FES 524 Winter 2022 Lab 6

Contents

Working with repeated measures

In lab 6 we will once again by working with a factorial design for the fixed effects, but now we will be explicitly accounting for possible correlations among errors due to repeated measurements taken over space. Analysis methods for repeated measurements through time are very similar.

Load R packages needed today

```
library(dplyr)
library(ggplot2)
library(nlme)
library(emmeans)
```

Read in the dataset

We will be working with the dataset `lab6.example.data.csv`, so make sure you have this file and have changed your working directory appropriately. As usual when we read in a dataset we'll take a look at the structure and make any necessary changes to it before fitting any models. Our factors of interest today are whether a birch is present or absent in a transect (`birch`) and the categorical distance of an oak tree from the birch or random point (`POSITION`). Our response variable is the height of oak trees (`HT`).

```
comp = read.csv("lab6.example.data.csv")

head(comp)
```

```
  AZIMUT POSITION  HT   birch transect
1     90        1 2.7 Nobirch       11
2     90        2 4.2 Nobirch       11
3     90        3 4.1 Nobirch       11
4      0        1 3.9 Nobirch       14
5      0        2 3.8 Nobirch       14
6      0        3 4.0 Nobirch       14
```

Check the structure of the dataset.

```
'data.frame':   114 obs. of  5 variables:
 $ AZIMUT  : int  90 90 90 0 0 0 270 270 270 0 ...
 $ POSITION: int  1 2 3 1 2 3 1 2 3 1 ...
 $ HT      : num  2.7 4.2 4.1 3.9 3.8 4 3.9 2.8 4.3 4 ...
 $ birch   : chr  "Nobirch" "Nobirch" "Nobirch" "Nobirch" ...
 $ transect: int  11 11 11 14 14 14 17 17 17 2 ...
```

Notice that some of the variable names are in all uppercase letters. I like to work with lowercase letters for ease of typing, so I'm going to assign new, lowercase names using the `tolower` function along with the `names` function.

```
?tolower
```

```
# Current names
names(comp)
```

```
[1] "AZIMUT"   "POSITION" "HT"       "birch"    "transect"
```

```
# Make names lowercase
names(comp) = tolower( names(comp) )
# Changed names
names(comp)
```

```
[1] "azimut"   "position" "ht"       "birch"    "transect"
```

Let's also make `position` and `transect` into factors, as they are currently integers but we will use them as categorical variables.

```
comp$position = factor(comp$position)
comp$transect = factor(comp$transect)
```

Unique names for the physical units

Checking for unique names using `semi_join()`

We need to check and make sure that all transects in the study have unique names. People often reuse the same names for the physical units of a study if the physical units are in different groups. We'll check to see if that's the case here, using `semi_join()` from **dplyr**.

`semi_join()` returns all rows of one dataset that has matching rows in the second dataset. In this case we'll compare the `Birch` subset of data to the `Nobirch` subset of data and see if they share any names. Notice I use `filter()` from **dplyr**, which is similar to `subset()` from base R. We join on `transect`.

Since the code returns rows we know we have shared transect names between the Birch and Nobirch groups. We can see that four of the transects share names between `Birch` and `Nobirch`.

```
semi_join( filter(comp, birch == "Birch"),
         filter(comp, birch == "Nobirch"),
         by = "transect")
```

```
   azimut position  ht birch transect
1      95        1 4.1 Birch       38
2      95        2 2.9 Birch       38
3      95        3 2.8 Birch       38
4     110        1 3.6 Birch       47
5     110        2 4.4 Birch       47
6     110        3 5.2 Birch       47
7     245        1 3.0 Birch       56
8     245        2 3.7 Birch       56
9     245        3 3.7 Birch       56
10    205        1 1.8 Birch       60
11    205        2 5.0 Birch       60
12    205        3 4.5 Birch       60
```

Making unique names via `paste()`

Since we have shared transect names between our two groups we can't use `transect` in the random effects as it currently is. R will treat all transects with the same name as if they were the same physical unit. Let's make the names unique by combining the levels of the `birch` variable with the levels of the `transect` variable.

This is the same trick we used last week to make unique names, but this week we are going to do it with the `paste()` function instead of using the `interaction()` function. The `paste()` function is useful in any situation where you want to combine any number of character strings together, and so has more general uses than `interaction()` does.

```
?paste
```

In the `paste()` function, we choose the variables we want to *paste* together along with the symbol we want to put between the pasted together values. By default, this symbol is a space. I'm going to change that to a period using `sep = "."`. Another common symbol to use as a separator is an underscore, `_`.

We'll make the new variable `transunique` using `paste()` inside `mutate()` to combine the levels of `birch` and `transect` to make unique transect names.

```
comp = mutate(comp, transunique = paste(birch, transect, sep = ".") )

# The new variable, with a unique name for every transect
comp$transunique
```

```
  [1] "Nobirch.11" "Nobirch.11" "Nobirch.11" "Nobirch.14" "Nobirch.14" "Nobirch.14" "Nobirch.17"
  [8] "Nobirch.17" "Nobirch.17" "Nobirch.2"  "Nobirch.2"  "Nobirch.2"  "Nobirch.20" "Nobirch.20"
 [15] "Nobirch.20" "Nobirch.23" "Nobirch.23" "Nobirch.23" "Nobirch.26" "Nobirch.26" "Nobirch.26"
 [22] "Nobirch.29" "Nobirch.29" "Nobirch.29" "Nobirch.32" "Nobirch.32" "Nobirch.32" "Nobirch.35"
 [29] "Nobirch.35" "Nobirch.35" "Nobirch.38" "Nobirch.38" "Nobirch.38" "Nobirch.41" "Nobirch.41"
 [36] "Nobirch.41" "Nobirch.47" "Nobirch.47" "Nobirch.47" "Nobirch.5"  "Nobirch.5"  "Nobirch.5"
 [43] "Nobirch.53" "Nobirch.53" "Nobirch.53" "Nobirch.56" "Nobirch.56" "Nobirch.56" "Nobirch.60"
 [50] "Nobirch.60" "Nobirch.60" "Nobirch.66" "Nobirch.66" "Nobirch.66" "Nobirch.69" "Nobirch.69"
 [57] "Nobirch.69" "Nobirch.78" "Nobirch.78" "Nobirch.78" "Nobirch.8"  "Nobirch.8"  "Nobirch.8"
 [64] "Nobirch.81" "Nobirch.81" "Nobirch.81" "Nobirch.84" "Nobirch.84" "Nobirch.84" "Birch.1"
 [71] "Birch.1"    "Birch.1"    "Birch.16"   "Birch.16"   "Birch.16"   "Birch.3"    "Birch.3"
 [78] "Birch.3"    "Birch.30"   "Birch.30"   "Birch.30"   "Birch.31"   "Birch.31"   "Birch.31"
 [85] "Birch.34"   "Birch.34"   "Birch.34"   "Birch.38"   "Birch.38"   "Birch.38"   "Birch.47"
 [92] "Birch.47"   "Birch.47"   "Birch.52"   "Birch.52"   "Birch.52"   "Birch.56"   "Birch.56"
 [99] "Birch.56"   "Birch.60"   "Birch.60"   "Birch.60"   "Birch.65"   "Birch.65"   "Birch.65"
[106] "Birch.68"   "Birch.68"   "Birch.68"   "Birch.75"   "Birch.75"   "Birch.75"   "Birch.82"
[113] "Birch.82"   "Birch.82"
```

Check the structure in the Environment pane.

```
'data.frame':   114 obs. of  6 variables:
 $ azimut     : int  90 90 90 0 0 0 270 270 270 0 ...
 $ position   : Factor w/ 3 levels "1","2","3": 1 2 3 1 2 3 1 2 3 1 ...
 $ ht         : num  2.7 4.2 4.1 3.9 3.8 4 3.9 2.8 4.3 4 ...
 $ birch      : chr  "Nobirch" "Nobirch" "Nobirch" "Nobirch" ...
 $ transect   : Factor w/ 34 levels "1","2","3","5",..: 6 6 6 7 7 7 9 9 9 2 ...
 $ transunique: chr  "Nobirch.11" "Nobirch.11" "Nobirch.11" "Nobirch.14" ...
```

Do we have unique names now? We can check by doing another `semi_join()`, using `transunique` as the joining variable.

No rows are returned, so it looks like we successfully made unique names for each transect.

```r
semi_join( filter(comp, birch == "Birch"),
         filter(comp, birch == "Nobirch"),
         by = "transunique")
```

```
[1] azimut      position    ht          birch       transect    transunique
<0 rows> (or 0-length row.names)
```

Initial data exploration

We'll start our data exploration by looking at our usual summary statistics. Notice that our two factors are still crossed as we have some observations for every factor combination. However, we do not have perfect balance between the `Birch` and `Nobirch` groups. This is the first time we've seen an unbalanced dataset this term.

