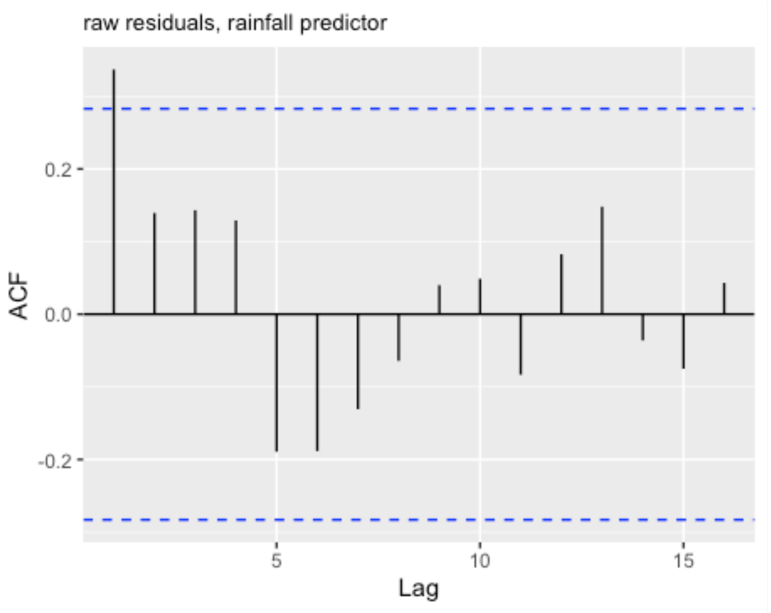
```
birds$resid.nopredictors.raw = resid(mod.nopredictors)
birds$resid.nopredictors.norm = resid(mod.nopredictors, type = "normali
zed")
p1 = ggplot(birds, aes(Year, resid.nopredictors.raw)) + geom_point() +
geom_line() + labs(title = 'raw over time') + geom_hline(yintercept =
0, col = 'purple', lty = 2)
p2 = ggplot(birds, aes(Year, resid.nopredictors.norm)) + geom_point() +
 geom_line() + labs(title = 'normalized over time') + geom_hline(yinter
cept = 0, col = 'purple', lty = 2)
p3 = ggAcf(birds$resid.nopredictors.raw, lwd = 3) + labs(title = 'raw r
esiduals')
p4 = ggAcf(birds$resid.nopredictors.norm, lwd = 3) + labs(title = 'norm
alized residuals')
plot_grid(plotlist = list(p1, p2, p3, p4), nrow = 2)
```



Here I've plotted both raw and normalized residuals, as a time series and using acf(). The raw residuals show the same pattern as when we did acf() on the raw data, which is how it should be, because this model has no predictors. The normalized residuals show no significant autocorrelation, especially not at lags 1 and 2, which were the problematic ones. Likewise, the time series looks more like it just bounces around randomly, compared to the raw residuals. It looks like the variance may increase a little over time, but it's not so bad that I will worry about it. From this we can conclude that the AR(1) model is sufficient to account for the temporal structure of this data, in the sense of making a regression model with assumptions consistent with the data.

Now let's finally get around to using rainfall to predict the abundance of coots. There is an important thing to remember when considering non-independence in a time series (or any other regression context). We're assuming the data is independently distributed, *once the predictors are accounted for*. Because rainfall fluctuates over time, it could potentially explain the fluctuations in coot abundance, leaving residual variation that is *not* autocorrelated. We can inspect this by just fitting a rainfall model with lm():

```
mod = lm(log(Coot.Maui) ~ Rainfall, data = birds)
birds$resid.rainpredictor.raw = resid(mod)
ggAcf(birds$resid.rainpredictor.raw) + labs(title = 'raw residuals, rai
nfall predictor')
```

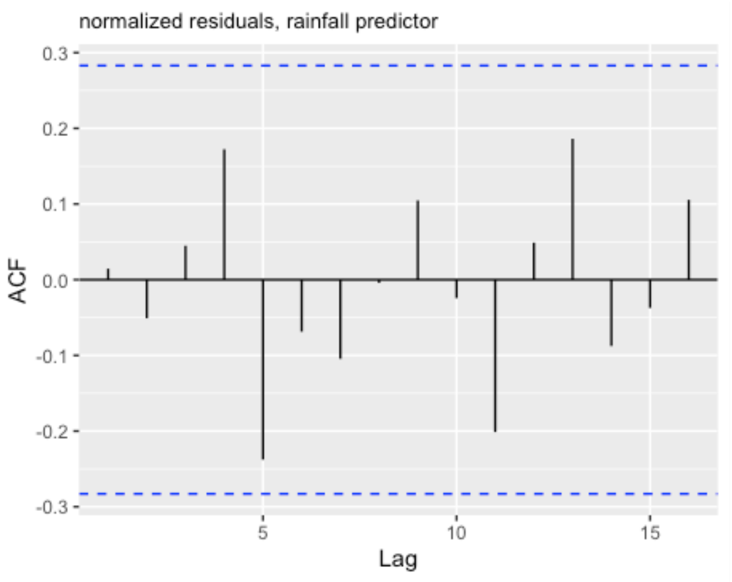


This model still has some correlation at lag-1, which we should deal with, but it's noteworthy that the correlation is reduced compared to the raw data (from almost

0.6 to about 0.35). This suggests that rainfall is part of the reason why the data are autocorrelated: consecutive years tend to have similar rainfall.

