Longitudinal Data Analysis for the Clinical Sciences

Keith Lohse, PhD
Department of Health, Kinesiology, and Recreation
Department of Physical Therapy and Athletic Training
University of Utah

Introduction

In this document, I introduce the tools and methods for conducting longitudinal data analysis in R.

Importing data and Quality Assurance (QA)

First, we will check to which folder the working directory is set. You can think of the working directory as the folder on the computer in which R will look for files and in which R will store files that you create. Running the "get working directory" command should return the My Documents folder (or similar).

```r
getwd()
```

```
## [1] "C:/Users/u6015231/Documents"
```

Outside of R, we created a project folder called "LMER_Clinical_Science". Let's now set our working directory to that folder. The exact code will be different on your computer, but it should be something like:

```r
setwd("C:/Users/u6015231/Documents/GitHub/LMER_Clinical_Science/")
```

Now that we have set the working directory to our project folder. We can see what files are inside. We can also see what files are inside the "data" sub-folder by including the "./" characters in our code. Including the "." means to append all of the characters that follow to the address of the working directory. This can be a big time saver, because we don't want to be retyping the working directory all of the time.

```r
list.files()
```

```
## [1] "data"                              "lohse ACRM 2017 workshop.pptx"
## [3] "READ_ME.txt"                       "scripts"
```

```r
# These are all of the files/folders in the working directory.
list.files("./data/")
```

```
##  [1] "data_DIFF_UNIQUE.txt"    "data_DIFFICULTY.txt"
##  [3] "data_LOHSE_EXAMPLE.csv"  "data_LOHSE_EXAMPLE.xlsx"
##  [5] "data_UNIQUE.csv"         "individual_slopes.csv"
##  [7] "long_3_missing_data.txt" "MPLS.LS.csv"
##  [9] "MPLS.LS.Rdata"           "MPLS.Rdata"
## [11] "MPLS.W.Rdata"            "MPLScomp1.txt"
## [13] "MPLScomp2.txt"           "MPLSdata.txt"
## [15] "MPLSdata_Chapt1.csv"     "MPLSdata_Chapt1.txt"
## [17] "MPLSdata_Chapt8.txt"
```

```r
# These are all of the files in the data sub-folder of the working directory.
```

Next, we will want to open several packages whose functions we will want to use. Packages contain custom written functions that allow you perform tasks beyond the basic capabilities of R. All of the packages we will use have validated and peer-reviewed documentation, so we can be sure these functions are working correctly.

We will need to install these packages the very first time we want to use them. After a package is installed, you can import its functions into R using the library() function. Because I already have these packages installed on my machine, I have "commented out" the installation code using a "#". To install these packages, simply delete the "#" and run the code as written.

If you are new to coding, comments are an author's way of telling the computer that everything that comes after the symbol is not code to run. Thus, by using comments, you can leave useful notes to yourself and others!

```
# install.packages("ggplot2"); install.packages("lme4");
# install.packages("dplyr"); install.packages("AICcmodavg")
library("ggplot2");library("lme4");library("dplyr");library("AICcmodavg")
```

```
## Loading required package: Matrix

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

##
## Attaching package: 'AICcmodavg'

## The following object is masked from 'package:lme4':
##
##     checkConv
```

Next, we will read in a "dummy" dataset that I created. This data emulates data structures in the Brain Recovery Core Database. See Lang et al. *J Neurologic Physical Therapy.* 2011 and Lohse et al. *Arch Phys Med Rehabil.* 2016.

```
DATA<-read.csv("./data/data_LOHSE_EXAMPLE.csv", header = TRUE)
DATA[1:20,c(1,2,4,5,6,8,10,12)]
```

```
##    subID time IRF DPS Age BERG X10mSpeed ARAT
## 1    p01   -1   1  67  58    7 0.0000000   NA
## 2    p01    0   1  67  58    6 0.0000000   57
## 3    p01    1   1  67  58   21 0.0000000   58
## 4    p01    2   1  67  58   22        NA   58
## 5    p01    3   1  67  58   26 0.1294666   58
## 6    p01    4   1  67  58   23 0.1214329   58
## 7    p02   -1   0 680  50   NA        NA   NA
## 8    p02    0   0 680  50    3 0.0000000   34
## 9    p02    1   0 680  50   NA        NA   41
## 10   p02    2   0 680  50   NA        NA   44
## 11   p02    3   0 680  50    4 0.0000000   40
## 12   p02    4   0 680  50   NA        NA   45
## 13   p02    5   0 680  50    4 0.0000000   45
## 14   p02    6   0 680  50    5 0.0000000   46
## 15   p03   -1   0  55  63    3        NA   NA
## 16   p03    0   0  55  63   33 0.1954652    0
## 17   p03    1   0  55  63   38 0.2687450    0
## 18   p03    2   0  55  63   40 0.3092146    0
## 19   p03    3    0   55   63    44 0.4189359    3
## 20   p03    4    0   55   63    47 0.4878049    7
```

```
# Note that I am calling the dataframe "DATA", but you can call it anything.
# For the ease of reading, I am also only printing select columns and the
# first 20 rows of the data frame.
```

Clearly the raw data is a bit of a mess, but notice NAs in multiple rows and columns. Some participants are missing data for entire assessments, others are missing data for individual timepoints.

Even though this dataset is relatively small, it can be good for us to get a quick summary of the data by using either the head() function (which lets us look at the first six rows) or the str() function (which tells us the structure of the dataframe variable by variable).

```
# Note that now I am looking at all of the columns, but only the first six rows.
head(DATA)
```

```
##   subID time                event_name IRF DPS Age  Sex BERG BERG_ACUTE_AD
## 1   p01   -1        00_acuteadmission   1  67  58 Male    7             7
## 2   p01    0       01_admission_arm_1   1  67  58 Male    6             7
## 3   p01    1 05_onemonths_evalu_arm_1   1  67  58 Male   21             7
## 4   p01    2 06_twomonths_evalu_arm_1   1  67  58 Male   22             7
## 5   p01    3 07_threemonths_eva_arm_1   1  67  58 Male   26             7
## 6   p01    4 08_fourmonths_eval_arm_1   1  67  58 Male   23             7
##   X10mSpeed X10mWT_ACUTE_AD ARAT
## 1 0.0000000               0   NA
## 2 0.0000000               0   57
## 3 0.0000000               0   58
## 4        NA               0   58
## 5 0.1294666               0   58
## 6 0.1214329               0   58
```

```
# The structure function does not show data, but instead shows properties of
# the different columns/variables.
str(DATA)
```

```
## 'data.frame':    60 obs. of  12 variables:
##  $ subID          : Factor w/ 12 levels "p01","p02","p03",..: 1 1 1 1 1 1 2 2 2 2 ...
##  $ time           : int  -1 0 1 2 3 4 -1 0 1 2 ...
##  $ event_name     : Factor w/ 10 levels "00_acuteadmission",..: 1 2 3 4 5 7 1 2 3 4 ...
##  $ IRF            : int  1 1 1 1 1 1 0 0 0 0 ...
##  $ DPS            : int  67 67 67 67 67 67 680 680 680 680 ...
##  $ Age            : int  58 58 58 58 58 58 50 50 50 50 ...
##  $ Sex            : Factor w/ 2 levels "Female","Male": 2 2 2 2 2 2 1 1 1 1 ...
##  $ BERG           : int  7 6 21 22 26 23 NA 3 NA NA ...
##  $ BERG_ACUTE_AD  : int  7 7 7 7 7 7 NA NA NA NA ...
##  $ X10mSpeed      : num  0 0 0 NA 0.129 ...
##  $ X10mWT_ACUTE_AD: num  0 0 0 0 0 0 NA NA NA NA ...
##  $ ARAT           : int  NA 57 58 58 58 58 NA 34 41 44 ...
```

Creating Basic Plots

We are now ready to create some basic plots using the ggplot2 package. In these visualizations, we will be plotting Berg Balance Scale scores over time with a separate panel for each subject. We will get into all of the details of how to generate this kind of plot later, but in brief, this plot is composed of several different elements:

- Our x- and y-axes are always time from outpatient admission and *Berg Balance Scale* (BBS) scores, respectively.
- In each plot, we have separate data points and lines connecting them (ggplot refers to these objects as *geoms*).
- Over the top of these data points, we plot the linear model for the line of best fit.
- Finally, all of these elements are *faceted* so that each facet plots the data for a different participant.