Statistical summary

```r
( sumdat = comp %>%
    group_by("Birch" = birch, "Position" = position) %>%
    summarise(n = n(),
            mean = mean(ht),
            sd = sd(ht),
            min = min(ht),
            max = max(ht),
            .groups = "drop") )
```

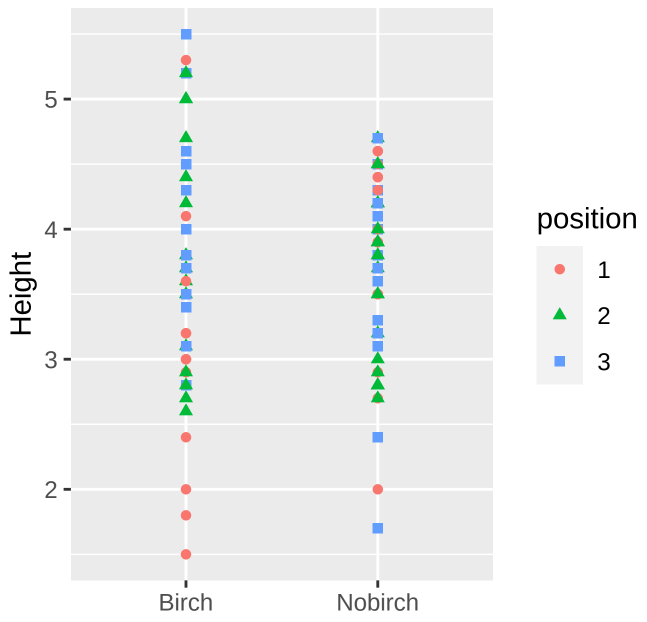
```
# A tibble: 6 x 7
  Birch   Position     n  mean    sd   min   max
  <chr>   <fct>    <int> <dbl> <dbl> <dbl> <dbl>
1 Birch   1           15  3.09 1.01    1.5   5.3
2 Birch   2           15  3.81 0.897   2.6   5.2
3 Birch   3           15  4.05 0.828   2.8   5.5
4 Nobirch 1           23  3.8  0.654   2     4.6
5 Nobirch 2           23  3.62 0.633   2.7   4.7
6 Nobirch 3           23  3.69 0.684   1.7   4.7
```

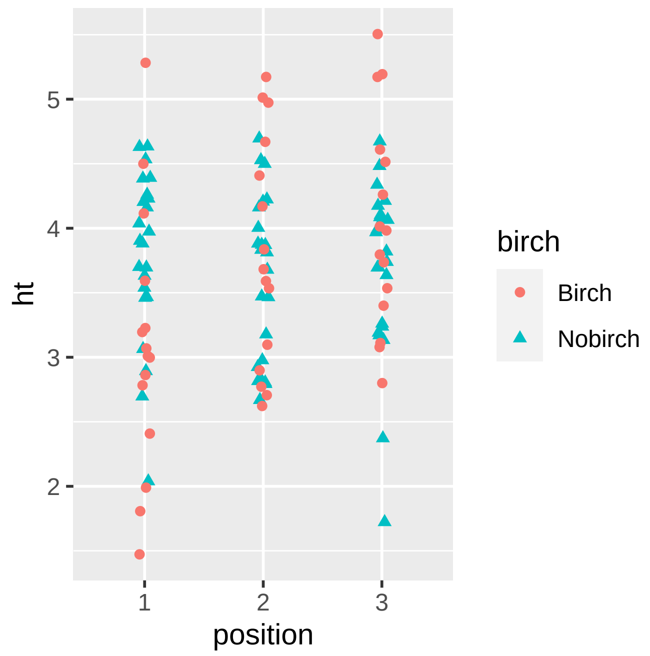
Graphical data exploration

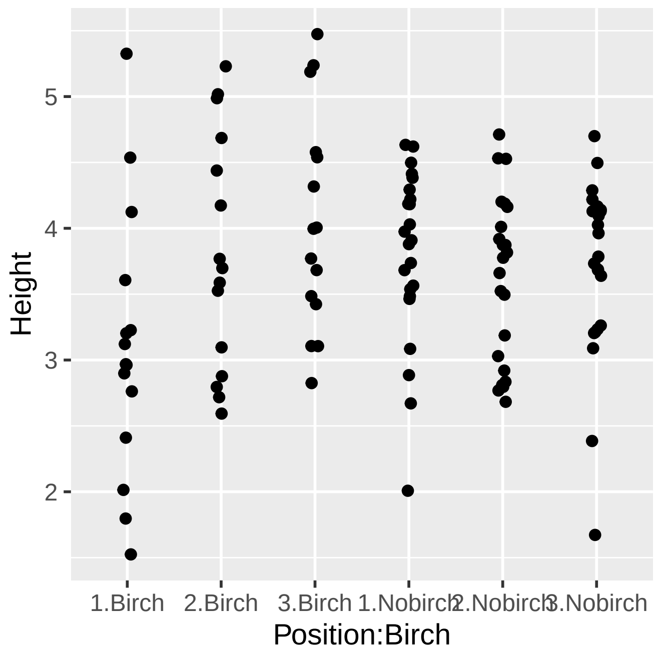
Below are some of the standard scatterplots we've been making each week to explore the raw data. You know how to make other plots that will help you when exploring your own dataset before analysis so we won't make all of them this week.

We can see that the heights for oaks in position 1 when a birch is present is more variable than other groups. This makes sense, since oaks vary in how close they are to the competing birch in the transect.

```r
# Scatter plot of height vs birch
qplot(x = birch, y = ht,
     color = position,
      shape = position,
      data = comp,
       xlab = "",
       ylab = "Height")

# Scatter plot of height vs position
# We need some jittering here to see the individual points better,
    # so switch to ggplot()
ggplot(comp, aes(x = position, y = ht,
              color = birch,
              shape = birch) ) +
    geom_jitter(width = .05)

# Scatter plot of factor combination vs ht, with jittering
ggplot(comp, aes(x = interaction(position, birch), y = ht) ) +
    geom_jitter(width = .05) +
    xlab("Position:Birch") +
    ylab("Height")
```



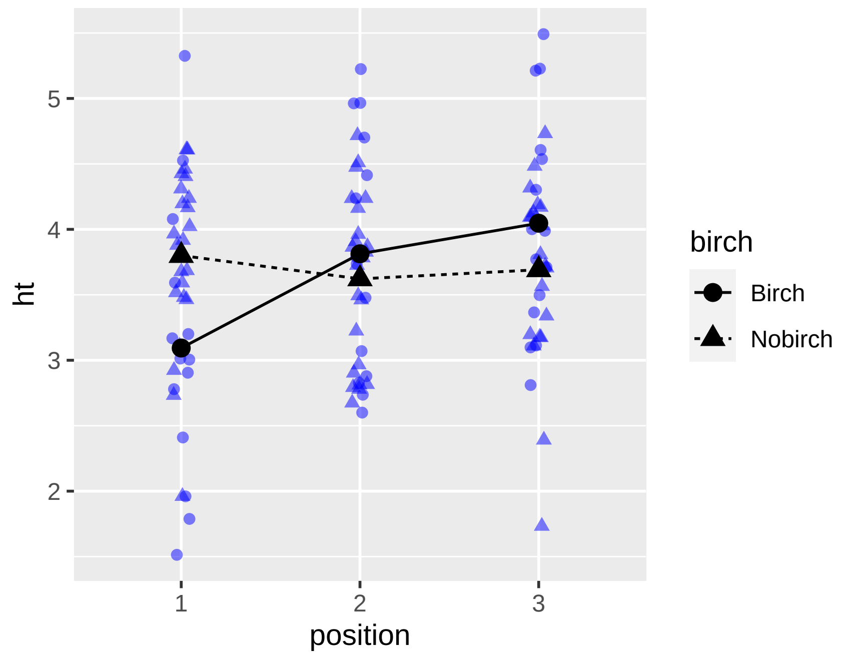




We'll make an interaction plot much like we have the last few weeks, to explore how the effect of birch presence may depend on the oak position. Researchers expected this interaction.

We'll use `geom_jitter()` to add the raw data points to avoid overlap, and use point shape to distinguish among groups. The aesthetics `alpha`, `size`, and `color` control how the points look when they are plotted. Notice the differences in mean height for birch presence vs absence appear fairly large at position 1. At the other positions the differences are in the opposite direction and are smaller in absolute value.

```
# Interaction plot
# We'll add the data on this week to make it clearer
    # that this is an exploratory plot
ggplot(comp, aes(x = position, y = ht,
              group = birch, linetype = birch, shape = birch) ) +
    geom_jitter(alpha = .5, width = .05, size = 1.75, color = "blue" ) +
        # Jitter observed data and make more transparent
    stat_summary(fun = mean, geom = "point", size = 3) + # Add points for means
    stat_summary(fun = mean, geom = "line") # Connect points with lines
```



Initial model for repeated measures data

The design of this study has a repeated measures element to it, as three trees were measured in each transect. We believe that trees closer in space might be more alike because they share the same local environment (e.g., soil, microclimate). This is similar to how we think about correlations through time.

