Maximum Likelihood Lab 4: Population Dynamics

< your name here >

BIOHOPK 143H - Winter 2021

```
knitr::opts_chunk$set(fig.height = 4, fig.width = 6, message = FALSE, warning = FALSE)

### load the tidyverse packages
if (!require("tidyverse")) install.packages("tidyverse"); library(tidyverse)
```

```
## Loading required package: tidyverse
```

```
## Warning: package 'tidyverse' was built under R version 4.1.2
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.4     v dplyr   1.0.7
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   2.0.1     v forcats 0.5.1
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
###  mvtnorm package for building autocorrelative model
if (!require("mvtnorm")) install.packages("mvtnorm"); library(mvtnorm)
```

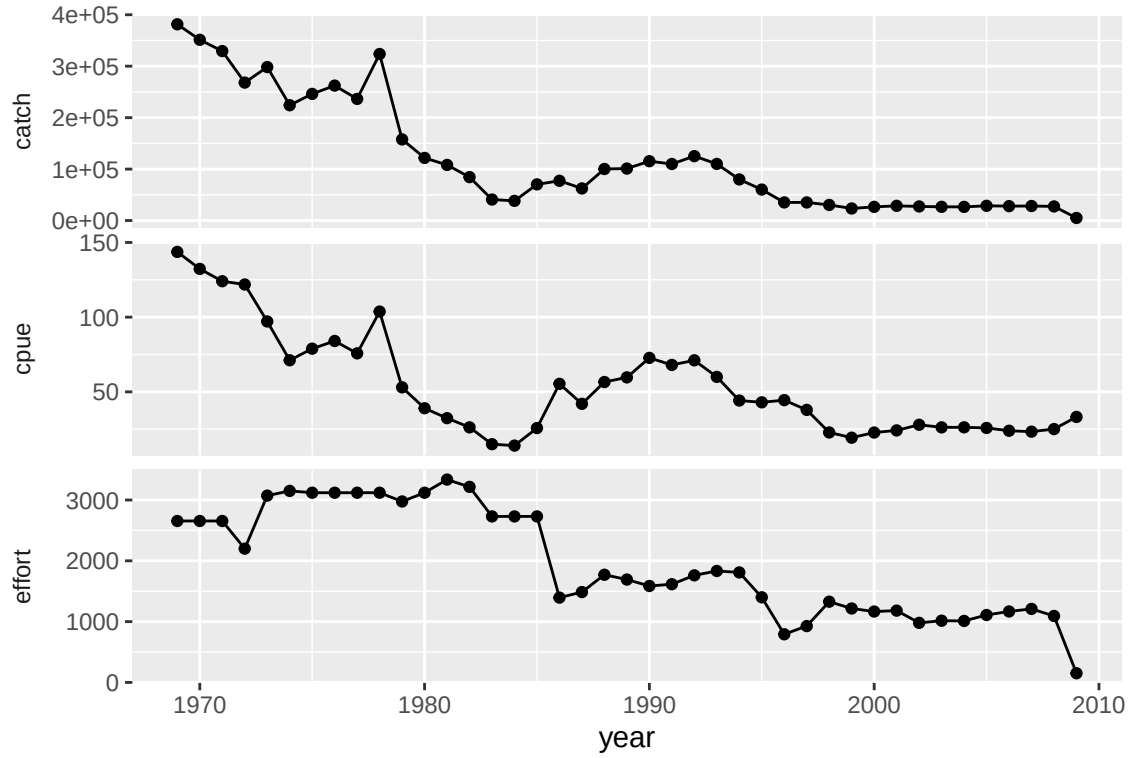
```
## Loading required package: mvtnorm
```

Load and plot data

These data contain fisheries catch (in tonnes) and effort for pink and green abalone on Isla Natividad from 1969 to 2009 - catch data were not distinguished at the species level until 2002. The `cpue` column is just the catch divided by the effort. For the purposes of this exercise, we'll assume that pink and green abalone have the same demographic parameters - i.e., we'll be modeling this data as if it comes from a single population.

```
natividad <- read.csv("isla_natividad_abalone.csv")

natividad %>%
  pivot_longer(-year, names_to = "var") %>%
  ggplot(aes(year, value)) +
  geom_line() +
  geom_point() +
  facet_wrap(~var, scales = "free_y", strip.position = "left", nrow = 3) +
  theme(strip.placement = "outside",
        strip.background = element_blank(),
        axis.title.y = element_blank())
```



The Beverton-Holt model

In the Beverton-Holt model, which is discrete, n_{t+1} given n_t, the growth rate λ, and the density-dependent parameter α is:

$$n_{t+1} = \frac{\lambda n_t}{1 + \alpha n_t}$$

The carrying capacity for this model is given by:

$$K = \frac{\lambda - 1}{\alpha}$$

The model has the general solution for the population size n_t at time t:

$$n_t = \frac{K n_0}{n_0 + (K - n_0)\lambda^{-t}}$$

We don't have n at *any* time step, but we want to estimate it - to do so, we have to estimate λ, α, and n_0. But what data do we have to estimate these parameters with? Well, we can infer these parameters through the relationship between the observed catch and effort. In a continuous time model, the catch is simply the population size multiplied by the fisheries effort and catchability q (we have to estimate this) at a given time t:

$$\text{Catch}_t = n_t \times q \times \text{Effort}_t$$

In a discrete-time model, we have:

$$\text{Catch}_t = n_t \times \left(1 - e^{-q \times \text{Effort}_t}\right)$$

So, the three parameters we have to estimate for the deterministic part of our maximum-likelihood model are the growth rate λ, the density-dependent paramter α, and the catchability q, and we'll do so by comparing the observed catch (which we'll denote obs. Catch_t) to the estimated catch (which we'll denote with μCatch_t).