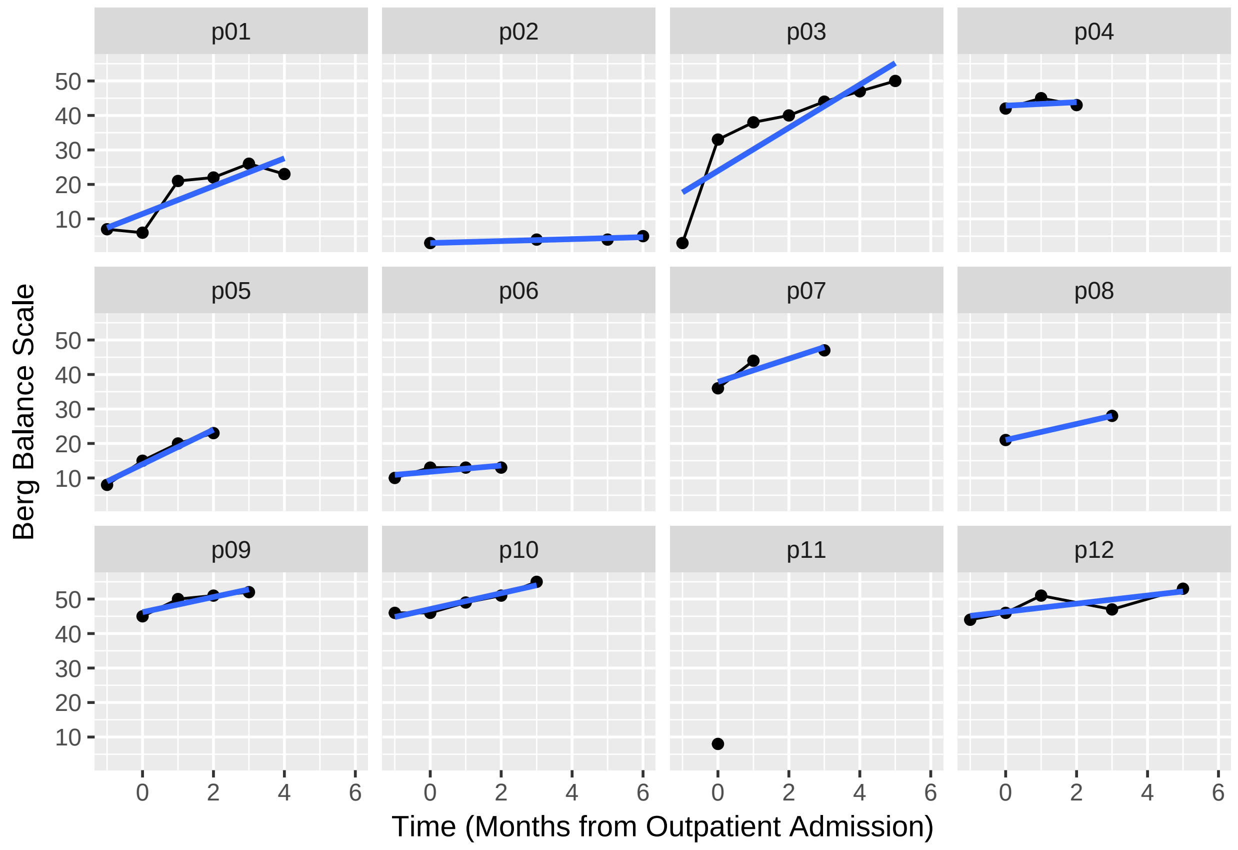
```r
myX<-scale_x_continuous(name = "Time (Months from Outpatient Admission)")
myY<-scale_y_continuous(name = "Berg Balance Scale")
g1<-ggplot(data=DATA, aes(x = time, y = BERG, group = subID))+geom_line()
g2<-g1+geom_point()+facet_wrap(~subID)+myX+myY
g3<-g2 + stat_smooth(method=lm, se=FALSE)
plot(g3)
```

```
## Warning: Removed 12 rows containing non-finite values (stat_smooth).
```

```
## Warning: Removed 6 rows containing missing values (geom_path).
```

```
## geom_path: Each group consists of only one observation. Do you need to
## adjust the group aesthetic?
```

```
## Warning: Removed 12 rows containing missing values (geom_point).
```



Note that participants have different amounts of data collected at different time points. This is one of the strengths of the linear mixed-effects regression (LMER) approach. If we were using RM ANOVA, we would need to throw out any participants with missing data! Using LMER, however, we can include participants with missing data and participants whose data were collected at different times.

These visualizations are obviously important for publications, but they also help us understand our data better throughout our analysis. As such, sometimes we will spend a lot of time creating a publication quality visualization, whereas other times we will want some quick and simple visualizations that help us understand the data better.

There are also some basic functions that help us understand our data better and get summary statistics quickly. Specifically, we will use the head(), tail(), and summary() functions.

```
head(DATA) # Peak at the top rows.
```

```
##   subID time                event_name IRF DPS Age  Sex BERG BERG_ACUTE_AD
## 1   p01   -1        00_acuteadmission   1  67  58 Male    7             7
## 2   p01    0        01_admission_arm_1   1  67  58 Male    6             7
## 3   p01    1 05_onemonths_evalu_arm_1   1  67  58 Male   21             7
## 4   p01    2 06_twomonths_evalu_arm_1   1  67  58 Male   22             7
## 5   p01    3 07_threemonths_eva_arm_1   1  67  58 Male   26             7
## 6   p01    4 08_fourmonths_eval_arm_1   1  67  58 Male   23             7
##   X10mSpeed X10mWT_ACUTE_AD ARAT
## 1 0.0000000               0   NA
## 2 0.0000000               0   57
## 3 0.0000000               0   58
## 4        NA               0   58
## 5 0.1294666               0   58
## 6 0.1214329               0   58
```

```
tail(DATA) # Peak at the bottom rows.
```

```
##    subID time                  event_name IRF DPS Age    Sex BERG
## 55   p11    0         01_admission_arm_1   1 334  65   Male    8
## 56   p12   -1          00_acuteadmission   0  10  71 Female   44
## 57   p12    0         01_admission_arm_1   0  10  71 Female   46
## 58   p12    1   05_onemonths_evalu_arm_1   0  10  71 Female   51
## 59   p12    3 07_threemonths_evalu_arm_1   0  10  71 Female   47
## 60   p12    5    09_fivemonths_eva_arm_1   0  10  71 Female   53
##    BERG_ACUTE_AD X10mSpeed X10mWT_ACUTE_AD ARAT
## 55             8     0.120            0.12   45
## 56            44     0.110            0.11   NA
## 57            44     0.610            0.11   57
## 58            44     0.840            0.11   58
## 59            44     0.885            0.11   58
## 60            44     0.795            0.11   58
```

```
summary(DATA$time) # Using the summary function on a numeric variable
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  -1.000   0.000   1.000   1.383   3.000   6.000
```

```
summary(DATA$event_name) # Using the summary function on a factor variable
```

```
##          00_acuteadmission         01_admission_arm_1
##                         11                         12
##   05_onemonths_evalu_arm_1   06_twomonths_evalu_arm_1
##                         13                         10
##   07_threemonths_eva_arm_1 07_threemonths_evalu_arm_1
##                          5                          1
##   08_fourmonths_eval_arm_1    09_fivemonths_eva_arm_1
##                          3                          1
##   09_fivemonths_eval_arm_1   10_sixmonths_evalu_arm_1
##                          2                          2
```

Creating Tidy Data

The goal of *tidy* data is to structure our data in such a way that the data are easy for a computer to work with. In general, this means that our data are in "long" format, where every row is a separate observation and every column is a unique dependent variable (see Wickham, *J Stat Soft*, 2014). Our practice dataset already adheres to these principles. For your own data (or other data you might encounter "in the wild"), you will want to start collecting data in this long format or learn how to transform data from wide to long format. (Note that long format is also often referred to as "person-period" format because each row is a unique time-point for each person, whereas wide format is also referred to as "person" format because each row is a unique person, but the time-points are in different columns.)