Whenever we have repeated measurements in space or time we are going guess that the assumption of independence of errors might not be reasonably met, even after accounting for all the effects in the model. Today we will learn how to relax the assumption that the errors within transects are independent of each other. We have actually already relaxed this assumption by using mixed models. Today we'll formally discuss what sort of correlation we've been assuming using mixed models and then go on to discuss other types of correlation structures we can use in package **nlme**.

Mixed model

Let's fit the initial model based on the study design, where individual trees are nested within transects and so transect is included as a random effect.

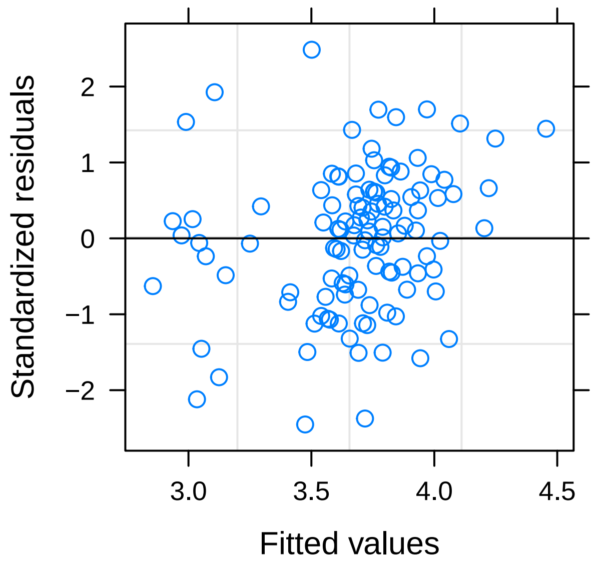
```
# Initial model based on design (trees nested in transects)
init = lme(ht ~ birch*position, data = comp, random = ~1|transunique)
```

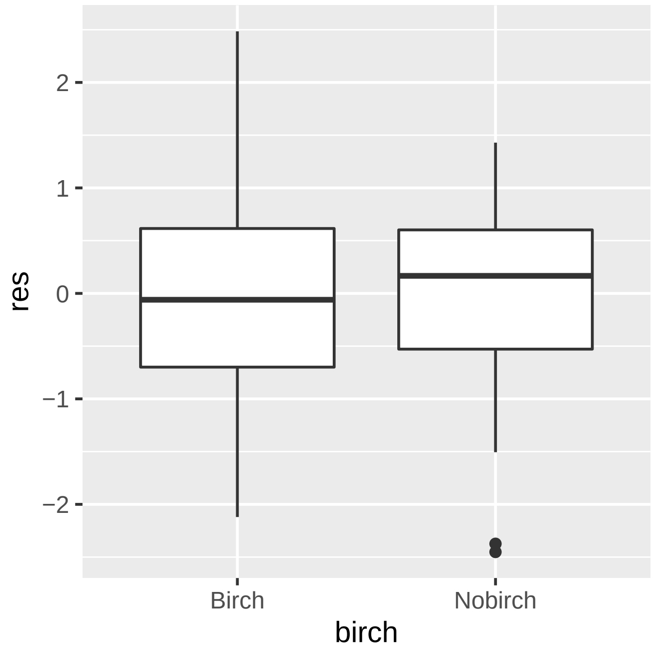
Initial check of assumptions We will check the assumptions via the residuals of this model, looking specifically for problems with nonconstant variance and normality (symmetry) as we have every week.

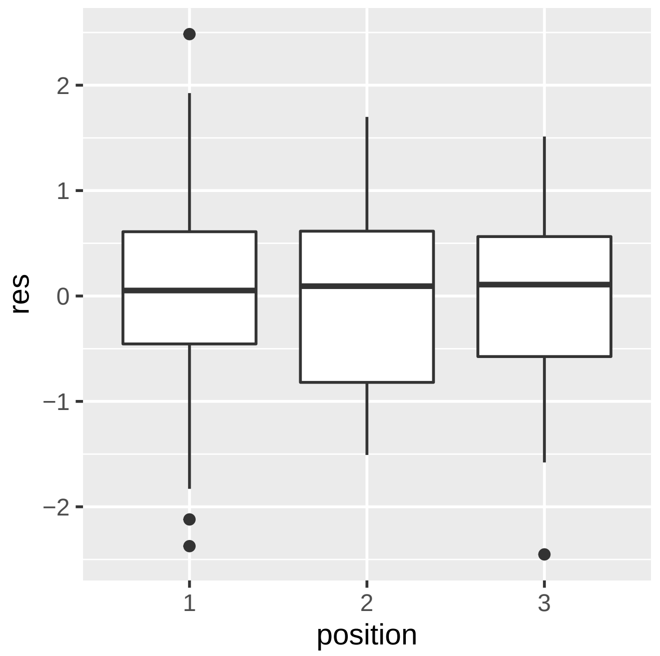
There is a possibility that the spread varies among groups this week, primarily due the trees in the first position when a birch is present. The spread of one group was less than twice as large as other groups, though, so I decided that the differences in spread weren't large enough to be overly concerned about. Someone else could have certainly made a different choice.

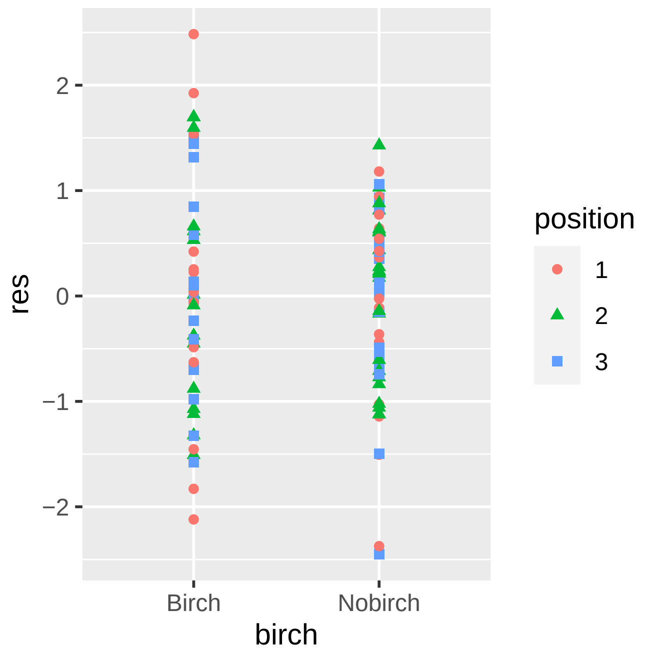
```
# Residuals
comp$res = residuals(init, type = "pearson")

# Let's look at our standard residual plots
plot(init)

# Boxplots
qplot(x = birch, y = res,
     data = comp, geom = "boxplot")
qplot(position, res,
     data = comp, geom = "boxplot")
```

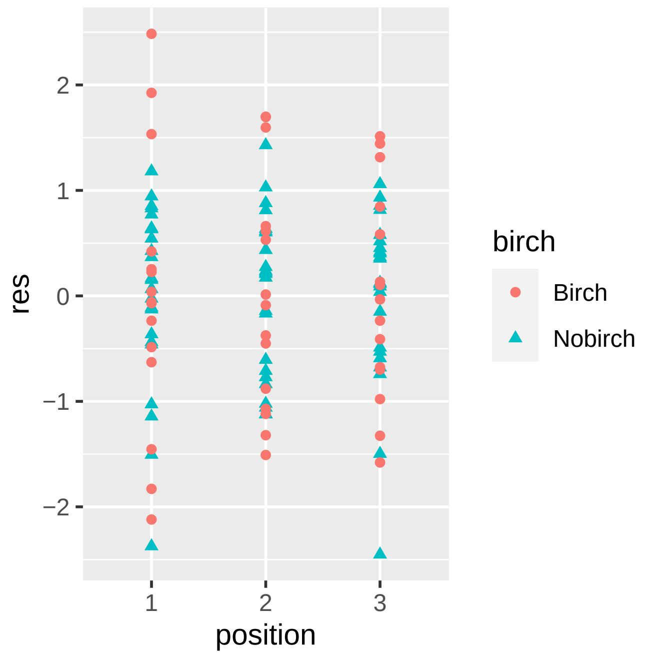
```r
# Scatterplots
qplot(x = birch, y = res,
      color = position,
      shape = position,
      data = comp)
qplot(x = position, y = res,
      color = birch,
      shape = birch,
      data = comp)

# Let's dodge this last scatterplot to get a better picture
    # (so switch to ggplot)
ggplot(comp, aes(x = position, y = res,
               color = birch,
               shape = birch)) +
     geom_point(position = position_dodge(width = .5) )

# Check symmetry of residuals
qplot(x = "res", y = res,
     data = comp, geom = "boxplot")
```