There's one thing we're missing here though - the initial population size n_0. We can estimate this as a separate parameter, or we can use our catchability estimate to derive one using the relationship between catch, effort, and the population size that we've defined above. Reorganizing the equation above, we have:

$$n_0 = \frac{\mu\text{Catch}_0}{1 - e^{-q \times \text{Effort}_0}}$$

The caveat here is that we need an estimate of n_0 to estimate the true catch at any given time μCatch_t, so for the first time step $t = 0$, we'll have to assume that the true and observed catch are the same:

$$n_0 = \frac{\text{obs. Catch}_0}{1 - e^{-q \times \text{Effort}_0}}$$

Maximum Likelihood Estimation

By now, we're pretty familiar with the process of creating a likelihood function for our model. Not much is different here, but since this model is a discrete dynamic model, we're going to loop through time to obtain our estimates.

In this model, our independent variable is fishery effort, and our dependent variable is fishery catch. Because the variance in catch doesn't increase much as the catch increases in size, and because the catch never gets close to zero (the minimum possible value), a Normal distribution is probably adequate for this data. If it turns out that a Normal distribution isn't the best choice, that'll show in our **residuals plots** - just like we'd do for an ordinary linear regression.

So, our model is:

$$\begin{aligned}
\text{obs. Catch}_t &\sim \text{Normal}(\mu\text{Catch}_t, \sigma^2) \\
\mu\text{Catch}_t &= n_t \times \left(1 - e^{-q \times \text{Effort}_t}\right) \\
n_t &= \frac{\lambda n_{t-1} \times e^{-q \times \text{Effort}_{t-1}}}{1 + \alpha\left(n_{t-1} \times e^{-q \times \text{Effort}_{t-1}}\right)} \; ; \; t \in (1, 2, \ldots, t_{\max}) \\
n_0 &= \frac{\text{obs. Catch}_0}{1 - e^{-q \times \text{Effort}_0}}
\end{aligned}$$

Likelihood function

Here's a function that returns a negative log-likelihood given the parameters (`lambda`, `alpha`, `q`, and `sigma`) and some data (`t`, the time steps, `obs_catch`, the observed catch at time t, and `obs_effort`, the observed effort at time t).

Here, we've raised `alpha` and `q` to the power of 10 inside the function, and multiplied `sigma` by 10,000 - that way, the coefficients are all on similar scales to make things easier for the optimizer.

Take a minute to make sure you understand the code:

```
beverton_holt_nll <- function(par, t, obs_catch, obs_effort) {

  lambda <- par[1]; alpha <- 10^par[2]; q <- 10^par[3]; sigma <- par[4]*1e4
```

```
  tmax <- length(t)

  ## empty vectors to store simulated catch
  mu_catch <- rep(0, tmax)
  mu_stock <- rep(0, tmax)

  ## estimate for stock size at time t = 1
  mu_stock[1] <- obs_catch[1]/(1 - exp(-q * obs_effort[1]))

  ## loop through time, store simulated catch and stock
  for (ti in 1:tmax) {
    if(ti < tmax) {
      mu_stock[ti + 1] <- (lambda * mu_stock[ti] * exp(-q * obs_effort[ti])) /
        (1 + alpha * (mu_stock[ti] * exp(-q * obs_effort[ti])))
    }
    mu_catch[ti] <- mu_stock[ti] * (1 - exp(-q * obs_effort[ti]))
  }

  if (sigma >= 0) {
    nll <- -sum(dnorm(obs_catch, mean = mu_catch, sd = sigma, log = TRUE))
    return(nll)
  }  else {
    return(5000)
  }

}
```

Maximum Likelihood Estimates

Not much is new here. I won't go through finding initial values for this model, since we'll do that next week when we fit a Bayesian Beverton-Holt model to a different dataset, but one thing that's new here is the `control` argument - we've increased the maximum number of iterations (`maxit`).

Here are the maximum likelihood estimates:

```
bh_mle <- optim(
  par = c(lambda = 1.2, alpha = log(0.0025), q = log(0.049), sigma = 3.9),
  fn = beverton_holt_nll,
  t = natividad$year,
  obs_catch = natividad$catch,
  obs_effort = natividad$effort,
  control = list(maxit = 1000),
  hessian = TRUE
)

bh_mle
```

```
## $par
##    lambda     alpha         q     sigma
##  1.090678 -7.694453 -4.303808  3.900584
##
## $value
## [1] 491.6009
##
## $counts
```

```
## function gradient
##      615       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
##
## $hessian
##               lambda         alpha             q         sigma
## lambda  2.884522e+04 -2.063770e+03 -7.016462e+03 -0.0442179768
## alpha  -2.063770e+03  1.615596e+02  5.105124e+02  0.0006655583
## q      -7.016462e+03  5.105124e+02  1.753809e+03 -0.0069437789
## sigma  -4.421798e-02  6.655583e-04 -6.943779e-03  5.3873423838
```

and their standard errors:

```
var_cov <- solve(bh_mle$hessian)
se <- sqrt(diag(var_cov))
se
```

```
##     lambda      alpha          q      sigma
## 0.03677964 0.28436654 0.15452469 0.43083699
```

Catch & CPUE Estimates

Now, we can plot the estimated stock and catch over time, using the parameters estimated from our model. First, though, we need to define a function to return our estimated stock and catch, given the parameter estimates:

```
beverton_holt <- function(lambda, alpha, q, n0, t, effort) {

  tmax <- length(t)
  stock <- rep(0, tmax); stock[1] <- n0
  catch <- rep(0, tmax)

  for (ti in 1:tmax) {
    if(ti < tmax) {
      stock[ti + 1] <- (lambda * stock[ti] * exp(-q * effort[ti])) /
        (1 + alpha * (stock[ti] * exp(-q * effort[ti])))
    }
    catch[ti] <- stock[ti] * (1 - exp(-q * effort[ti]))
  }

  data.frame(t = t, stock = stock, catch = catch)
}
```

Applying this function to our data and parameter estimates:

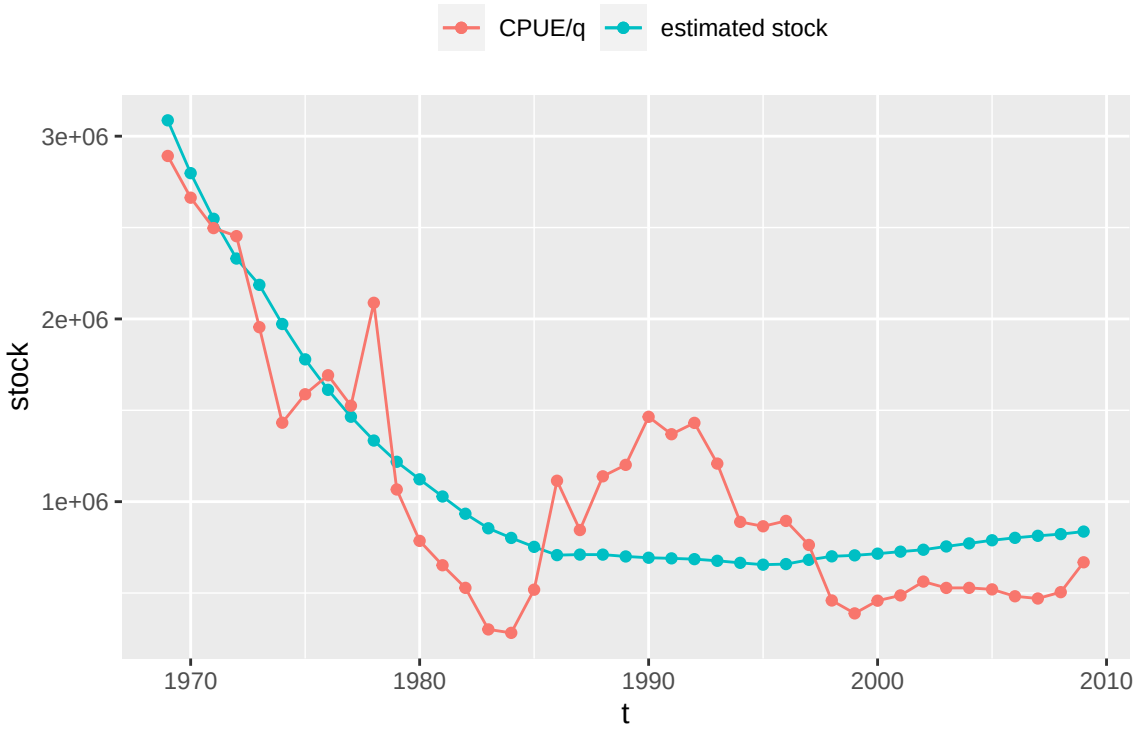
```
n0 <- as.numeric(natividad$catch[1]/(1 - exp(-10^bh_mle$par[["q"]] * natividad$effort[1])))

bh_fit <- beverton_holt(
  lambda = bh_mle$par[["lambda"]],
  alpha = 10^bh_mle$par[["alpha"]],
  q = 10^bh_mle$par[["q"]],
  n0 = n0,
```

```
  t = natividad$year,
  effort = natividad$effort
)
```

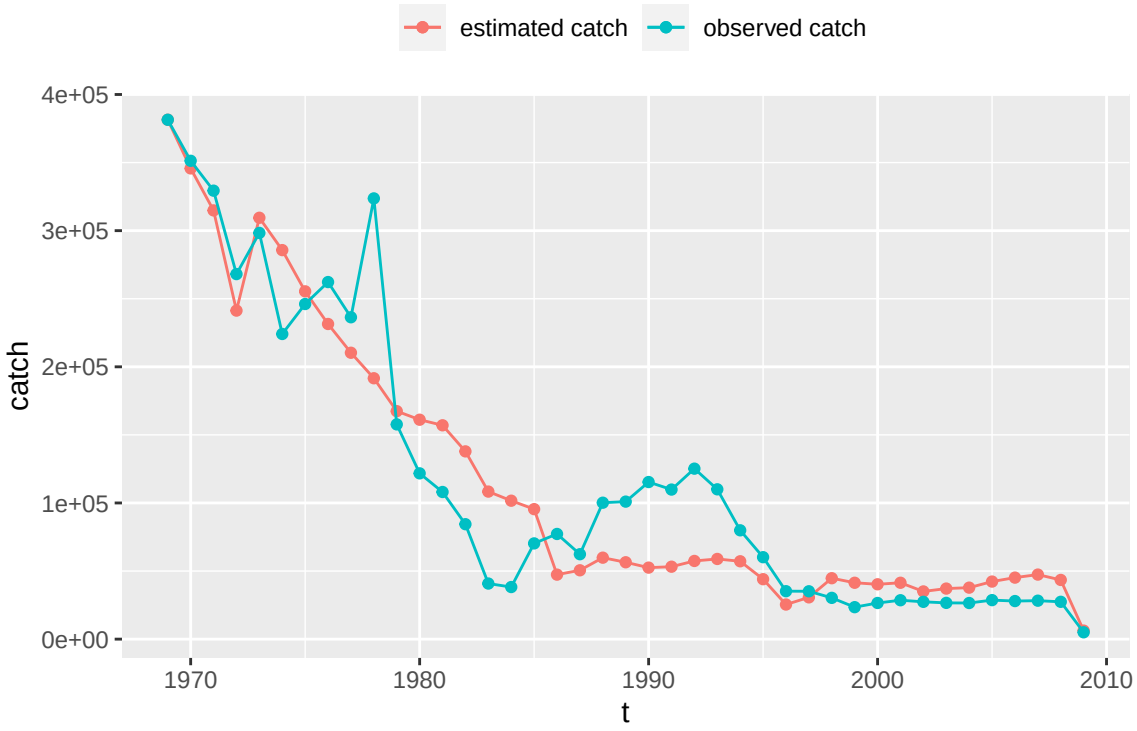
Plotting the estimated stock, as well as the catch-per-unit-effort divided by the catchability:

```
bh_fit %>%
  ggplot(aes(t, stock)) +
  geom_line(aes(color = "estimated stock")) +
  geom_point(aes(color = "estimated stock")) +
  geom_line(aes(x = year, y = cpue/(10^bh_mle$par[["q"]]), color = "CPUE/q"),
            data = natividad) +
  geom_point(aes(x = year, y = cpue/(10^bh_mle$par[["q"]]), color = "CPUE/q"),
             data = natividad) +
  theme(legend.position = "top",
        legend.title = element_blank())
```