Next, we are going to filter out data to remove some unwanted observations. Specifically, we can create a new dataset that removes the row for the acute admission time-point from each participant. We are doing this because we already have separate columns in our dataset that give the acute admission scores. These variables are *static* variables, meaning that they do not change overtime for our individuals. All acute admission time points are coded as -1 in the *time* variable, so we can remove those data points like this:

```
DAT2<-subset(DATA, time != -1)
head(DAT2)
```

```
##   subID time                event_name IRF DPS Age    Sex BERG
## 2   p01    0        01_admission_arm_1   1  67  58   Male    6
## 3   p01    1 05_onemonths_evalu_arm_1   1  67  58   Male   21
## 4   p01    2 06_twomonths_evalu_arm_1   1  67  58   Male   22
## 5   p01    3 07_threemonths_eva_arm_1   1  67  58   Male   26
## 6   p01    4 08_fourmonths_eval_arm_1   1  67  58   Male   23
## 8   p02    0        01_admission_arm_1   0 680  50 Female    3
##   BERG_ACUTE_AD X10mSpeed X10mWT_ACUTE_AD ARAT
## 2             7 0.0000000               0   57
## 3             7 0.0000000               0   58
## 4             7        NA               0   58
## 5             7 0.1294666               0   58
## 6             7 0.1214329               0   58
## 8            NA 0.0000000              NA   34
```

Note that in our new dataset (*DAT2*) the time variable only ranges from 0 to 6 months.

```
summary(as.factor(DAT2$time))
```

```
##  0  1  2  3  4  5  6
## 12 11 10  8  3  3  2
```

```
# We are using the as.factor() funtion to treat time as a factor even though it
# is numeric. As such, the summary function returns the different categories of
# time (top row) and the number of observations in each category (bottom row).
```

Visualizing Longitudinal Data

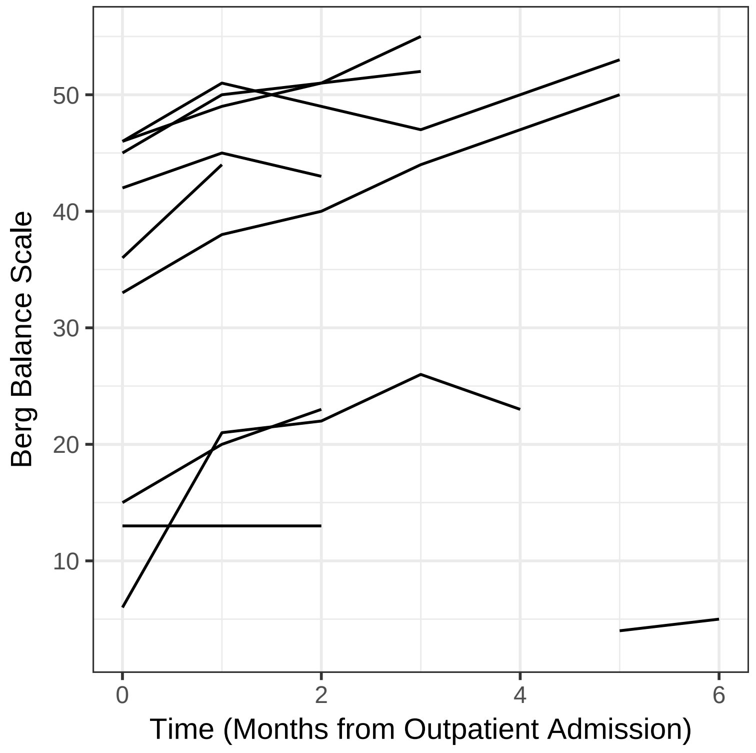
Spaghetti Plots

Longitudinal data analysis is all about seeing how our data change over time. As such, most of the time we will be plotting our dependent variable on the y-axis and time on the x-axis and using different symbols/colors to denote different groups of participants. One of the most common plots for achieving this function is the *spaghetti plot* which plots the dependent variable over time with a separate line for each participant (in a shape vaguely reminiscent of spaghetti!). We will create these plots for all of our dependent variables: the *Berg Balance Scale* (BBS), the *10-meter Walk Test* (10mWT), and the *Action Research Arm Test* (ARAT).

Berg Balance Scale

```r
myX<-scale_x_continuous(name = "Time (Months from Outpatient Admission)")
myY<-scale_y_continuous(name = "Berg Balance Scale")
g1<-ggplot(data=DAT2, aes(x=time, y = BERG, group=subID))+geom_line()+myX+myY
g2<-g1+theme_bw()
print(g2)
```

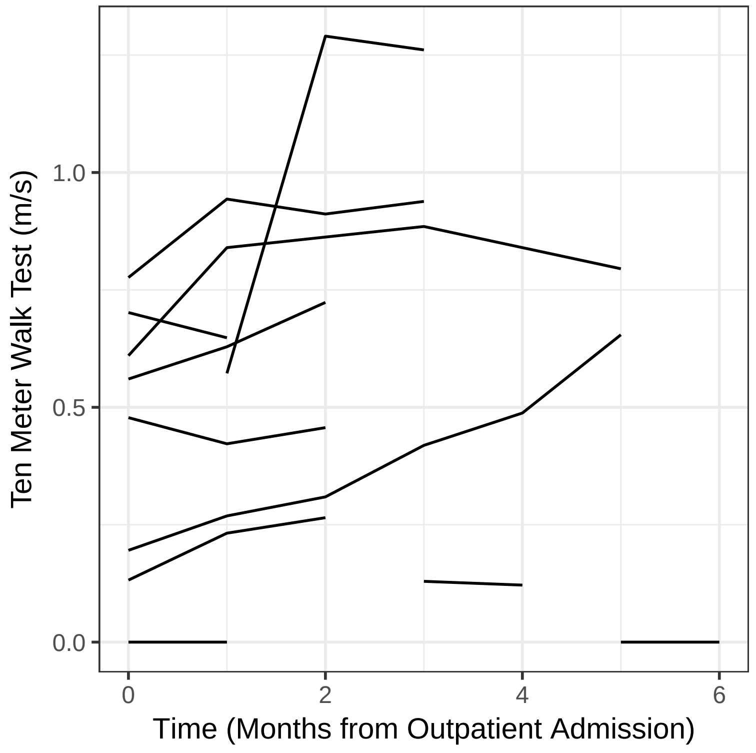
```
## Warning: Removed 1 rows containing missing values (geom_path).
```