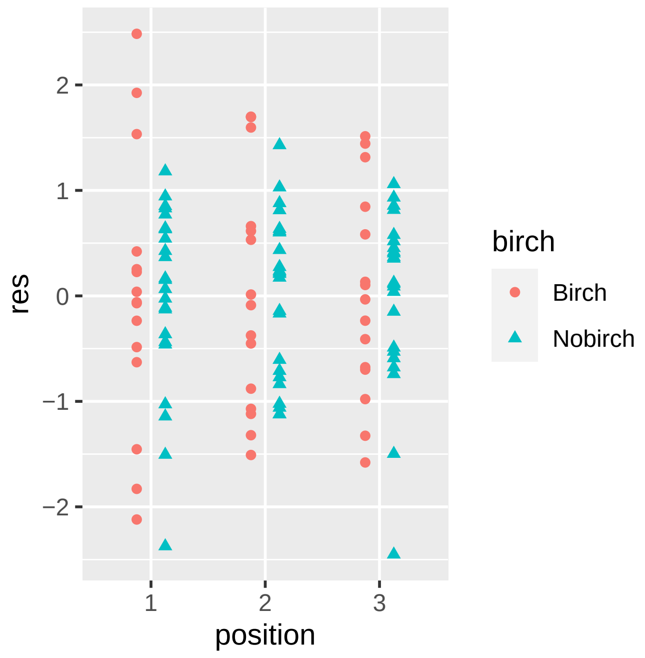


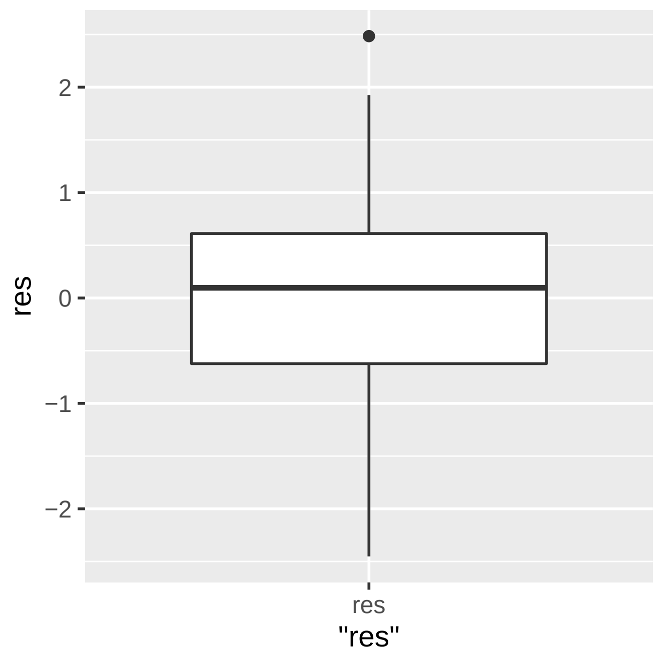



res
2
1
0
−1
−2
1
2
3
position


res
2
1
0
−1
−2
Birch
Nobirch
birch
position
● 1
▲ 2
■ 3


res
2
1
0
−1
−2
1
2
3
position
birch
● Birch
▲ Nobirch


res
2
1
0
−1
−2
1
2
3
position
birch
● Birch
▲ Nobirch



How to check for autocorrelation in the residuals

Remember that we measured each transect three times (three different trees measured in each transect). Ideally, we could graphically check the residuals for any evidence of autocorrelation. Unfortunately, checking the residuals for correlation is difficult when there are only a few repeated measurements. If we had more measurements per transect, we would have more tools at our disposal for exploring the type of correlation that is present among residuals within a transect. I'll outline those tools below so you know where to start if you run into situations like these in your own work.

Equal spacing among repeated measurements When we have equal spacing among measurement times or locations (often years or days but also things like depth), we would look for patterns in plots of residuals vs time/location. We could also use autocorrelation function (ACF) plots on the residuals to check for possible time series correlation structures. The **nlme** package has the `ACF()` and `plot.ACF` functions for this reason.

Unequal spacing among repeated measurements For spatial data or time series data with unequal spacing among the measurement locations/times, we can use semivariograms to explore the correlation among the residuals along the continuous distance variable(s). The **nlme** package functions `Variogram()` and `plot.Variogram` are available for this purpose. For spatial data we can also use bubble plots, where we plot the residuals across the landscape using the x and y coordinates as the axis variables. In a bubble plot the size of the points change based on the size of the residual. See Zuur et al. 2009 Chapter 7 for good examples.

Two good sources for information on correlation structures and how to use them in package **nlme** are:

1. Pinheiro and Bates, 2000, Mixed Effects Models in S and S-plus. This is the book by the authors of package **nlme**. See chapter 5 for working with correlation structures.
2. Zuur et al. 2009, Mixed Effects Models and Extensions in Ecology with R. This book covers time series and spatial non-independence and how to fit models in package **nlme** in Chapters 6 and 7. Ignore their poor advice on model selection.

Correlations for within-group errors in `lme()`

Because we have no good way to check the residuals for the presence of autocorrelation with very small repeated measures studies, our only option is to fit a few models with reasonable correlation structures for the errors within transects and choose among them with some statistical metric.

In this case we'll fit models that allow for specific correlation structures among errors within transects. For a full list and description of correlation structures in package **nlme**, see the help page for the function `corStruct()`. The trees in the transects are equally spaced, so simple time-series type correlation structures are reasonable in this case.

```
?corStruct
```

The analysis approach when there are only few repeated measurements

You'll see me use the word *reasonable* a lot throughout this section of lab. We're not using a "kitchen sink" modeling approach here, where we fit as many models as we can think of and choose among them based on some statistical metric. That approach has a lot of problems. Instead we are doing some hard thinking to come up with a few plausible alternatives for a correlation structure to make sure we have a final model where the assumptions have been *reasonably* met.

Correlation structure of a linear mixed model

Let's start by talking about the correlation structure of our linear mixed model, `init`, a little more. This is the first model under consideration. It's possible that the mixed model has already accounted for any within-transect error correlations and we can simply proceed with this model. This is because the mixed model *induces* a correlation structure based on assuming a systematic effect of the random effect (transect in this example).

To use a linear mixed model we assume transects are far enough apart to be independent of each other. Correlations are only assumed *within* transects.

Our mixed model assumes constant correlation among observations from different positions within transects. Note that for all the explicit correlations discussed below, we are assuming the *errors* are correlated after accounting for the transect effect. In a mixed model, the correlation is among observations after accounting for the fixed effects (aka, marginal errors). It may be that the errors are independent after accounting for the correlation induced by the transect random effect.

Constant correlation means we would think it was reasonable that every position in the transect is correlated in the same way with every other position in the same transect. So, e.g., position 1 would be correlated the same amount with position 2 and with position 3 across all transects.

For a mixed model, the correlation among observations within units can only fall between 0 and 1; the correlation is forced to be positive. Positive correlation is common, but there are situations where we might have negative correlations within groups. A negative correlation indicates that groups are less alike within the group than between groups. An example of this is some sports teams, where members are chosen because they have very different skills.

As a reminder, here's what the model looks like in R. The correlation is implied (aka *induced*), and we don't explicitly put in the `correlation` argument.

```
init = lme(ht ~ birch*position, data = comp, random = ~1|transunique)
```

Compound symmetry

We can account for any additional correlation among errors that is not accounted for by the mixed model by defining reasonable correlation structures in the model using the `correlation` argument of `lme()`. A simple correlation structure we could assume is compound symmetry, using `corCompSymm()`.

The compound symmetry model assumes constant correlation among errors from different positions within a transect, much like the implied correlation of the mixed model. However, the correlation among repeated measurements with compound symmetry can fall between -1 and 1. The difference between compound symmetry and the correlation induced by a mixed model is that the correlations can be negative when using compound symmetry. Many times, in practice, the compound symmetry correlation structure doesn't add anything beyond the induced correlation of the mixed model.

The code below shows how we add the `correlation` argument and allow for compound symmetry. We make it clear that the correlation only occurs within transects by explicitly using the `transunique` variable after the `|` in the `form` equation. Remember, we are assuming that transects are independent from each other and are only focusing on the *within transect* correlation.

We also need a variable that represents the order of the positions of the trees so R knows the spacing of the positions in each transect (so position 1 is 1 unit from position 2 and 2 units away from position 3). This variable needs to be an integer for equally spaced repeated measures (i.e., 1, 2, 3), not a factor. Because `position` is a factor with the order of the levels in the order of the tree positions, we can use `position` with `as.integer()` directly in `form`.

See the `form` argument description on the help page for `corCompSymm()` for more information on the integer-valued covariate and the grouping variable. While including the integer-valued variable is always safest, it isn't always strictly necessary if the data are in order within each group already.

Notice that when we allow for correlations, we get new output giving estimates of correlation parameters. The correlation the model estimated is called `Rho` in the model output when using the compound symmetry correlation structure.

```
( cs = lme(ht ~ birch*position, data = comp, random = ~1|transunique,
    correlation = corCompSymm(form = ~ as.integer(position)|transunique) ) )
```

```
Linear mixed-effects model fit by REML
  Data: comp
  Log-restricted-likelihood: -132.7577
  Fixed: ht ~ birch * position
          (Intercept)           birchNobirch              position2              position3
            3.0933333              0.7066667              0.7200000              0.9533333
birchNobirch:position2 birchNobirch:position3
           -0.8982609             -1.0620290

Random effects:
 Formula: ~1 | transunique
        (Intercept)  Residual
StdDev:   0.2682298 0.7186091

Correlation Structure: Compound symmetry
 Formula: ~as.integer(position) | transunique
 Parameter estimate(s):
        Rho
-0.01468935
Number of Observations: 114
Number of Groups: 38
```

Autoregressive lag 1

Autoregressive correlation of lag 1, called *AR1*, is a reasonable and relatively simple structure when repeated measurements are equally spaced. It is a structure most often used for time series, but can also be used for repeated measurements in space given equal distances between locations.