And now plotting the estimated catch against our observed catch. Looks like a pretty good fit!

```
bh_fit %>%
  ggplot(aes(t, catch)) +
  geom_line(aes(color = "estimated catch")) +
  geom_point(aes(color = "estimated catch")) +
  geom_line(aes(x = year, y = catch, color = "observed catch"),
            data = natividad) +
  geom_point(aes(x = year, y = catch, color = "observed catch"),
             data = natividad) +
  theme(legend.position = "top",
        legend.title = element_blank())
```



Model Assumptions

We've assumed that our data are:

$$\text{obs. Catch}_t \sim \text{i.i.d. Normal}(\mu\text{Catch}_t, \sigma^2)$$

This implies a few things:

- The residuals are approximately normal (of course).
- The variance of the residuals is constant - σ^2 does not depend on the value of the response or the predictors.
- Once we know our estimated catch (μCatch_t), the observations are **independent** - in other words, knowing which part of Isla Natividad the data came from, or which year the data were collected in, would not provide us with any **additional** information that the model hasn't already.

Let's examine whether the assumptions are realistic. For some of these assumptions, it can be hard to tell with only 41 observations, but we should still give it a shot - if something is super off, it'll probably show up even in small samples.

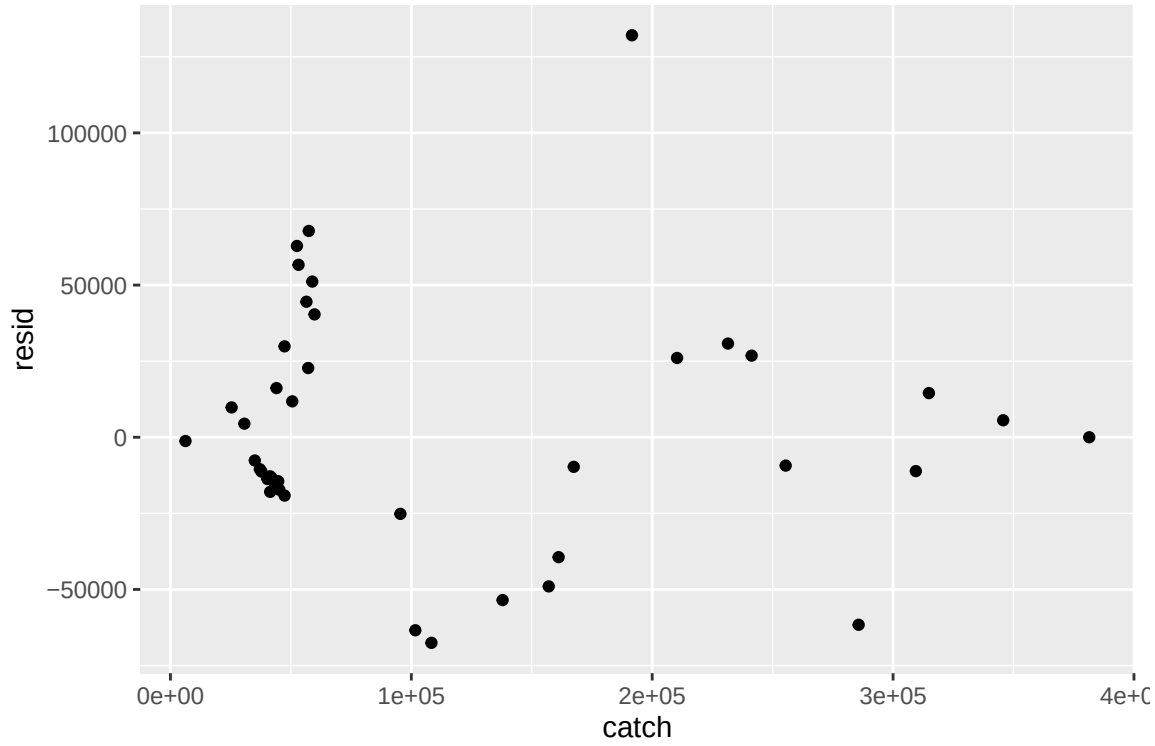
Heteroskedasticity

Let's plot the residuals (the observed catch minus the estimated catch) against the fitted values (the estimated catch). What we're looking for here is evidence of **heteroskedasticity** - whether the variance in the residuals is constant over the range of the fitted values.

How do you feel about these residuals? As your eyes move across the x-axis, does the dispersion along the y axis seem to change much?

```
bh_fit$resid <- natividad$catch - bh_fit$catch

bh_fit %>% ggplot(aes(catch, resid)) + geom_point()
```