10 Meter Walk Test

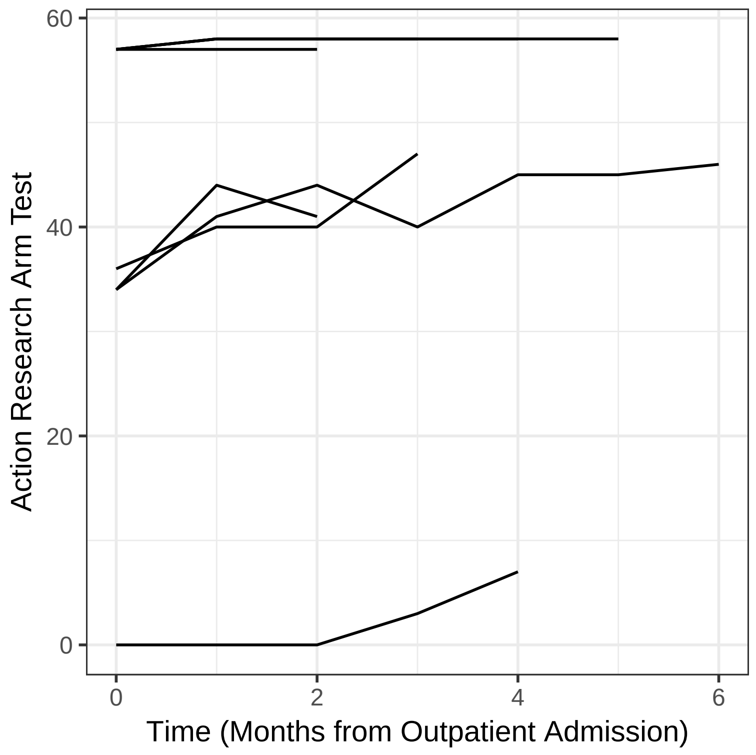
```
myX<-scale_x_continuous(name = "Time (Months from Outpatient Admission)")
myY<-scale_y_continuous(name = "Ten Meter Walk Test (m/s)")
g1<-ggplot(data=DAT2, aes(x=time, y = X10mSpeed, group=subID))+geom_line()+myX+myY
g2<-g1+theme_bw()
print(g2)
```

```
## Warning: Removed 2 rows containing missing values (geom_path).
```



Action Research Arm Test

```r
myX<-scale_x_continuous(name = "Time (Months from Outpatient Admission)")
myY<-scale_y_continuous(name = "Action Research Arm Test")
g1<-ggplot(data=DAT2, aes(x=time, y = ARAT, group=subID))+geom_line()+myX+myY
g2<-g1+theme_bw()
print(g2)
```



Faceted/Lattice Plots

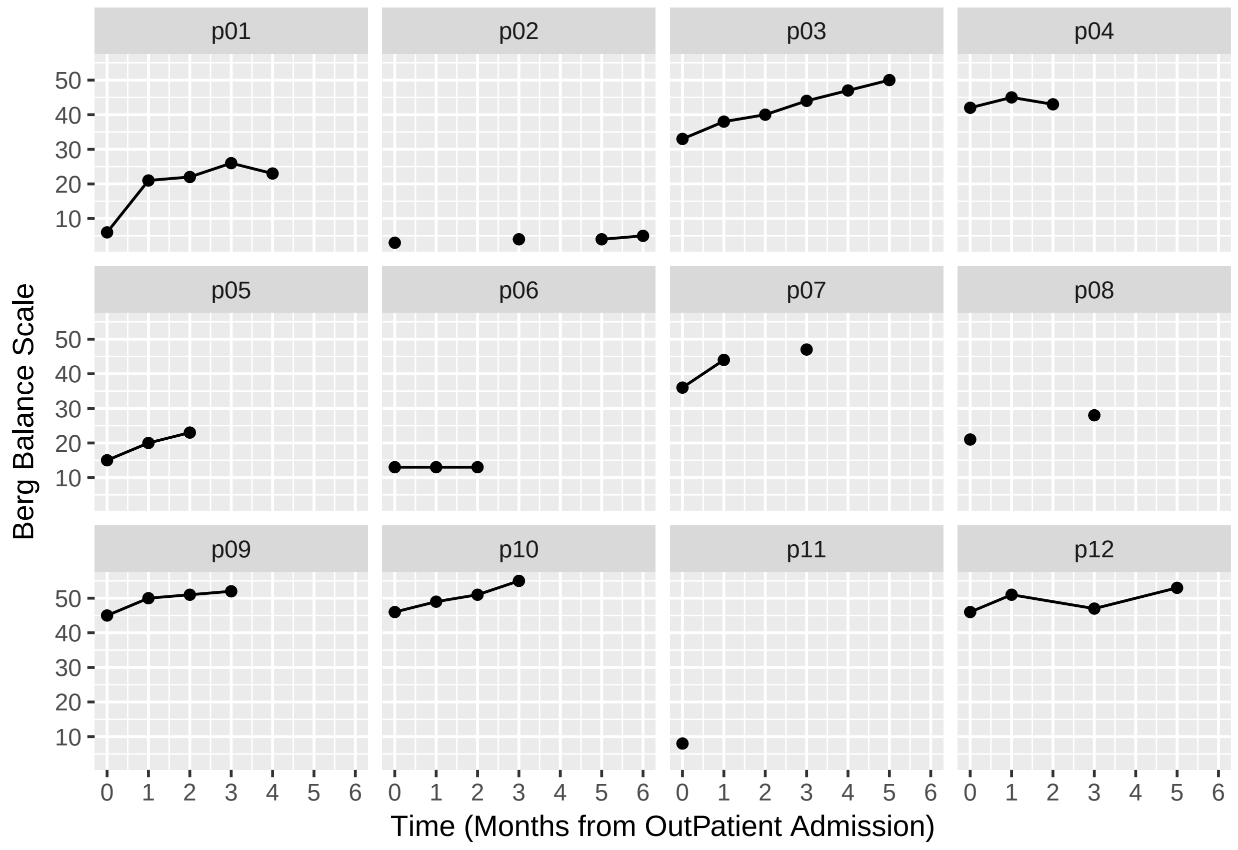
Similar to the spaghetti plot, we can create *faceted* plots (also known as *lattice* plots) in which different participants or groups are shown in different facets of the same plot. Grouping our data into facets can be a very effective way to visualize data, especially when our datasets are large (which can make spaghetti plots very hard to interpret). In the plots below, we will create facets for individual participants. These plots can be great, because they show individual participant data very clearly… but these plots are also limited because it can be difficult to meaningfully plot data from a lot of participants at once. As such, in large datasets, we will sometimes select a subset of participants and then create a facet plot for that subset.

Berg Balance Scale

```r
myX<-scale_x_continuous(breaks = 0:12,
                        name = "Time (Months from OutPatient Admission)")
myY<-scale_y_continuous(name = "Berg Balance Scale")
g5<-ggplot(data=DAT2, aes(x = time, y = BERG, group = subID))+geom_line()
g6<-g5+geom_point()+facet_wrap(~subID)+myX+myY
plot(g6)
```

```
## Warning: Removed 1 rows containing missing values (geom_path).
```

