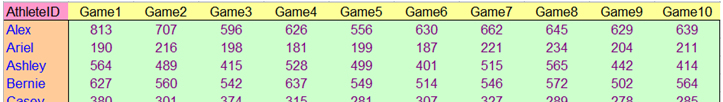
**SAS Reliability Mixed Models**

Before proceeding, view the slideshow **Linear mixed models - the basics.pptx**. This document gets you started with the procedure for general linear mixed models in SAS, Proc Mixed. The data and analyses are the same as in the Sportscience resource for mixed modeling with SPSS, with two exceptions. First, I have removed comparisons with the reliability spreadsheets at Sportscience. Secondly, SPSS has an ANOVA-based procedure for performing reliability analyses with data in wide format, which is useful for getting the "alpha" (Cronbach) reliability of a subset of Likert-scale items from a psychometric inventory or questionnaire. Unfortunately SAS does not have such a dedicated procedure. SAS's correlation procedure (Proc Corr) does have an option to output alpha reliability, but there are no confidence limits, so we will skip any analysis of the data in wide format in SAS. (The quickest way to get alpha reliability with confidence limits is to use the 2-way spreadsheet in the reliability workbook at Sportscience.)

Links to the reliability analyses:   
[One-way](#oneway)  
[Two-way](#twoway)  
[Two-way with Missing Values](#twowaymiss)  
[Two-way with a Fixed Effect](#twowaymissplus)  
[Extending the Reliability Mixed Model to Monitoring and Clustering](#extending) (and introducing graphs in Studio)   
[Using "Nesting" to Specify Repeated Measurements](#nesting)

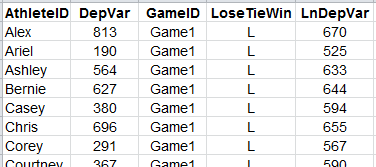
**One-way Reliability Model**

1. Find and open the workbook **reliability data.xlsx** in the location where you have saved the workshop. It should open on the tab (spreadsheet) **2-way wide**. Here are the first few observations (rows) of data:



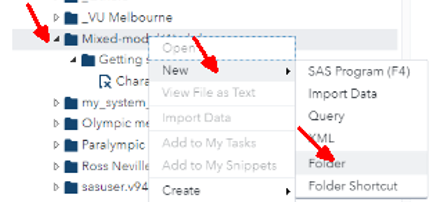
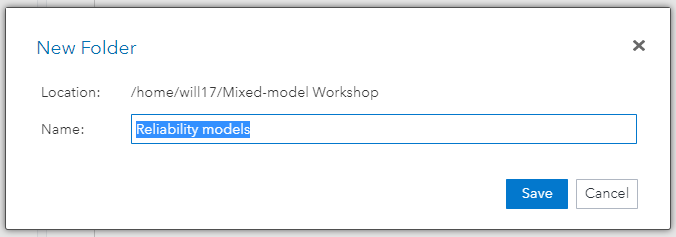
I used simulation to create these data. They represent 10 repeated measurements on each of 20 subjects. The data could be something like distance of high-speed running in games.

Click on the tab **2-way long** to see the data in long format, with the above values represented by a variable DepVar, plus two more variables: LoseTieWin and a log transformation of the dependent variable, LnDepVar (= 100× the natural log of DepVar):

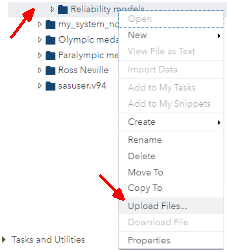


Mixed modeling is all about standard deviations (SDs) as well as means. In the following analyses we will derive SDs representing between-athlete differences and within-athlete game-to-game variation. We will also derive an overall mean and means for winning, tying and losing. The analyses have to be done with data in long format, so let's import the **2-way long** spreadsheet into SAS.

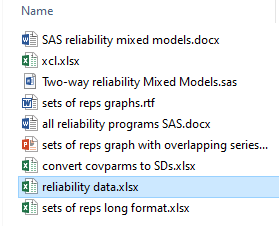
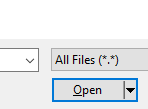
1. Logon to SAS Studio, right-click on Mixed-model Workshop in the navigation window, and create a new folder called Reliability models:

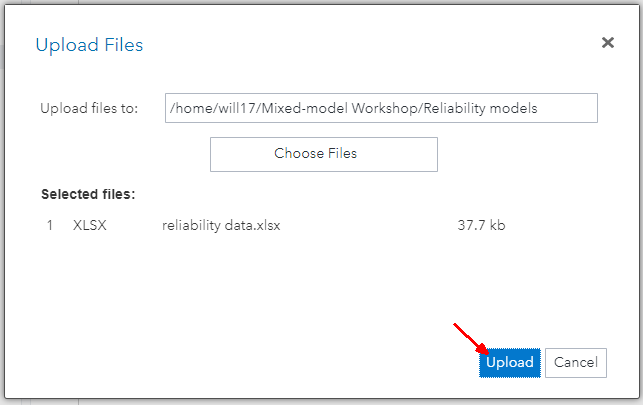
, 

1. Right-click on the folder and select Upload files:

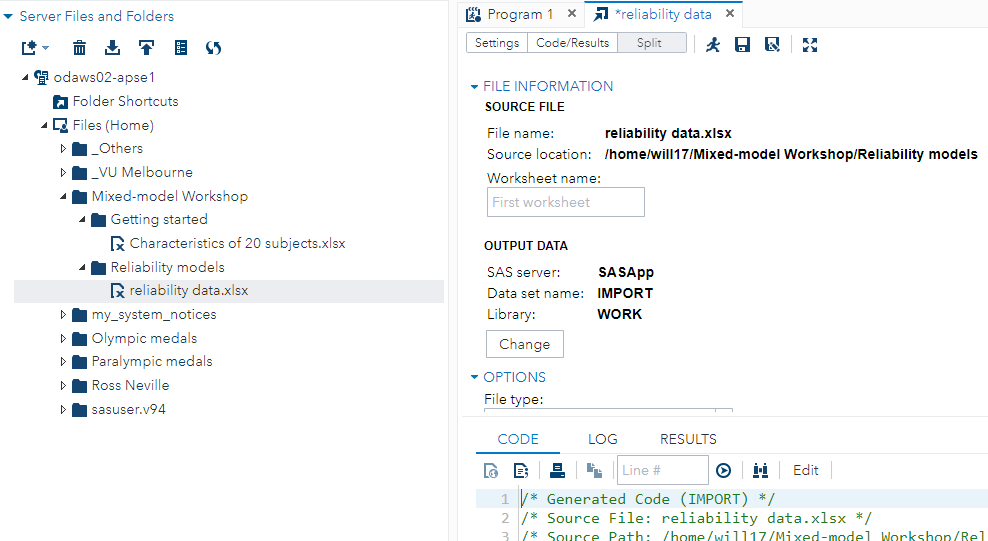


Find the spreadsheet **reliability data.xlsx** on your computer and open (upload) it.

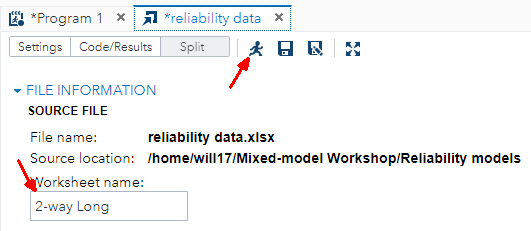
 



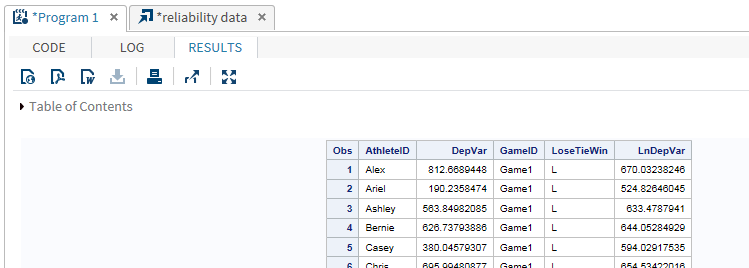
1. Open the Reliability models folder, then open the spreadsheet by double-clicking it. You get the following data-import window:



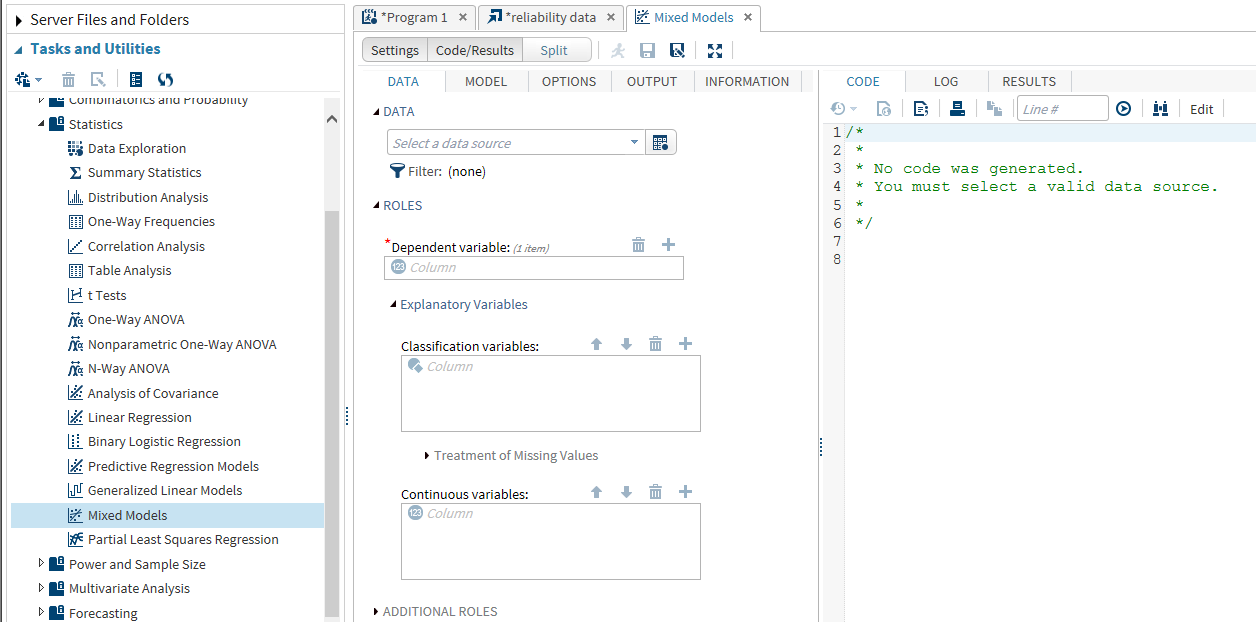
1. Where it says Worksheet name: type 2-way long (it's not case-sensitive), then click Run (red arrow):