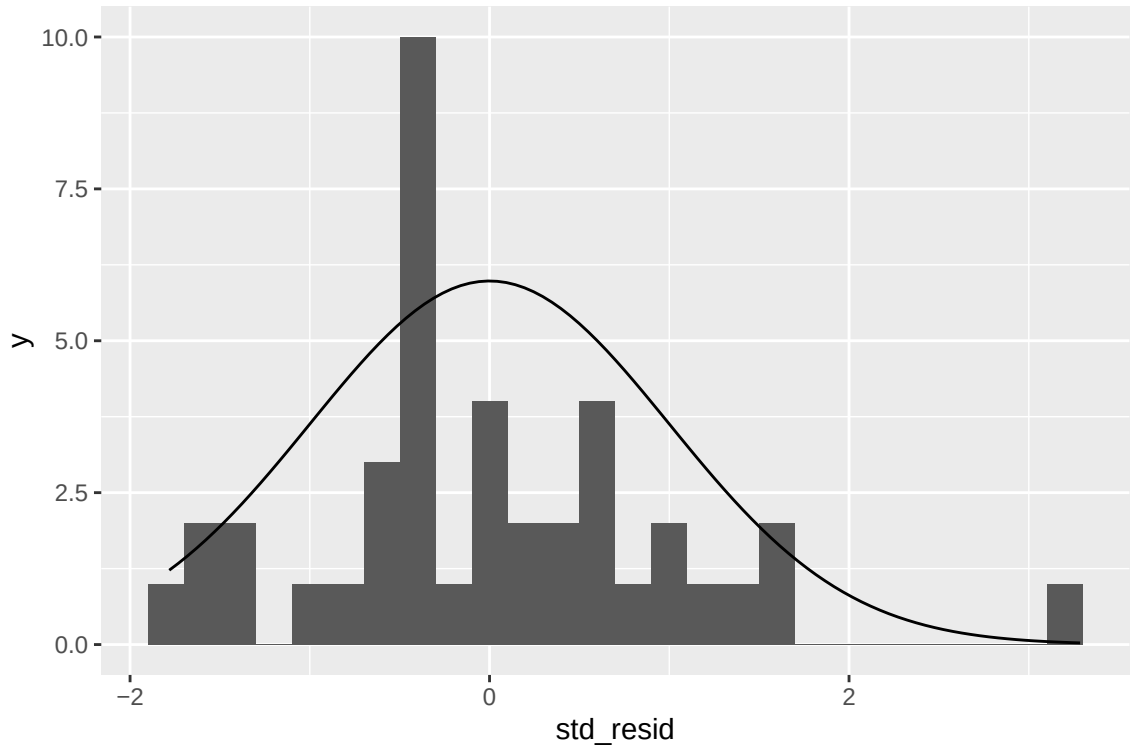


Normality of the Residuals

Now let's plot a histogram of the standardized residuals (I've superimposed a normal distribution for good measure).

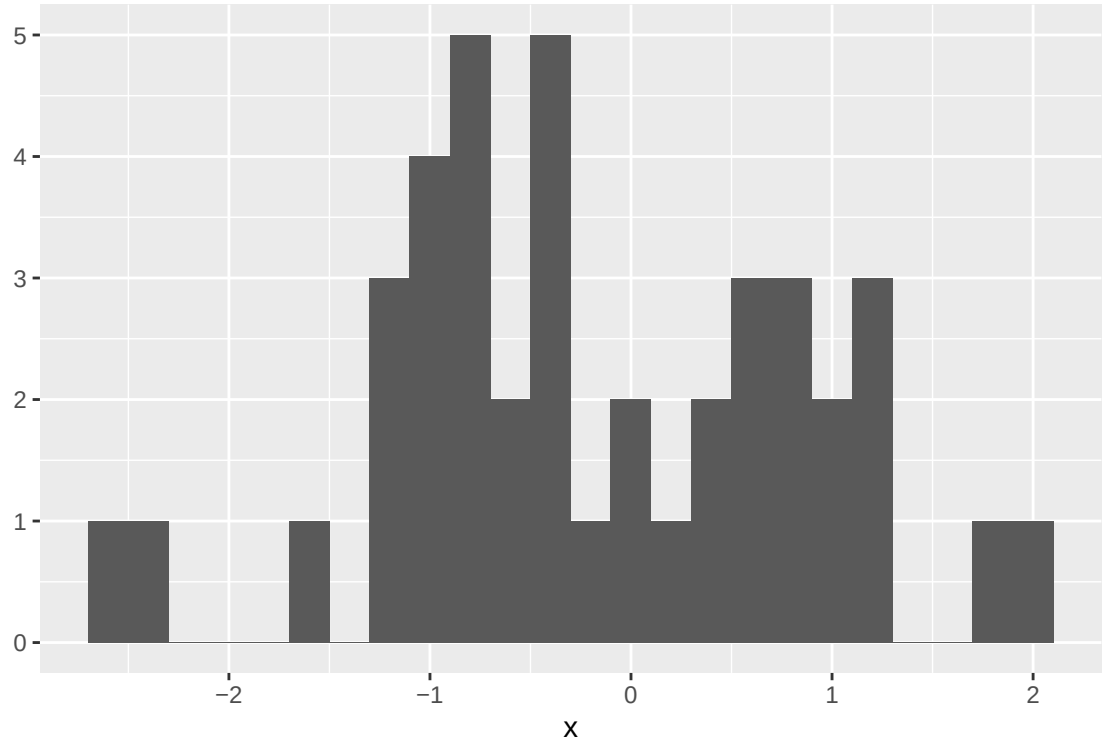
```
## standardized residuals
bh_fit$std_resid <- scale(bh_fit$resid)

bh_fit %>%
  ggplot(aes(std_resid)) +
  geom_histogram(binwidth = 0.2) +
  stat_function(fun = function(x, ...) dnorm(x, ...)*15,
                args = list(mean = 0, sd = 1))
```



Do these residuals look approximately normal to you? One good way to get a sense for what normal residuals should look like with 41 data points is to plot histograms of 41 random samples from a normal distribution and compare them. Do this three or four times by re-running the R chunk below. Do you think the residuals seem normal or not?

```
n <- nrow(natividad)
x <- rnorm(n, mean = 0, sd = 1)
qplot(x, binwidth = 0.2)
```



Residual Autocorrelation

Take a look at the above plot of the estimated catch vs. the observed catch - for any given year which we've over-estimated catch, it seems like we've also over-estimated catch for the adjacent years. Same goes for the years we under-estimate catch. Which of the assumptions that we outlined above does this violate? Why?