We use an AR1 correlation when we think things closer together in time or space might be more correlated than things that are farther apart. The correlation among repeated measurements one unit apart in time or space can range between -1 and 1. A negative correlation indicates observations one unit apart are dissimilar, while a positive correlation indicates the opposite.

With AR1, we only estimate a single correlation. In this case, the errors from positions one unit apart will be estimated to have a correlation of `Phi`. Measurements further apart are less correlated. The correlation between errors in positions two units apart is `Phi` squared. If we had more trees in a transect, the correlation between errors in positions three units apart would be `Phi` cubed, etc.

When we code for AR1 correlation, we will use the `corAR1()` function. We again indicate that correlations are within each transect only and tell R the order of the positions in the transect via `as.integer(position)`. The estimated correlation is called `Phi` for the AR1 correlation structure.

```
( ar1 = lme(ht ~ birch*position, data = comp, random = ~1|transunique,
    correlation = corAR1(form = ~ as.integer(position)|transunique) ) )
```

```
Linear mixed-effects model fit by REML
  Data: comp
  Log-restricted-likelihood: -132.1092
  Fixed: ht ~ birch * position
          (Intercept)           birchNobirch              position2              position3
            3.0933333              0.7066667              0.7200000              0.9533333
birchNobirch:position2 birchNobirch:position3
           -0.8982609             -1.0620290

Random effects:
 Formula: ~1 | transunique
         (Intercept)  Residual
```

```
StdDev: 8.833892e-05 0.7672748

Correlation Structure: AR(1)
 Formula: ~as.integer(position) | transunique
 Parameter estimate(s):
     Phi
0.187699
Number of Observations: 114
Number of Groups: 38
```

General correlation structure

The most complicated correlation structure we can fit is referred to as the *general* correlation structure. It is very complex, as many correlations are estimated. This correlation structure is often too complex to be useful when we have longer time series or many repeated measurements in space but can be useful when we have only a few repeated measures.

The reason this is such a complicated correlation structure is that errors from each position are allowed to have a different correlation with errors from every other position in the transect. In this case, with three positions, this means we'll be estimating three unique correlations instead of only one like we did when using compound symmetry and AR1. In a general correlation structure, all correlations can fall between -1 and 1.

We use the `corSymm()` function to fit a general correlation structure. Notice that we now get three correlations in the model output, one for each pair of tree positions. In the output below, you can see the estimated correlation between positions 1 and 2 is `-0.367` but the estimated correlation between positions 2 and 3 is `0.555`, so in the opposite direction.

```r
( gen = lme(ht ~ birch*position, data = comp, random = ~1|transunique,
    correlation = corSymm(form = ~ as.integer(position)|transunique) ) )
```

```
Linear mixed-effects model fit by REML
  Data: comp
  Log-restricted-likelihood: -124.7057
  Fixed: ht ~ birch * position
          (Intercept)           birchNobirch                position2               position3
            3.0933333              0.7066667                0.7200000               0.9533333
birchNobirch:position2 birchNobirch:position3
           -0.8982609             -1.0620290

Random effects:
 Formula: ~1 | transunique
        (Intercept)  Residual
StdDev:   0.2825362 0.7181328

Correlation Structure: General
 Formula: ~as.integer(position) | transunique
 Parameter estimate(s):
 Correlation:
  1      2
2 -0.367
3 -0.201  0.555
Number of Observations: 114
Number of Groups: 38
```

Allowing for both variance heterogeneity and correlations

We can simultaneously relax the assumption of independence and the assumption of constant variance by using the `correlation` and `weights` argument at the same time. We saw the `weights` argument previously in lab 4.

When checking model assumptions initially, I decided any differences in residual spread weren't large enough to need to allow for the variances to be different among the levels of `birch` in this example. However, I wanted to show you how this works for use in your own work and so you have the option when working on the assignment.

We'll first fit a model allowing for compound symmetry and different variances between `Birch` and `Nobirch`. Now we have even more information in the model output.

```
( cshet = lme(ht ~ birch*position, data = comp, random = ~1|transunique,
    correlation = corCompSymm(form = ~ as.integer(position)|transunique),
    weights = varIdent(form = ~ 1|birch) ) )
```

```
Linear mixed-effects model fit by REML
  Data: comp
  Log-restricted-likelihood: -129.9841
  Fixed: ht ~ birch * position
           (Intercept)           birchNobirch                position2                position3
             3.0933333              0.7066667                0.7200000                0.9533333
birchNobirch:position2 birchNobirch:position3
            -0.8982609             -1.0620290

Random effects:
 Formula: ~1 | transunique
        (Intercept)  Residual
StdDev: 0.0001207712 0.9104519

Correlation Structure: Compound symmetry
 Formula: ~as.integer(position) | transunique
 Parameter estimate(s):
      Rho
0.1009945
Variance function:
 Structure: Different standard deviations per stratum
 Formula: ~1 | birch
 Parameter estimates:
    Birch   Nobirch
1.0000000 0.7233659
Number of Observations: 114
Number of Groups: 38
```

We just wrote out a whole new model in order to add the `weights` argument. However, notice that the model above is the same as the `cs` model with a `weights` argument added. When we are building on existing model objects it can be useful to *update* the model with whatever we want to add using the `update()` function.

```
?update
```

We can allow variance heterogeneity with any of the correlation structures we've used. Here is one last example, updating the `gen` model with the general correlation structure to allow for nonconstant variance among the levels of `birch`. We'll use the `update()` function for this to save typing, *updating* the `gen` model with the `weights` argument.

```
( genhet = update(gen, weights = varIdent(form = ~ 1|birch) ) )
```

```
Linear mixed-effects model fit by REML
  Data: comp
  Log-restricted-likelihood: -120.9696
  Fixed: ht ~ birch * position
           (Intercept)           birchNobirch                position2                position3
             3.0933333              0.7066667                0.7200000                0.9533333
birchNobirch:position2 birchNobirch:position3
            -0.8982609             -1.0620290

Random effects:
 Formula: ~1 | transunique
        (Intercept)  Residual
StdDev: 9.726747e-05 0.9453126
```

```
Correlation Structure: General
 Formula: ~as.integer(position) | transunique
 Parameter estimate(s):
 Correlation:
  1      2
2 -0.218
3 -0.081  0.630
Variance function:
 Structure: Different standard deviations per stratum
 Formula: ~1 | birch
 Parameter estimates:
    Birch   Nobirch
1.0000000 0.6856724
Number of Observations: 114
Number of Groups: 38
```

Choosing among models with different covariance structures

Once you have a *few* reasonable models to account for any within-transect correlation and/or non-constant variance, you can choose among them using an information criterion of some sort. This is possible when we have not changed the fixed effects portion of the model in any way and have fit the model using the default `lme()` fitting method, `REML`. We cannot do likelihood ratio tests, which are like the extra-sums-of-squares F tests you've learned in previous classes, for all of these models because the models are not nested. In other words, each model isn't a special case of another model.

Using AIC or BIC via the `anova()` function

We can use the `anova()` function to use AIC or BIC to compare models. For both AIC and BIC, smaller is better. If you want to use AIC, you may want to switch to using AICc (see Burnham and Anderson 2002 for more information), which we don't get from the `anova()` function for `lme()` objects.

We have four models under consideration here. I didn't think the possible nonconstant variance between birch presence/absence was large enough to be concerned about, so I won't entertain any of those models as part of my model set. If I did think the nonconstant variance of the errors was a problem, I would *only* have fit and chosen among models that allowed for nonconstant variance because the models without it wouldn't be reasonable options.

Today we will use AIC for choosing among four models: the mixed model `init`, the compound symmetry model `cs`, the AR1 model `ar1`, and the general correlation model `gen`.

```r
anova(init, cs, ar1, gen)
```

```
     Model df      AIC      BIC    logLik   Test  L.Ratio p-value
init     1  8 281.5153 302.9724 -132.7577
cs       2  9 283.5153 307.6545 -132.7577 1 vs 2  0.00000   1e+00
ar1      3  9 282.2183 306.3575 -132.1092
gen      4 11 271.4114 300.9148 -124.7057 3 vs 4 14.80701   6e-04
```

We will take the model with the smallest AIC value as our *best* model of the four, and then recheck our assumptions to make sure the model has met assumptions before using it for inference. We would need to report the results of this model selection in our write-up, outlining the four models we considered and the values of the information criterion we used to make our decision. I usually make the results into a table to include in my write up.

If you are using something like AIC to compare different *fixed effects* structures, it is not standard to simply choose a best model like we did here. Instead we might use some sort of multi-model inference or switch to doing exploratory data analysis instead of confirmatory data analysis such as we've been doing in this class.