Finally, I'll make a model with rainfall and the AR(1) residuals:

```
mod1 = gls(log(Coot.Maui) ~ Rainfall, data = birds, correlation = corAR
1(form =~ Year), method = "ML")
ggAcf(resid(mod1, type = 'normalized')) + labs(title = 'normalized resi
duals, rainfall predictor')
```



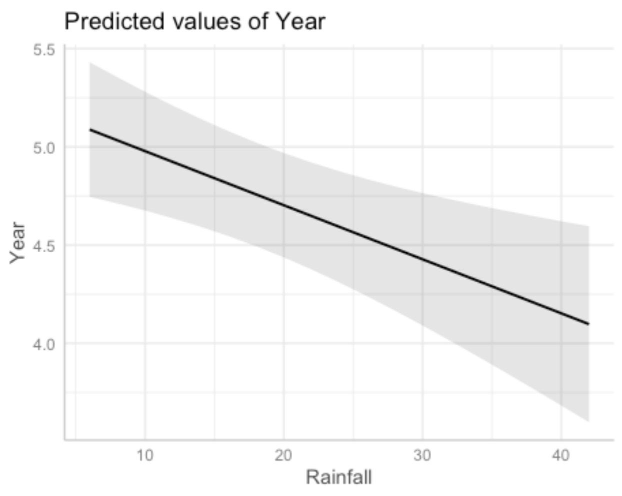
Now there's not evidence of autocorrelation, so this is a satisfactory model. Let's see what it looks like:

```
summary(mod1)

## Generalized least squares fit by maximum likelihood
##   Model: log(Coot.Maui) ~ Rainfall
##   Data: birds
##   AIC    BIC logLik
##    74 81.48    -33
##
## Correlation Structure: AR(1)
##  Formula: ~Year
##  Parameter estimate(s):
##    Phi
## 0.4696
##
## Coefficients:
##              Value Std.Error t-value p-value
## (Intercept)  5.253   0.20829  25.222  0.0000
## Rainfall    -0.028   0.00887  -3.105  0.0033
##
##  Correlation:
```

```
##          (Intr)
## Rainfall -0.777
##
## Standardized residuals:
##      Min       Q1      Med       Q3      Max
## -1.86302 -0.80130  0.02423  0.64319  2.43593
##
## Residual standard error: 0.5436
## Degrees of freedom: 48 total; 46 residual
```

The effect of rainfall is highly significant, and the residuals have a lag-1 correlation of 0.47.



The effect size for rainfall is reasonably large. Over the range of observed rainfall, coots are expected to vary from exp(5) = 148 to exp(4.2) = 67.

The last thing I'll note for this example is that we could also consider more complex autocorrelation structures. They do not appear to be necessary here, but to introduce you to them, we can fit AR(p) models +/- MA(q) models using corARMA, i.e. autoregressive moving average models. So the syntax for an AR(2) model would look like this:

```
mod.gls2 = gls(log(Stilt.Maui) ~ Rainfall + Year, data = birds, correlation =
corARMA(form =~ Year, p = 2, q = 0))
```

And the syntax for an ARMA(2,2) model, which is AR(2) + MA(2), would look like this:

```
mod.gls4 = gls(log(Stilt.Maui) ~ Rainfall + Year, data = birds, correlation =
corARMA(form =~ Year, p = 2, q = 2))
```

We would consider these more complex correlation structures if the AR(1) didn't appear sufficient. And we could compare among these options using AIC:

```
AICc(mod.gls)

## [1] 23.94

AICc(mod.gls2)

## [1] 29.42

AICc(mod.gls4)

## [1] 27.69
```

The AR(1) model has a convincingly lower AIC, so the added complexity is not helping. We can also compare to a model that does not account for autocorrelation:

```
mod = gls(log(Coot.Maui) ~ Rainfall, data = birds
```

```
AICc(mod)

## [1] 33.89
```

This model is worse than any of the other options, indicating that accounting for residual autocorrelation yields a much better fitting model. Note that for this comparison I fit the no-correlation model with gls(). In general you should only compare AIC for models fit using the same function, because functions can differ in some details of how the log-likelihood is calculated.