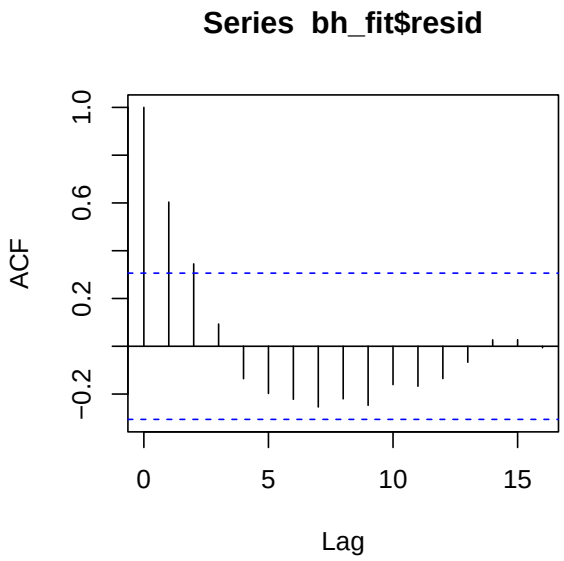
One way to check this is to plot the **autocorrelation function** (ACF) - the correlation of the residuals at time t with the residuals at time $t-1$. The fist plot below displays the ACF, starting with lag 0 (i.e., the correlation of the data at time t with that at time t, which is always equal to 1), and increasing in lag. The lag 1 autocorrelation is about 0.6 here, and the lag 2 autocorrelation is about 0.36.

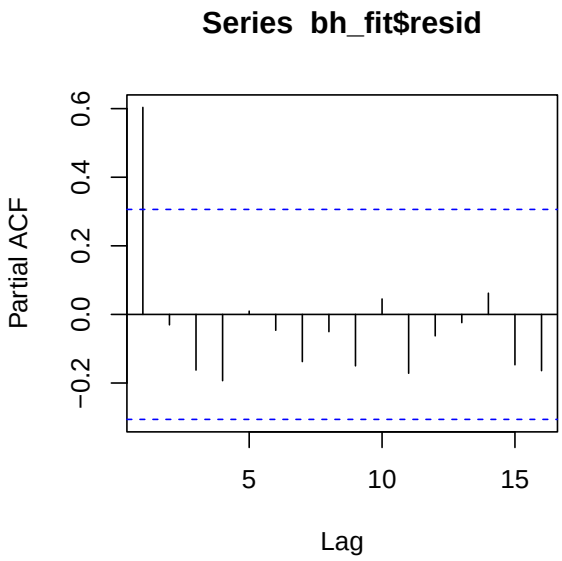
We can also plot the **partial autocorrelation function** - which, for each lag, is the ACF at that lage, *adjusted for the lower-order lags* - notice that the x-axis on this plot starts at lag 1, not lag 0.

What the combination of these plots tell us is that the correlation between the observed catch at any given time t and the observed catch at time $t-1$, after taking into account our model estimates for the catch, is *still* about 0.6!

That's a huge issue if we want standard errors and confidence intervals, because the model, in effect, thinks we have a lot more independent observations than we do.

```
par(mfrow = c(1, 2))
acf(bh_fit$resid)
acf(bh_fit$resid, type = "partial")
```






```
par(mfrow = c(1, 1))
```

We can fit an AR1 (auto-regressive order 1) model to the residuals to get an estimate of our residual autocorrelation (we'll call this the "AR1 coefficient").

```
resid_AR1 <- arima(bh_fit$resid, order = c(1, 0, 0))
phi <- as.numeric(coef(resid_AR1)[1])
phi
```

```
## [1] 0.5898104
```

Autoregressive Model

Residual covariance matrix

To address this issue, we're going to model the observations as arising from a **multivariate normal distribution**, where the expected covariance between the residuals of two observations is going to be a function of how far apart those observations are in time.

This means that, instead of a single value σ for the standard deviation of our normal distribution, we're going to have a **matrix** of variances (on the diagonal) and covariances (off the diagonal), which we derive from only two parameters: σ (our residual standard deviation) and ϕ, the correlation between adjacent observations.

Let's visualize the variance-covariance matrix by plotting it for different values of ϕ between 0 and 1. As you change the value of ϕ, what do you notice? Interpret this in the context of the model.