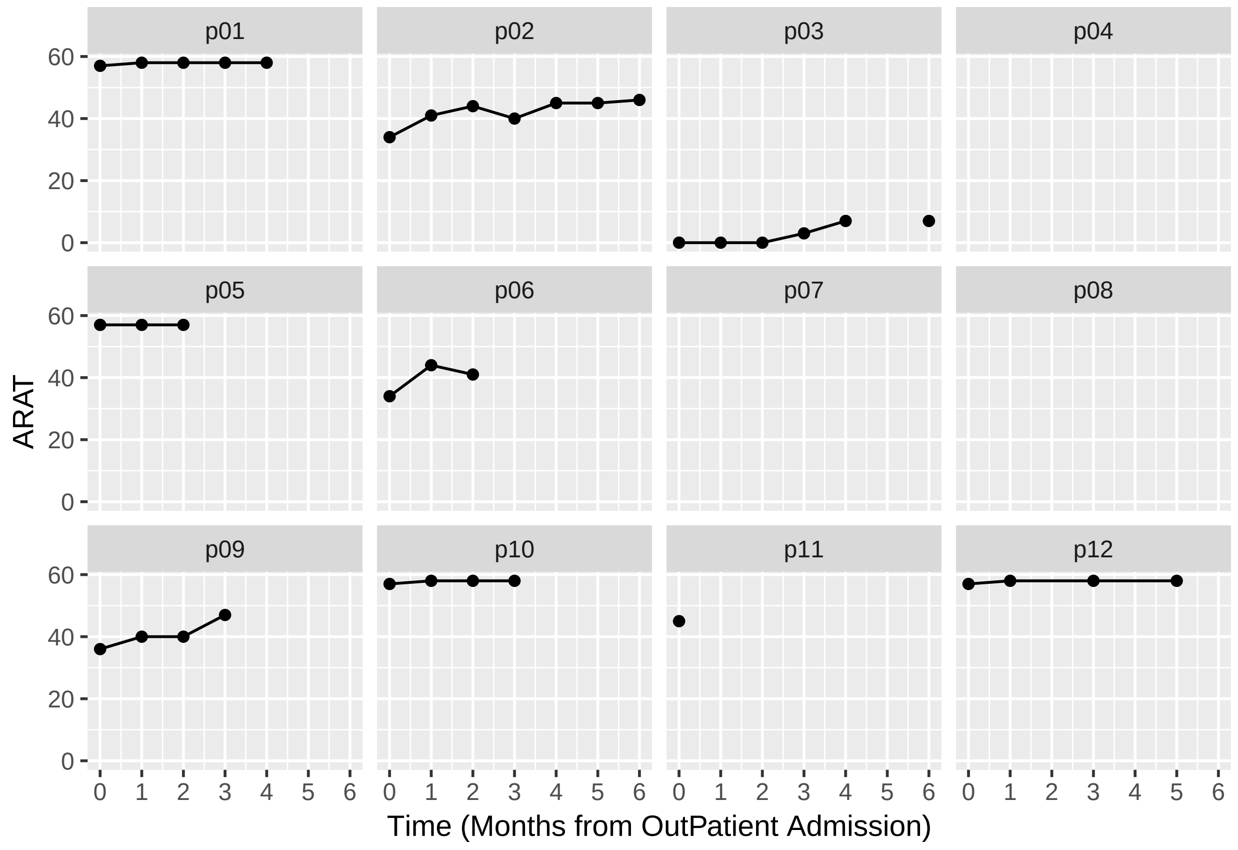
```
## Warning: Removed 7 rows containing missing values (geom_point).
```



Action Research Arm Test

```
myX<-scale_x_continuous(breaks = 0:12,
                        name = "Time (Months from OutPatient Admission)")
myY<-scale_y_continuous(name = "ARAT")
g1<-ggplot(data=DAT2, aes(x = time, y = ARAT, group = subID))+geom_line()
g2<-g1+geom_point()+facet_wrap(~subID)+myX+myY
plot(g2)
```

```
## Warning: Removed 12 rows containing missing values (geom_point).
```

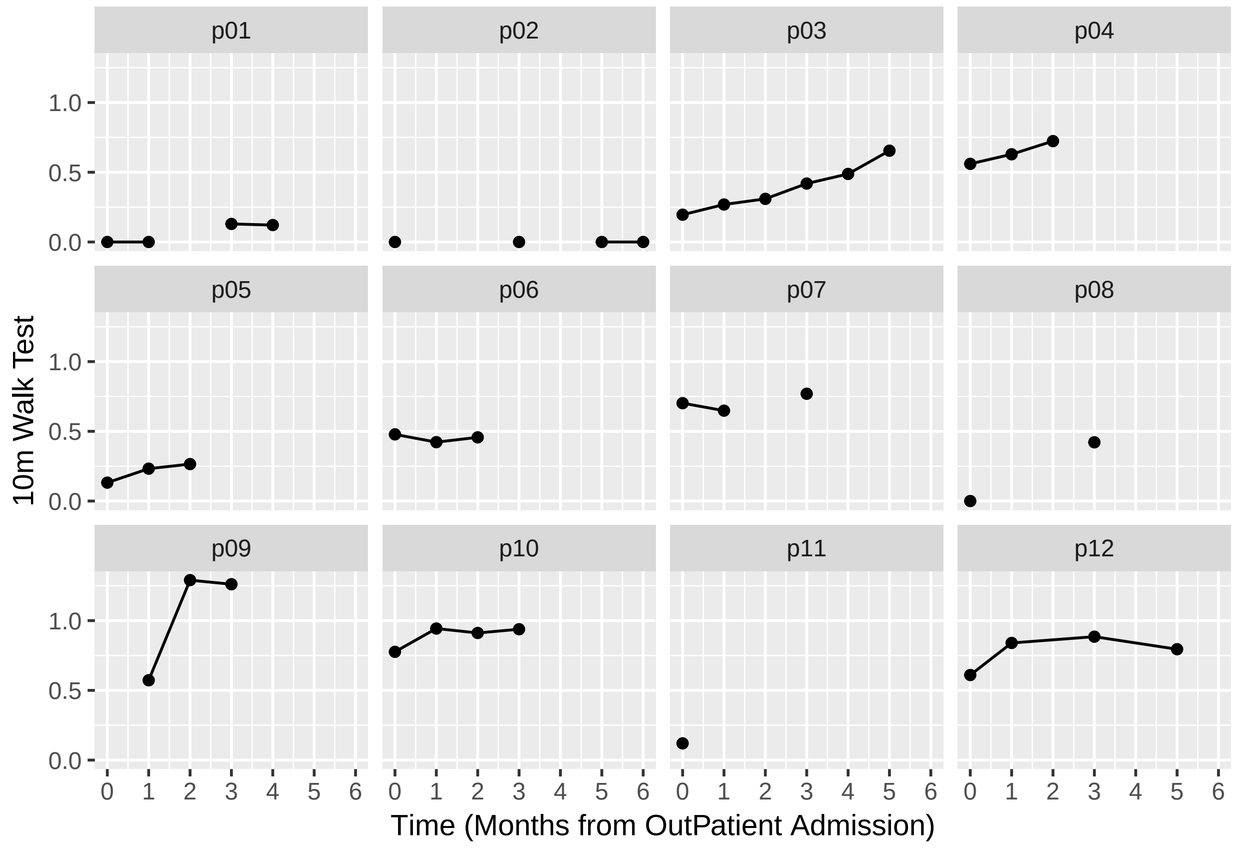


10 Meter Walk Test

```
myX<-scale_x_continuous(breaks = 0:12,
                        name = "Time (Months from OutPatient Admission)")
myY<-scale_y_continuous(name = "10m Walk Test")
g7<-ggplot(data=DAT2, aes(x = time, y = X10mSpeed, group = subID))+geom_line()
g8<-g7+geom_point()+facet_wrap(~subID)+myX+myY
plot(g8)
```

```
## Warning: Removed 2 rows containing missing values (geom_path).
```

```
## Warning: Removed 9 rows containing missing values (geom_point).
```



Conditional Plots

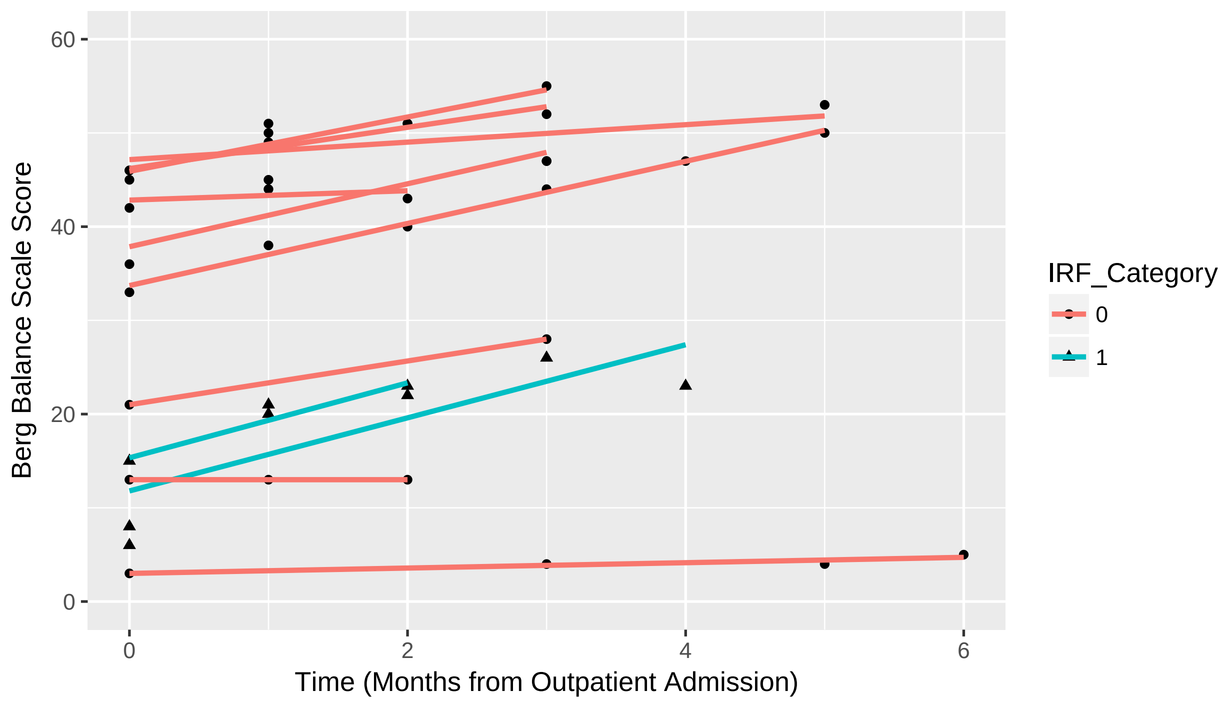
We can also make aspects of our plots conditional on some properties of the data. For instance, we can make the shape of our data-points conditional on whether or not a participant went to an inpatent rehabilitation facility (IRF).