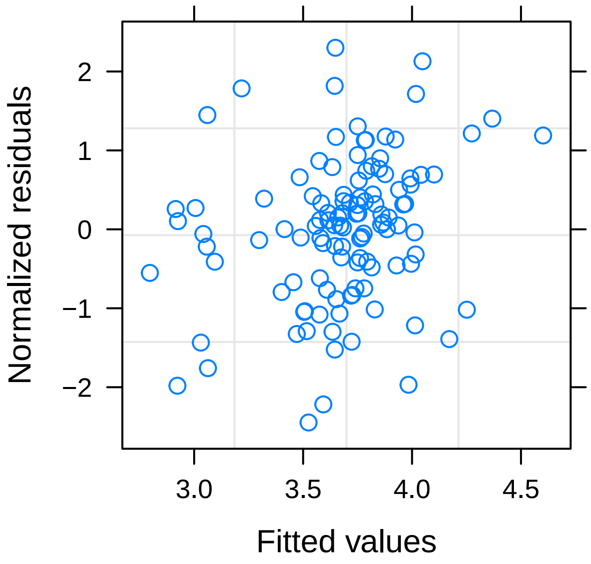
I want you to notice that the log likelihood of the mixed model and the compound symmetry model are identical; this indicates that the correlations in the two models are the same. As often happens, allowing for compound symmetry didn't add anything beyond the implied correlation of the mixed model.

Checking assumptions of the chosen model

Based on AIC, I chose the model with the general correlation structure as a "best" model. Now we'll need to recheck our assumptions with this model. If we see problems in our residual plots it doesn't matter if the model was "chosen" by some statistical metric; it wouldn't be valid for inference because the assumptions are not met.

Using normalized residuals for `lme()` models with correlation structures When allowing for correlation structures in `lme()`, we need to check our assumptions using the *normalized* residuals. Here is how we get the normalized residuals when making the residuals vs fitted plot for **lme** objects.

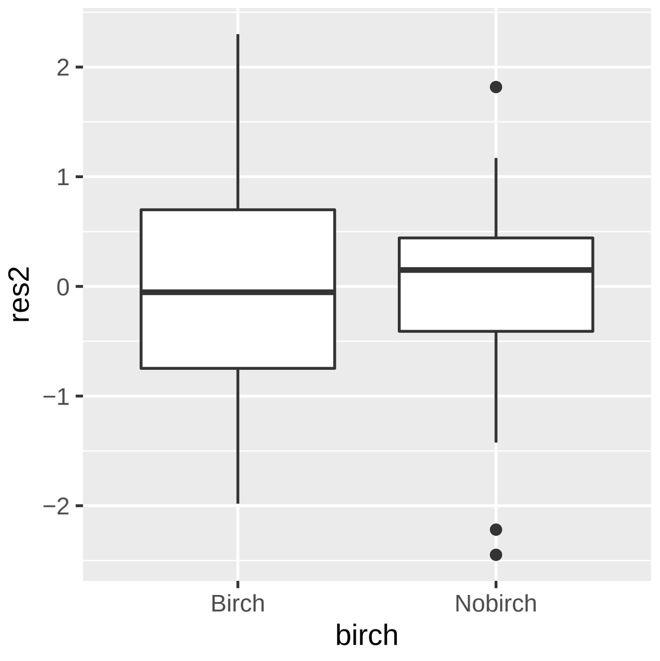
```
plot(gen, resid(., type = "normalized") ~ fitted(.) )
```

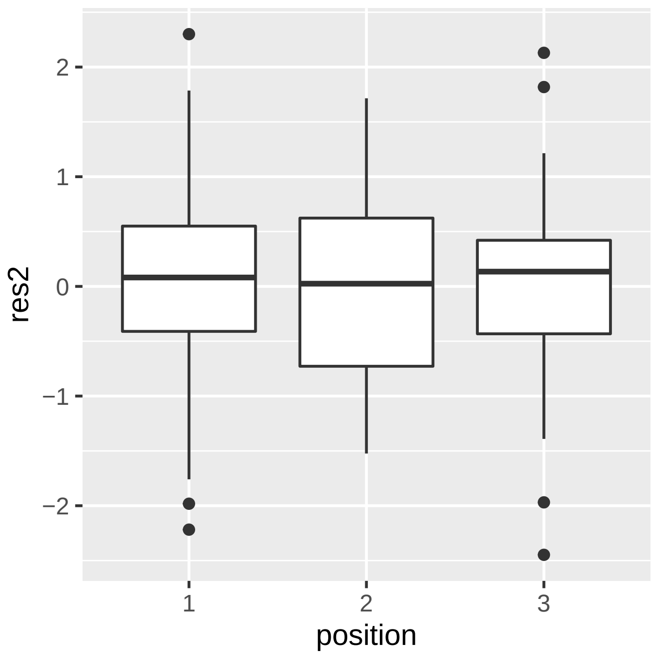


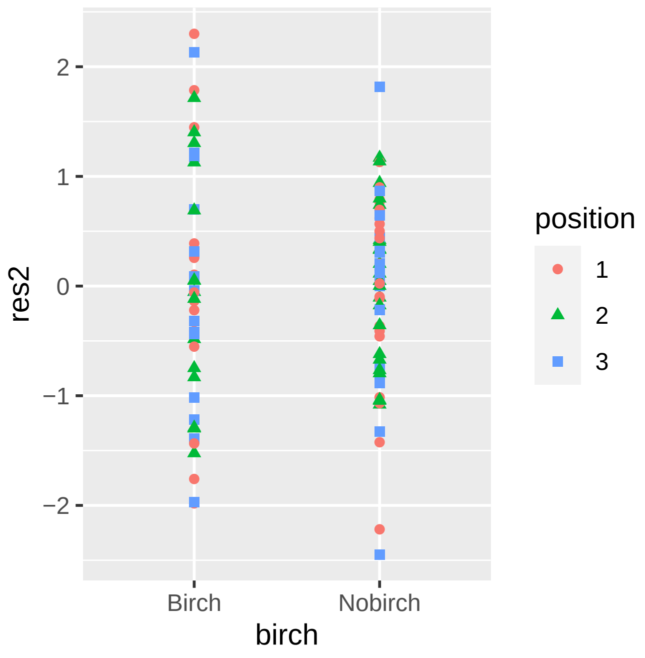
Let's make the rest of the standard residual plots we've been making, using normalized residuals. See the help page for `residuals.lme` if you need to remember the `type` options.

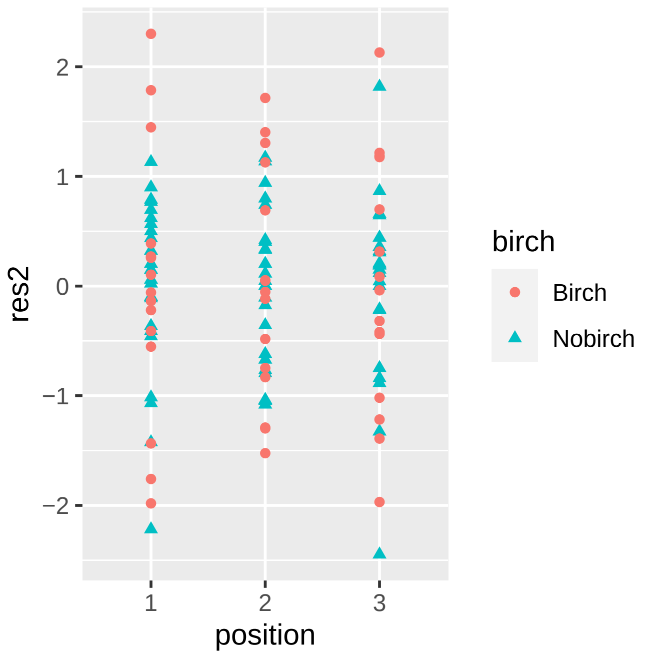
I noted no problems in these residual plots.