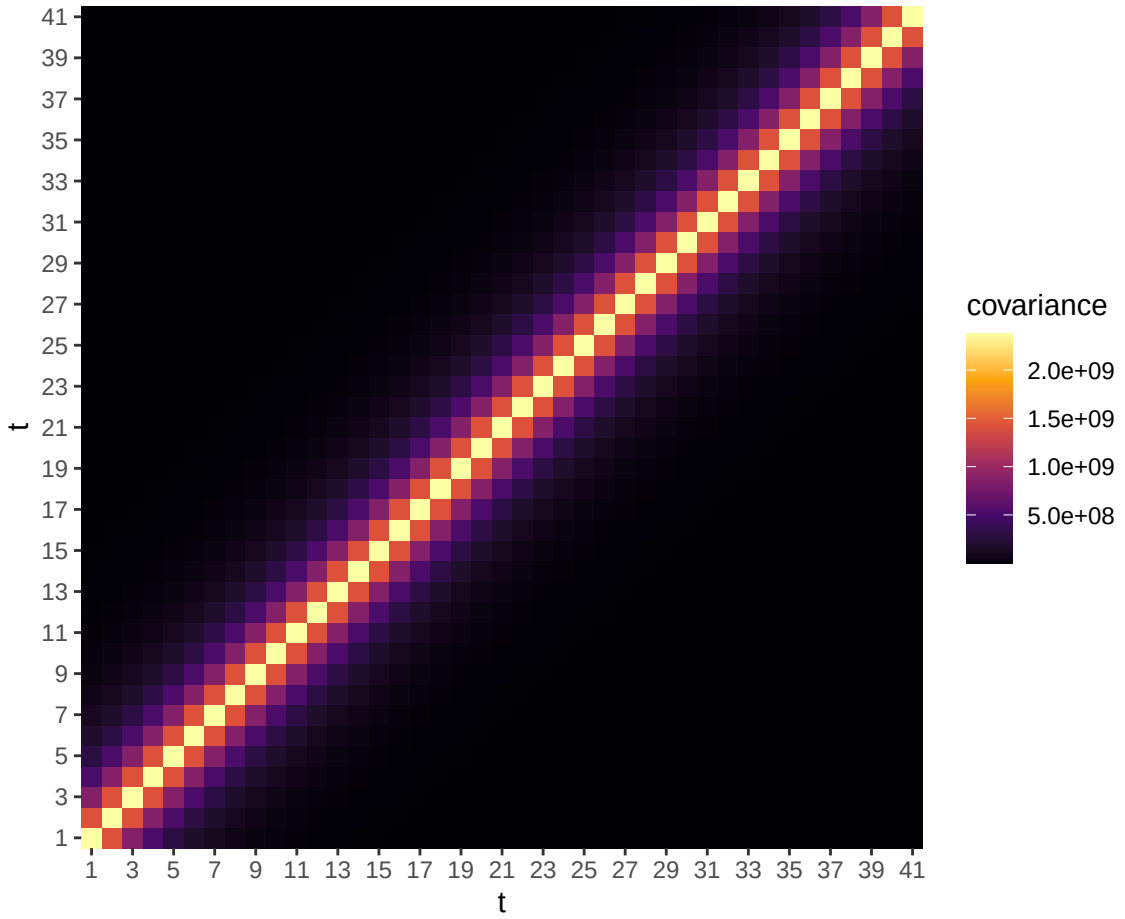
Given ϕ and σ, what is the covariance between a data point at time t and at time $t-1$? What about time t and $t-2$? Can you figure out a general expression for the covariance between data at time t and $t-\delta$?

```
plot_var_cov <- function(phi, sigma = bh_mle$par[["sigma"]]*1e4, tmax = nrow(natividad)) {

  var_cov <- sigma^2/(1-phi^2)*phi^(abs(outer(1:tmax, 1:tmax, "-")))
```

```
  reshape2::melt(var_cov) %>% ggplot(aes(Var1, Var2, fill = value)) +
    geom_raster() + scale_fill_viridis_c(option = "inferno") +
    scale_x_continuous(breaks = seq(1, tmax, 2)) +
    scale_y_continuous(breaks = seq(1, tmax, 2)) +
    coord_fixed(expand = FALSE) +
    labs(x = "t", y = "t", fill = "covariance")
}

plot_var_cov(phi = 0.6)
```



Likelihood Function

Now, let's modify the likelihood function we defined above to allow for autocorrelated observations. There's only a few changes here compared to the first version. What are they and what do they do?

```
beverton_holt_nll_AR1 <- function(par, t, obs_catch, obs_effort) {

  lambda <- par[1]; alpha <- 10^par[2]; q <- 10^par[3]; sigma <- par[4]*1e4; phi <- par[5]

  tmax <- length(t)

  mu_catch <- rep(0, tmax)
```

```
  mu_stock <- rep(0, tmax)

  mu_stock[1] <- obs_catch[1]/(1 - exp(-q * obs_effort[1]))

  for (ti in 1:tmax) {
    if(ti < tmax) {
      mu_stock[ti + 1] <- (lambda * mu_stock[ti] * exp(-q * obs_effort[ti])) /
        (1 + alpha * (mu_stock[ti] * exp(-q * obs_effort[ti])))
    }
    mu_catch[ti] <- mu_stock[ti] * (1 - exp(-q * obs_effort[ti]))
  }

  if (sigma >= 0) {
    var_cov <- sigma^2/(1-phi^2)*phi^(abs(outer(1:tmax, 1:tmax, "-")))
    nll <- -sum(dmvnorm(obs_catch, mean = mu_catch, sigma = var_cov, log = TRUE))
    return(nll)
  }  else {
    return(5000)
  }

}
```

Maximum Likelihood Estimates

Let's obtain maximum likelihood estimates for this model. How do these estimates compare to the previous ones? How do the standard errors compare? Why do you think the standard errors from this model differ from those from the model that does not include an autocorrelation term?

```
bh_mle_AR1 <- optim(
  par = c(lambda = 1.2, alpha = log(0.0025), q = log(0.049), sigma = 3.9, phi = phi),
  fn = beverton_holt_nll_AR1,
  t = natividad$year,
  obs_catch = natividad$catch,
  obs_effort = natividad$effort,
  control = list(maxit = 1000),
  hessian = TRUE
)

bh_mle_AR1
```

```
## $par
##     lambda      alpha          q      sigma        phi
##  1.0856174 -7.7196868 -4.3024733  3.0982155  0.5975134
##
## $value
## [1] 482.3976
##
## $counts
## function gradient
##      393       NA
##
## $convergence
## [1] 0
```

```
##
## $message
## NULL
##
## $hessian
##               lambda         alpha             q        sigma        phi
## lambda  8166.1650032 -5.378026e+02 -2.012410e+03 -0.195950534 169.450334
## alpha   -537.8026019  3.944884e+01  1.347246e+02  0.008666866 -10.834200
## q      -2012.4098090  1.347246e+02  5.115753e+02  0.038746307 -42.094332
## sigma     -0.1959505  8.666866e-03  3.874631e-02  8.550417746   0.611628
## phi      169.4503344 -1.083420e+01 -4.209433e+01  0.611628025  68.847602
```

```
var_cov_AR1 <- solve(bh_mle_AR1$hessian)
se_AR1 <- sqrt(diag(var_cov_AR1))
se_AR1
```

```
##     lambda      alpha          q      sigma        phi
## 0.06535688 0.51838632 0.26302083 0.34210089 0.12380389
```

Demographic Parameters

Density-dependence

Let's back-transform the estimate of α to obtain an estimate for the density-dependent parameter on the correct scale:

```
alpha <- 10^bh_mle_AR1$par[["alpha"]]
alpha
```

```
## [1] 1.906835e-08
```

Finite Growth Rate

We can obtain an asymptotic (Fisher) 95% confidence interval for λ with:

```
lambda <- bh_mle_AR1$par[["lambda"]]
lambda_CI <- qnorm(c(0.025, 0.975), lambda, sd = se_AR1[["lambda"]])
lambda_CI
```

```
## [1] 0.9575203 1.2137145
```

Compute an asymptotic confidence interval for λ using the model that does not include an AR1 error structure (the MLE is stored in `bh_mle` and the standard errors are stored in `se`).

```
### YOUR CODE HERE
```

Which confidence interval is wider? Based on the model *without* an AR1 correction, is there significant evidence that the population's finite growth rate is greater than 1? What about *with* an AR1 correction?