In this case, we can do this by adding an additional “shape” argument to our graphing code.

```
# First we will create a factor out of the IRF variable (which is coded as
# 1s and 0s).
DAT2$IRF_Category<-factor(DAT2$IRF)
# Note the the additional shape argument in aes()
myX<-scale_x_continuous(name = "Time (Months from Outpatient Admission)")
myY<-scale_y_continuous(name = "Berg Balance Scale Score", limits=c(0,60))
g1 <- ggplot(data = DAT2, aes(x = time, y = BERG, group=subID,
                              shape = IRF_Category)) + geom_point()
# We now specify that the linear fit for each participant is conditional on
# on whether or not that participant went to an IRF previously.
g2 <- g1 + geom_smooth(method=lm, se=FALSE, aes(color=IRF_Category)) + myX + myY
g3 <- g2 + theme_bw()
print(g2)
```

```
## Warning: Removed 7 rows containing non-finite values (stat_smooth).

## Warning: Removed 7 rows containing missing values (geom_point).
```



Building Statistical Models

In this section, we will start building statistical models using the BBS data as our example. The best way to think about these statistical models is as a way of formalizing what we see in the visualizations. Ideally, statistical output should be confirming patterns that we can see in the figures! But first, a word about the technical details of longitudinal data analysis…

Comparing Linear Mixed-Effect Regression to Traditional Linear Models

In a traditional *general linear models* (GLM), all of our data are independent (i.e., one data point per person). Statistically, we can write this as a linear model like:

$$y_i = B_0 + B_1(TIME_i) + \epsilon_i$$

Each subject's actual score (y_i) is the result of an intercept (B_0) and that constant is modified based on their grade (the slope, B_1 multiplied by their grade). The intercept and slope are collectively referred to as our statistical *MODEL*. Our model is not going to be perfect, however, so we need to include an error term (ϵ_i). Good models will have small errors and thus be a better approximation of our *DATA*. As such, we can more generally say that:

$$DATA = MODEL + ERROR$$

The code for LMER is essentially the same as GLM, except that we will be using lmer() instead of lm(). (Note that you need lme4 installed to have access to the lmer() function.) Conceptually, LMER is a lot like GLM but we need to 'partition' our variance into different sources.

$$y_{ij} = B_0 + U_{0j} + B_1(TIME_{ij}) + U_{1j} + \epsilon_{ij}$$

In LMER, we now have data indexed by time point (i) and by participant (j). Each datapoint, y_{ij}, can still be described by the overall intercept and slope, B_0 and B_1, plus a random effect for each subject, U_{0j} and U_{1j}.

Note that these random-effects could be positive or negative, because they represent how this participant deviates from the norm. Thus, in LMER our *MODEL* is the combination of our fixed-effects (all of the B's) and the random-effects (all of the U_j's). However, *DATA = MODEL + ERROR* still applies, so we need to include a random-error term for each data point, ϵ_{ij}.

In summary, we have the following terms in our DATA:

- The *MODEL* includes fixed effects and random effects.
- *Fixed-Effects* are the group-level B's, these effects parallel the traditional main-effects and interactions that you have probably encountered in other statistical analyses.
- *Random-Effects* are the participant-level U_j's that remove statistical dependency from our data. (This is bit of a simplification, but you can think of not including the appropriate random-effects like running a between-subjects ANOVA when you should be running a repeated-measures ANOVA.)
- The *ERRORS*, or more specifically *Random Errors*, are the difference between our *MODEL*'s predictions and the actual *DATA*.

We can write this as:

$$y_{ij} = B_0 + U_{0j} + (B_1 + U_{1j}) * (TIME_{ij}) + E_{ij}$$

Or in the equivalent form:

$$y_{ij} = B_0 + B_1(TIME_i) + (U_0 + U_1(TIME_i)|Subject)$$

This second form more closely resembles the R syntax. Note that we are not including the j subscript here anymore, but it is still implied by the model because we are saying that there are different random-effects (U_0 and U_1) for a given subject.

Building Linear Mixed-Effect Models

The "Random Intercepts" Model

In our Random Intercepts model, we estimate a constant (intercept) for each participant and the overall constant. This overall constant is the *fixed-effect*, and individual deviations away from this constant are our *random-effects*. (In writing up a study, we would refer to this as a random-effect of subject/participant.)

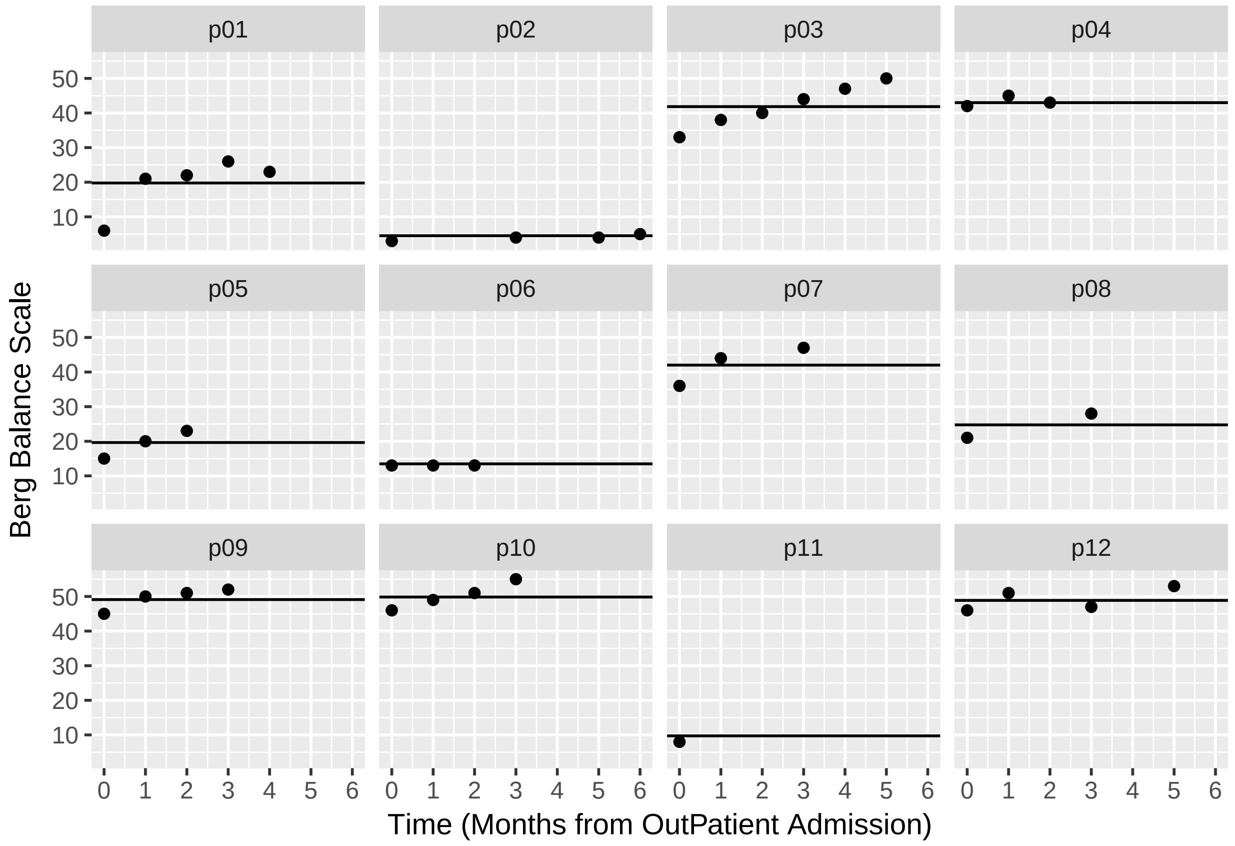
$$y_{ij} = B_0 + U_{0j} + \epsilon_{ij}$$