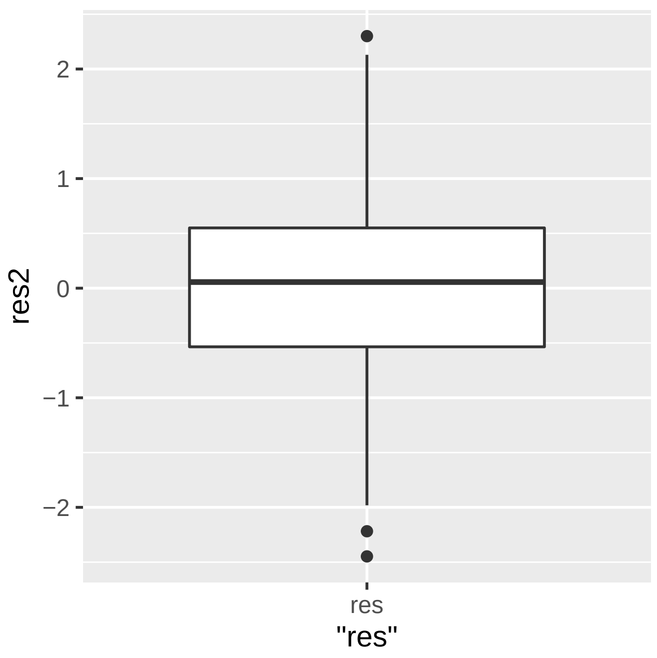
```
comp$res2 = resid(gen, type = "normalized")

# Plot residuals vs explanatory variables
# Boxplots
qplot(x = birch, y = res2,
     data = comp, geom = "boxplot")
qplot(x = position, y = res2,
     data = comp, geom = "boxplot")

# Scatterplots
qplot(x = birch, y = res2,
     color = position,
     shape = position,
     data = comp)
qplot(x = position, y = res2,
     color = birch,
     shape = birch,
     data = comp)

# Check symmetry of residuals
qplot(x = "res", y = res2,
     data = comp, geom = "boxplot")
```


res2
2
1
0
−1
−2
Birch
Nobirch
birch


res2
2
1
0
−1
−2
1
2
3
position


res2
2
1
0
−1
−2
Birch
Nobirch
birch
position
1
2
3


res2
2
1
0
−1
−2
1
2
3
position
birch
Birch
Nobirch



Model results

If everything looks good, we can use this as our final model for any inference and making estimates. If there are still problems in the residuals, we'd need to go back to the drawing board.

```
anova(gen)
```

```
               numDF denDF   F-value p-value
(Intercept)        1    72 2193.0352  <.0001
birch              1    36    1.4254  0.2403
position           2    72    1.8917  0.1582
birch:position     2    72    4.1482  0.0197
```

F-tests for unbalanced data

When we have perfect balance, as we have throughout the quarter, we can report any of the F tests without concern. However, without balance we'd need to think about the kind of tests we are doing for any main effects we want to report when an interaction is in the model. The function `anova()` for **lme** objects does *sequential* tests by default. In some cases we'd need to use *marginal* tests, such as in the case of reporting tests of main effects in the presence of an interaction when our data are unbalanced. We are not going to discuss it any more in this class, but see the discussion at https://stats.idre.ucla.edu/r/faq/how-can-i-get-type-iii-tests-of-fixed-effects-in-r/ for more background and what to do if this comes up in your own work. In particular, if you have an interaction see the section that starts with "If we have a more complex model…". Also see package **car** function `Anova()`, which offers Type III and Type II tests (I prefer the latter).

Answering the research questions

This study was set up to answer two research questions:

1. When a birch is present, is the mean height of in oak position 1 different than that of an oak in position 2 and different than that of an oak in position 3?
2. Are oaks closest to a birch different in mean height than oaks in all positions from transects with no birch present?

Two comparisons will answer the first question, and one will answer the second.

Comparisons of interest using `emmeans()`

We could get the two comparisons we need to answer the first question by getting all pairwise comparisons (with no adjustment) and pulling out only the ones we wanted. This works if not making an adjustment for multiple comparisons.

I do reverse pairwise comparisons so the position 1 group is subtracted from the others, since if competition is present then oaks in position 1 should have the lowest estimated value for mean height.

```
emmeans(gen, revpairwise ~ position|birch)$contrasts %>%
    summary( adjust = "none", infer = TRUE )
```

```
birch = Birch:
 contrast estimate    SE df lower.CL upper.CL t.ratio p.value
 2 - 1      0.7200 0.307 72    0.109    1.331   2.349  0.0216
 3 - 1      0.9533 0.287 72    0.380    1.526   3.317  0.0014
 3 - 2      0.2333 0.175 72   -0.115    0.582   1.334  0.1864

birch = Nobirch:
 contrast estimate    SE df lower.CL upper.CL t.ratio p.value
 2 - 1     -0.1783 0.248 72   -0.672    0.315  -0.720  0.4738
 3 - 1     -0.1087 0.232 72   -0.571    0.354  -0.468  0.6410
 3 - 2      0.0696 0.141 72   -0.212    0.351   0.493  0.6239

Degrees-of-freedom method: containment
Confidence level used: 0.95
```

This example is good to practice writing out the vectors for custom contrasts, since we'll need them for the second research question, anyway. So let's get the estimated marginal means for all group combinations.

```
emm_gen = emmeans(gen, ~position*birch)
emm_gen
```

```
 position birch   emmean    SE df lower.CL upper.CL
 1        Birch     3.09 0.199 37     2.69     3.50
 2        Birch     3.81 0.199 37     3.41     4.22
 3        Birch     4.05 0.199 37     3.64     4.45
 1        Nobirch   3.80 0.161 36     3.47     4.13
 2        Nobirch   3.62 0.161 36     3.30     3.95
 3        Nobirch   3.69 0.161 36     3.36     4.02

Degrees-of-freedom method: containment
Confidence level used: 0.95
```

Since the two research questions involve all six group combination means, I'll make vectors of 0 and 1 to represent each one.

```
birch1 = c(1, 0, 0, 0, 0, 0)
birch2 = c(0, 1, 0, 0, 0, 0)
birch3 = c(0, 0, 1, 0, 0, 0)
nobirch1 = c(0, 0, 0, 1, 0, 0)
nobirch2 = c(0, 0, 0, 0, 1, 0)
nobirch3 = c(0, 0, 0, 0, 0, 1)
```

Question 2 is about the heights of oak in transects without birches regardless of position, so I'll average over `position` for the `Nobirch` group.

```
nobirch = (nobirch1 + nobirch2 + nobirch3)/3
```

From last week you know that you could save the output from `contrast()` to use in `rbind()` if you want to correct for a full family of comparisons. I'm not doing a multiple comparisons correction this week, so I get the contrasts of interest, extract the CI, and put everything into a data.frame for plotting. I do make nicer names in anticipation of making plots.

```
birch_comp = contrast(emm_gen,
                  method = list("Birch Position 2 minus 1" = birch2 - birch1,
                                "Birch Position 3 minus 1" = birch3 - birch1,
                                "No Birch minus Birch Position 1" = nobirch - birch1) ) %>%
    summary( infer = c(TRUE, FALSE) )
birch_comp
```

```
 contrast                         estimate    SE df lower.CL upper.CL
 Birch Position 2 minus 1            0.720 0.307 72    0.109     1.33
 Birch Position 3 minus 1            0.953 0.287 72    0.380     1.53
 No Birch minus Birch Position 1     0.611 0.225 36    0.155     1.07

Degrees-of-freedom method: containment
Confidence level used: 0.95
```

Wrapping up the analysis

Plotting the results

Let's plot the results, which are already all together in a single data.frame.

The results plot looks similar to the other plots we've been making this quarter. Oak trees without competition should be taller than oak trees with competition effects, so the order that I did the comparisons was important for making this final graphic make sense.

```
( g1 = ggplot(birch_comp, aes(x = contrast, y = estimate) ) + # Set axes for graph
    geom_errorbar(width = .15, lwd = .75, aes(ymin = lower.CL, ymax = upper.CL) ) + # Add error bar
    geom_point(size = 3) + # Add points for means
    labs(y = "Difference in height (m)",
        x = NULL) + # Change x and y labels
    theme_bw(base_size = 16) + # change graph to black and white
    theme(panel.grid.major.x = element_blank(),
         panel.grid.minor = element_blank() ) + # remove gridlines
    geom_hline(yintercept = 0, colour = "grey34", lty = 2) + # horizontal line at 0
    geom_rect(xmin = 0, xmax = 5, ymin = -Inf, ymax  = .4, alpha = .1) +
        # difference at .4 m indicates no practical difference
    scale_y_continuous(breaks = c(seq(0, 1.5, by = .5), .4) ) ) # add 0.4 to axis
```