Carrying Capacity

We can obtain an estimate of the carrying capacity K using the estimates of λ and α:

$$\hat{K} = \frac{\hat{\lambda} - 1}{\hat{\alpha}}$$

The AR1 model gives an estimate of about 4,490,000 abalone:

```
K = (lambda - 1)/alpha
K
```

```
## [1] 4490026
```

The ordinary model agrees pretty closely:

```
as.numeric((bh_mle$par[["lambda"]] - 1)/(10^bh_mle$par["alpha"]))
```

```
## [1] 4486972
```

Maximum Sustainable Yield

For the Beverton-Holt model, the population size at MSY is given by:

$$N_{\text{msy}} = \frac{\lambda - \sqrt{\lambda}}{\alpha}$$

For our AR1 model, this gives:

```
N_msy = (lambda - sqrt(lambda))/alpha
N_msy
```

```
## [1] 2291113
```

MSY is given by:

$$Y_{\text{msy}} = N_{\text{msy}} - \frac{N_{\text{msy}}}{\lambda - \alpha \times N_{\text{msy}}}$$

Which, for our AR1 model, gives:

```
MSY = N_msy - N_msy/(lambda - alpha * N_msy)
MSY
```

```
## [1] 92199.68
```

Exercises

Read sections 11.1 - 11.4 of *Ecological Models and Data in R*, and answer the following questions based on the reading.

1. Process and Observation Error

What is observation error? What is process error? What problems can arise when we try to estimate parameters for systems with both process and observation error? For the Beverton-Holt model that we fit to the Isla Natividad abalone catches, which of these types of error did we use? Explain your answer.

2. Stochastic Simulation

Using the code provided below, run 15 simulations of (1) the observation-error-only Beverton-Holt model, and (2) the process-error-only Beverton-Holt model. The code for the observation error model is in the first chunk, and the code for the process error model is in the second chunk (all you need to do is re-run the code in each of these chunks at least 15 times each). How do the outcomes of these sets of simulations differ?

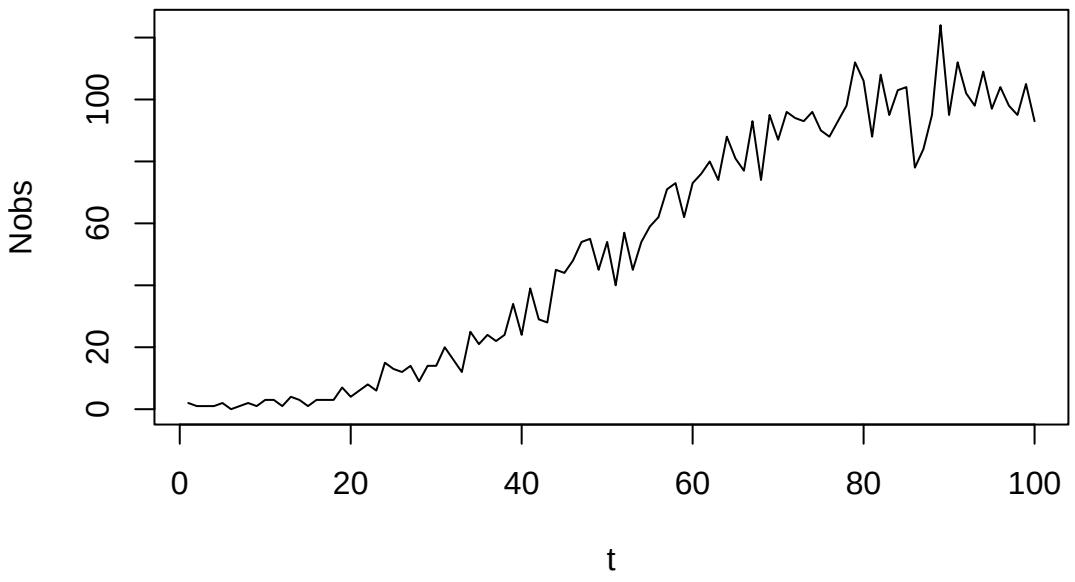
```r
### parameter values - same between observation and process models
N0 <- 10 # initial population size
lambda <- 1.1 ## finite growth rate
alpha <- 1e-04 ## density dependence
p <- 0.1 ## probability of observation

### empty population vectors
tmax <- 100 ## number of time steps
N <- rep(NA, tmax) ## true population size
Nobs <- rep(NA, tmax) ## observed population size
N[1] <- N0 ## initialize population

### loop over time, store true population
for (t in 1:(tmax - 1)) {
  N[t + 1] = (lambda * N[t])/(1 + alpha*N[t])
}
t <- 1:tmax

### observed population size
Nobs = rbinom(tmax, size = round(N), prob = p)

plot(Nobs ~ t, type = "l")
```




```r
### empty population vectors
tmax <- 100 ## number of time steps
Nproc <- rep(NA, tmax) ## true population size
Nproc[1] <- N0 ## initialize population
```

```r
### loop over time, store true population
for (t in 1:(tmax - 1)) {
  Nproc[t + 1] = rpois(1, lambda * Nproc[t])/(1 + alpha*Nproc[t])
}
t <- 1:tmax

plot(Nproc ~ t, type = "l")
```