True to it's name, the Random Intercepts model is going to estimate a flat line for each person. Each line will be different, however, because our random-effect of subject means that we are estimating a unique intercept for each person. This intercept will be equal to mean for each participant, because the mean is the value that will produce the smallest errors.

```
B0<-lmer(BERG~1+(1|subID),data=DAT2, REML=FALSE)
summary(B0)
```

```
## Linear mixed model fit by maximum likelihood  ['lmerMod']
## Formula: BERG ~ 1 + (1 | subID)
##    Data: DAT2
##
##      AIC      BIC   logLik deviance df.resid
##    300.0    305.2   -147.0    294.0       39
##
## Scaled residuals:
##      Min       1Q   Median       3Q      Max
## -2.90995 -0.38353  0.03968  0.46555  1.72257
##
## Random effects:
##  Groups   Name        Variance Std.Dev.
##  subID    (Intercept) 268.91   16.398
##  Residual              22.42    4.735
## Number of obs: 42, groups:  subID, 12
##
## Fixed effects:
##             Estimate Std. Error t value
## (Intercept)   30.549      4.802   6.362
```

Plot of Random-Intercepts Model for the Berg Balance Scale



Note that in the plot above, each dot represents the actual data for each participant. The lines in each panel, conversely, represent the model's predictions for each participant. Because *B0* is the random-intercepts model, our model is predicting a different flat line for each participant.

The "Random Intercept - Fixed Slope" Model

In our Random Intercept - Fixed Slope model, we estimate an intercept for each participant and an overall slope, but **the slope is the same for each person**. The overall intercepts and slopes are the fixed-effects. The prediction lines for each participant can have different heights (due to the random-intercepts), but all of these lines have the same slope (because there is no random-effect for the slope).

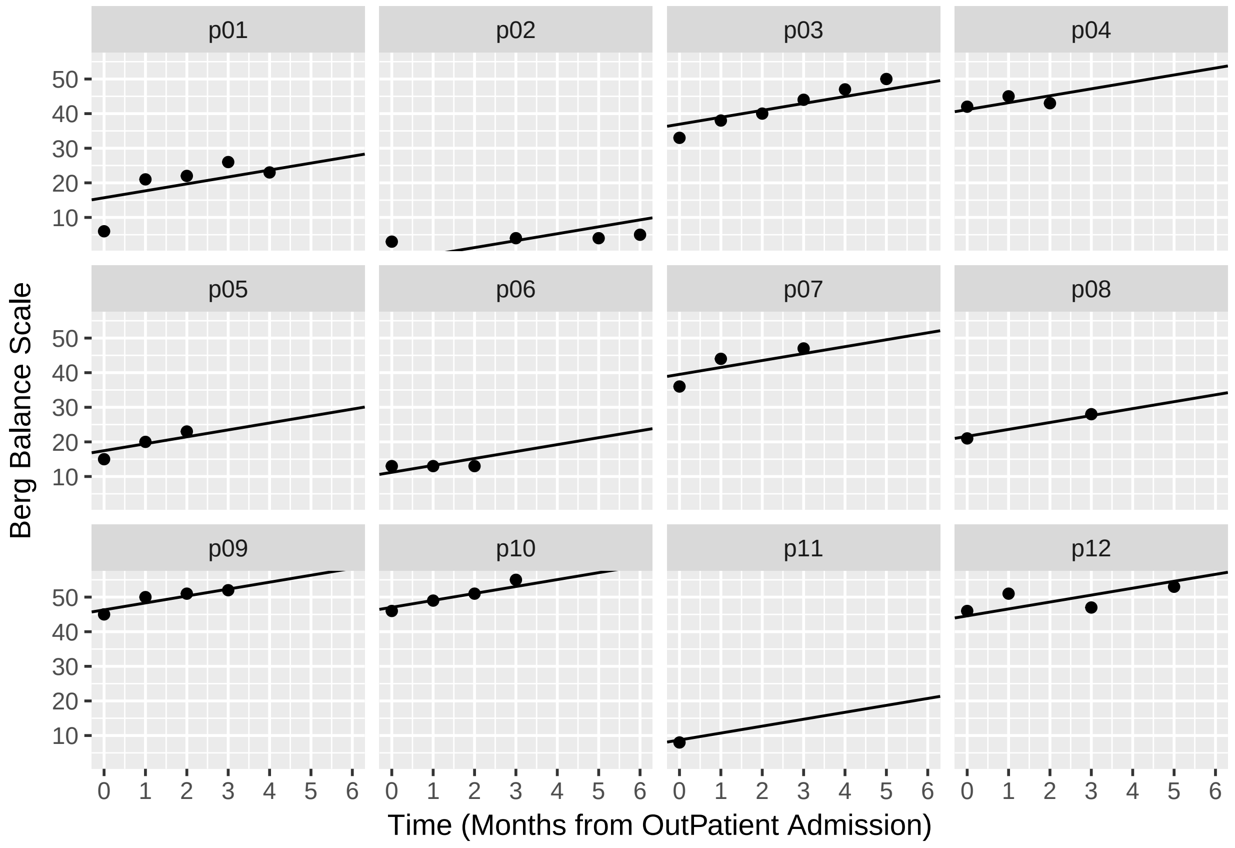
$$y_{ij} = B_0 + U_{0j} + B_1 * (TIME_{ij}) + \epsilon_{ij}$$

```
B1<-lmer(BERG~1+time+(1|subID),data=DAT2, REML=FALSE)
summary(B1)
```

```
## Linear mixed model fit by maximum likelihood  ['lmerMod']
## Formula: BERG ~ 1 + time + (1 | subID)
##    Data: DAT2
##
##      AIC      BIC   logLik deviance df.resid
##    279.4    286.4   -135.7    271.4       38
##
## Scaled residuals:
##      Min       1Q   Median       3Q      Max
## -2.98346 -0.38528  0.05064  0.54454  1.76134
##
## Random effects:
##  Groups   Name        Variance Std.Dev.
##  subID    (Intercept) 274.81   16.577
##  Residual              10.54    3.246
## Number of obs: 42, groups:  subID, 12
##
## Fixed effects:
##             Estimate Std. Error t value
## (Intercept)  27.2915     4.8486   5.629
## time          2.0017     0.3446   5.808
##
## Correlation of Fixed Effects:
##      (Intr)
## time -0.113
```

Plot of Random-Intercepts Fixed Slope Model for the Berg Balance Scale

```
##     subID Intercepts Slopes
## p01   p01  15.685644 2.0017
## p02   p02  -2.718155 2.0017
## p03   p03  36.934149 2.0017
## p04   p04  41.154404 2.0017
## p05   p05  17.457350 2.0017
## p06   p06  11.203960 2.0017
## p07   p07  39.508225 2.0017
## p08   p08  21.606475 2.0017
## p09   p09  46.315060 2.0017
## p10   p10  47.057937 2.0017
## p11   p11   8.712515 2.0017
## p12   p12  44.580436 2.0017
```



Note that in the plot above, each dot represents the actual data for each participant. The lines in each panel, conversely, represent the model's predictions for each participant. Because *B1* is the random-intercepts fixed slope model, our model is predicting a different intercept for each participant, but each participant has the same slope.