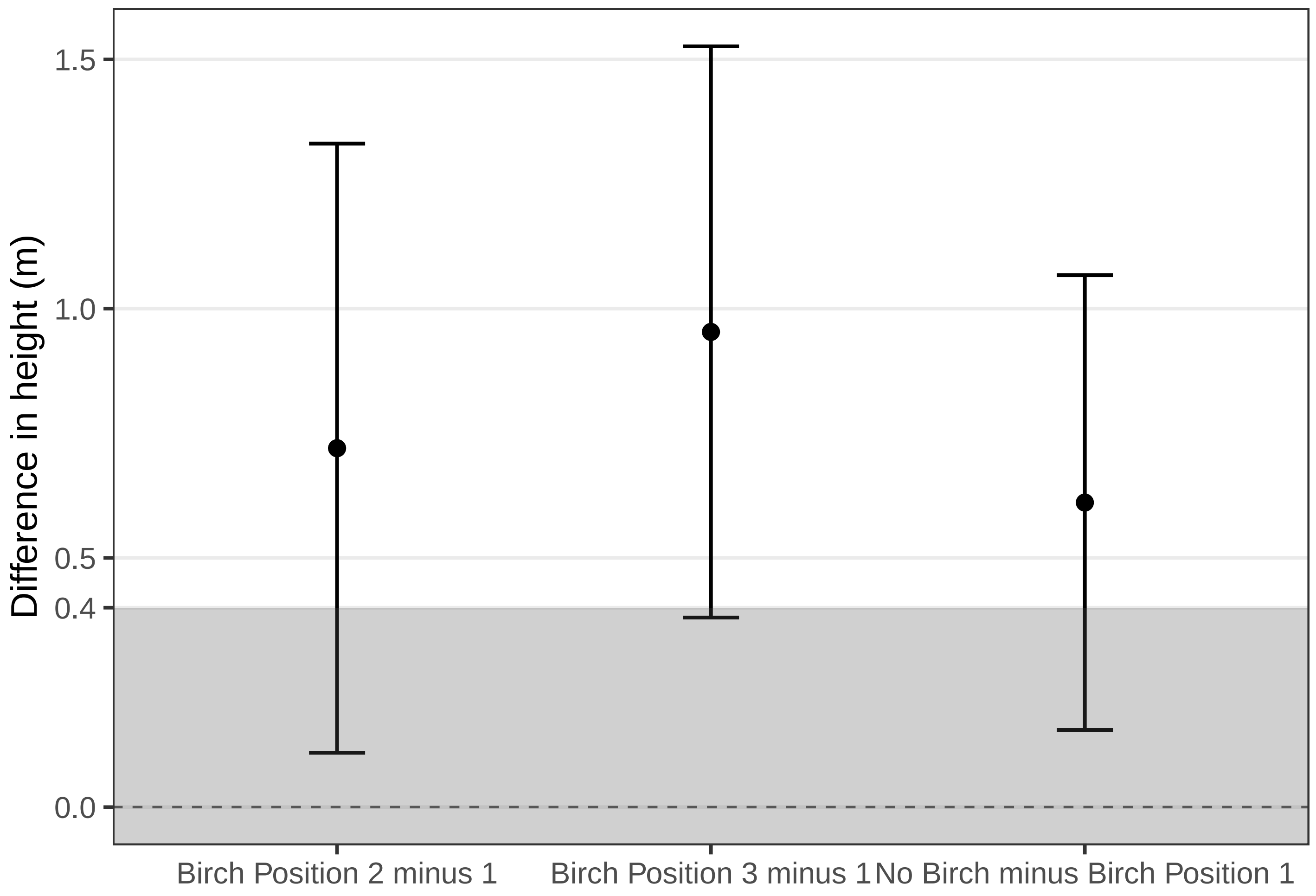


Table of summary statistics

A table of summary statistics might be useful to help the reader understand the effect of birch presence and oak tree position in transect.

```r
( sumtab = comp %>%
      group_by("Birch" = birch, "Position" = position) %>%
      summarise(n = n(),
                Mean = round(mean(ht), 1),
                SD = round(sd(ht), 1) ) )
```

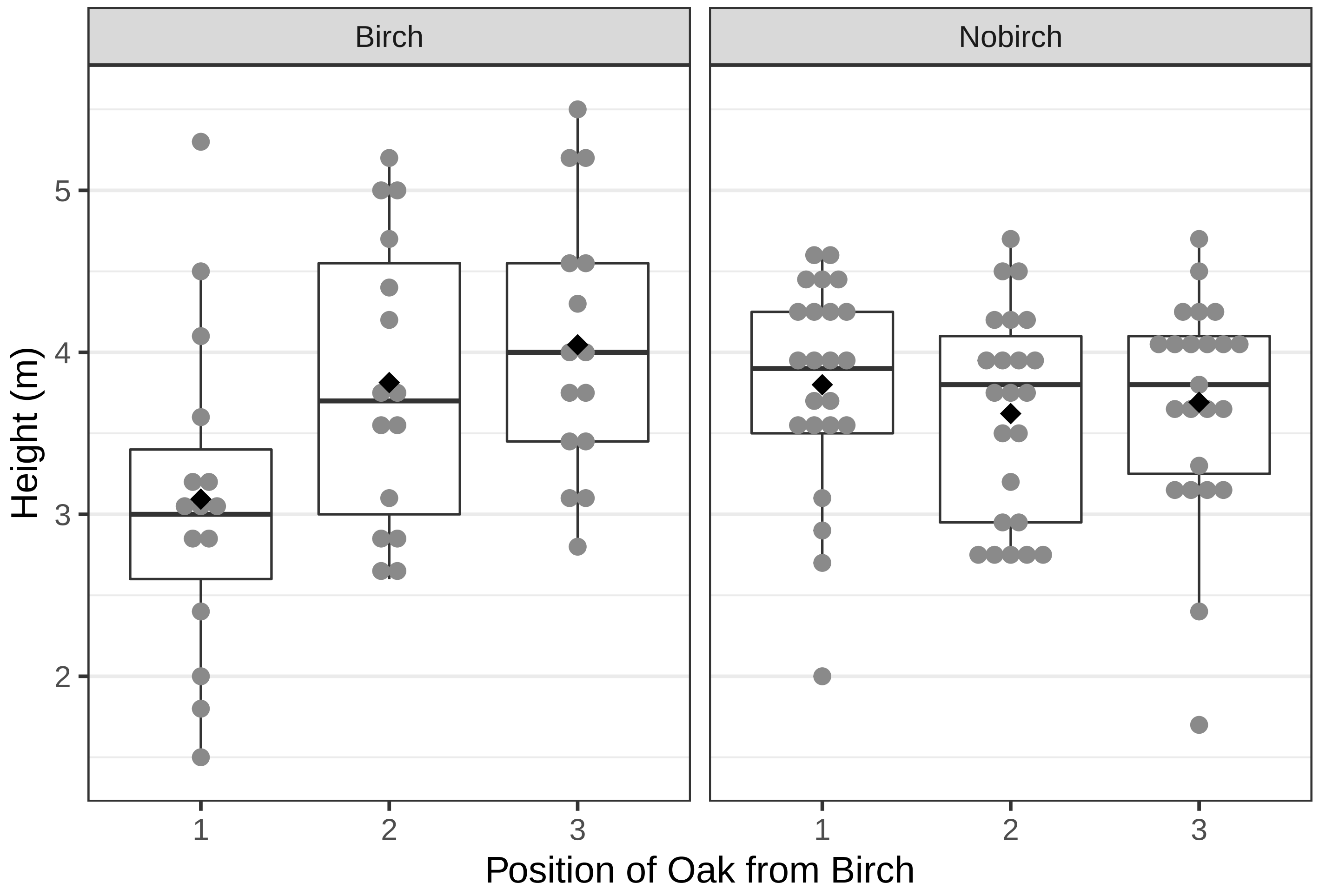
```
‘summarise()‘ has grouped output by ’Birch’. You can override using the ‘.groups‘ argument.

‘summarise()‘ has grouped output by ’Birch’. You can override using the ‘.groups‘ argument.
```

Birch	Position	n	Mean	SD
Birch	1	15	3.1	1.0
Birch	2	15	3.8	0.9
Birch	3	15	4.0	0.8
Nobirch	1	23	3.8	0.7
Nobirch	2	23	3.6	0.6
Nobirch	3	23	3.7	0.7

Plotting the raw data

Today we’ll also make a plot of the raw data much like we did in lab 2 because it really gives a good picture of what was going on in the dataset. In particular, note the variability of heights for oaks closest to the birch. This makes me wonder if the categorical `position` variable is too coarse of a measure to capture the distance from the birch. It is possible that treating distance from birch as continuous would be appropriate (if that was measured).

```
# First, change the names of the levels of the "birch" column to something nicer
# I know `Birch` comes before `Nobirch`,
    # so I'm taking a shortcut for changing levels
comp$birch2 = comp$birch
levels(comp$birch2) = c("Birch Present", "Birch Absent")

( g2 = ggplot(comp, aes(x = position, y = ht) ) + # Set x and y axes
    geom_boxplot() + # Add boxplots for each group
    geom_dotplot(binaxis = "y", stackdir = "center",
        dotsize = .75, fill = "grey54", colour = "grey54") +
        # Add a dotplot for each group, making dot size smaller
    facet_grid(~birch2) +  # now a separate graph (facet) for each birch level
    stat_summary(fun = mean, geom = "point", shape = 18, size = 4) +
        # Add means to the plot
    theme_bw(base_size = 16) +  # get rid of grey background
    theme(panel.grid.major.x = element_blank() ) + # remove x gridlines
    labs(y = "Height (m)",
        x = "Position of Oak from Birch" ) ) # change axis labels
```



R tidbit

Project-oriented workflows

For those who want to learn a little more about R, Jenny Bryan wrote up an article about working in R and how to make R scripts more portable and shareable. See the article at https://www.tidyverse.org/articles/2017/12/workflow-vs-script/.

This covers why writing out the working directories in `setwd()` makes scripts difficult to share, as others (including your future self) don't have access to your directory. I've already seen some of you using this `setwd()` approach, and right now might be a good time to change how you this before it gets to be a habit. It also discusses how to start a clean R session without code like `rm(list = ls() )`.

One way to avoid these things is to use RStudio Projects (and possibly package **here**), so ask if you want to know more about how to use these.