The “Random Slopes” Model

In our Random Slopes model, we estimate an overall intercept and slope and a unique intercept and slope for each participant. These overall intercepts and slopes are the fixed-effects (B’s), and deviations away from these values are random-effects ($B+U$’s).

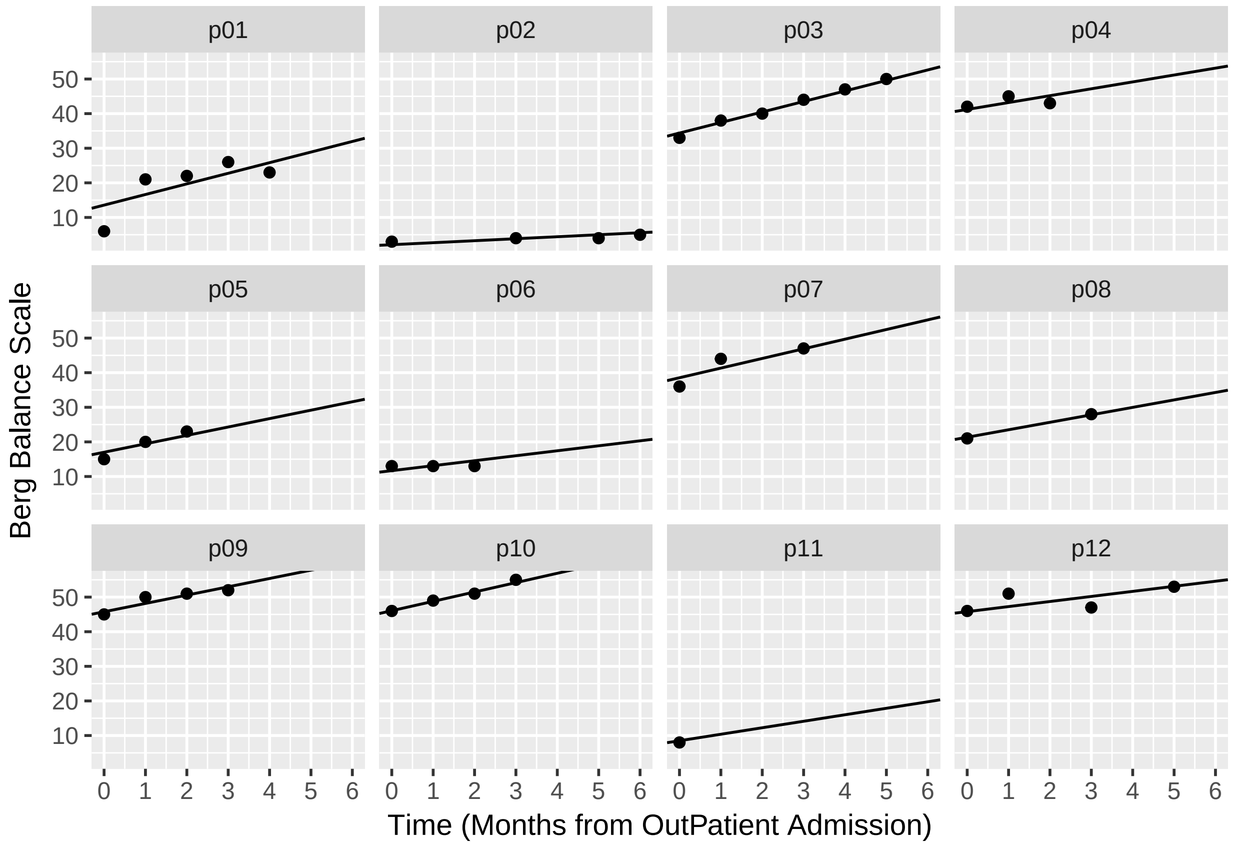
$$y_{ij} = B_0 + U_{0j} + (B_1 + U_{1j}) * (TIME_{ij}) + \epsilon_{ij}$$

```
B2<-lmer(BERG~1+time+(1+time|subID),data=DAT2, REML=FALSE)
summary(B2)
```

```
## Linear mixed model fit by maximum likelihood  ['lmerMod']
## Formula: BERG ~ 1 + time + (1 + time | subID)
##    Data: DAT2
##
##      AIC      BIC   logLik deviance df.resid
##    276.7    287.2   -132.4    264.7       36
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -2.9245 -0.2786  0.0650  0.3125  1.6897
##
## Random effects:
##  Groups   Name        Variance Std.Dev. Corr
##  subID    (Intercept) 250.714  15.834
##           time          1.065   1.032   0.23
##  Residual               6.690   2.586
## Number of obs: 42, groups:  subID, 12
##
## Fixed effects:
##             Estimate Std. Error t value
## (Intercept)  27.1662     4.6148   5.887
## time          2.1606     0.4481   4.821
##
## Correlation of Fixed Effects:
##      (Intr)
## time 0.085
```

Plot of Random Slopes Model for the Berg Balance Scale

```
##     subID Intercepts      time
## p01   p01  13.564121 3.0654644
## p02   p02   2.122334 0.5754571
## p03   p03  34.383588 3.0388763
## p04   p04  41.206360 1.9889862
## p05   p05  17.007389 2.4305898
## p06   p06  11.683921 1.4372448
## p07   p07  38.532319 2.7863482
## p08   p08  21.348696 2.1553786
## p09   p09  45.768934 2.4039091
## p10   p10  46.073869 2.7042603
## p11   p11   8.498068 1.8758309
## p12   p12  45.804802 1.4648542
```



Note that in the plot above, each dot represents the actual data for each participant. The lines in each panel, conversely, represent the model's predictions for each participant. Because *B2* is the random-slopes model, our model predicts different intercepts and slopes for each participant.

Conditional Models

These three models (Random Intercepts, Fixed Slopes, and Random Slopes) are our starting models that you will probably end up building in most LMER analyses that you run. These models contain valuable information about the variation within and between participants and essentially become the "models to beat" as you add other variables to your model. For instance, the Random Slopes model accounts for how participants change over time but I might be interested in much more than that. (However, sometimes we might also be interested in testing nonlinear effects of time and random slopes are not enough! You definitely can model time non-linearly, but it is an advanced topic that we will not discuss here.) Below are several types on hypotheses you might be interested in testing in your models:

- Do participants who went to an IRF start with worse BBS scores at admission than participants who did not go to an IRF? (The hypothesis asks about the effects of a *categorical* variable on the *intercepts*.)
- Do participants who went to an IRF show more rapid improvement in BBS scores over time compared to participants who did not go to an IRF? (The hypothesis asks about the effects of a *categorical* variable on the *slopes*.)
- Do older participants show show more rapid improvement in BBS scores over time compared to younger participants? (The hypothesis asks about the effects of a *continuous* variable on the *slopes*.)

And naturally there are many more hypotheses that you might be interested in testing! In general, however, when we are talking about data with two-levels (i.e., time nested within participants) we are usually interested in the effects that other variables have on the intercepts (i.e., does this variable affect where someone starts) and on the slopes (i.e., does this variable affect how someone changes over time). In the code below, we will walk through how to build these models using an exampel of a categorical predictor (IRF stay or not) and a continous predictor (days post-stroke or DPS).

Does IRF affect the intercept?

$$y_{ij} = B_0 + U_{0j} + (B_1 + U_{1j}) * (TIME_{ij}) + \epsilon_{ij}$$