1. We now have a dataset called import we can analyze. Type this in Program 1 to check:  
   proc print;  
   run;  
   Run it, and this is what you should get:



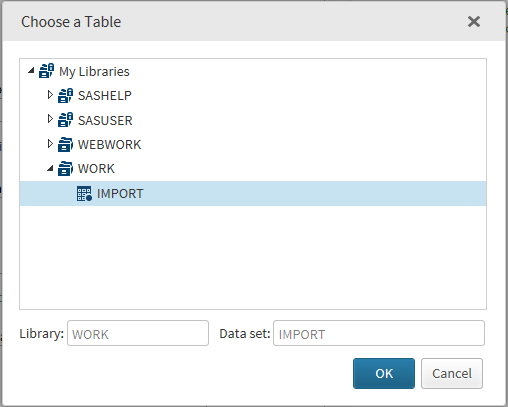
1. Now, at last, some mixed modeling! I could provide you with the complete code for the simplest of all possible mixed models using a Proc Mixed step, but let's allow SAS to guide us through it. In the Navigation frame, click on Tasks and Utilities, then Statistics, then scroll down to Mixed Models and double-click it:



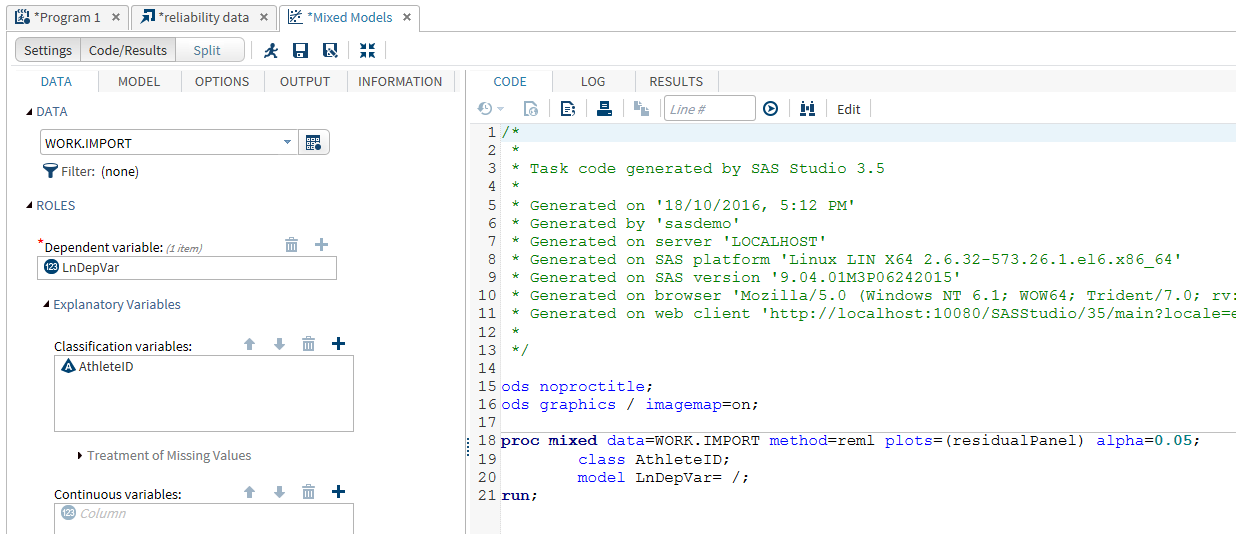
1. Click on the icon next to *Select a data source* (which prompts with Select a table)…



…then open WORK and choose IMPORT, OK:



1. At this stage, maximize the view to get rid of the navigation frame, and choose LnDepVar as the Dependent variable, and AthleteID as a classification variable:



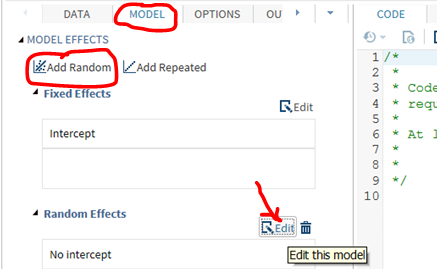
1. You'll notice code developing in a **proc mixed** statement in the CODE window on the right. If it's a bit cut off, make the Settings window narrower so you can see most or all of the code. The main things to notice in the code at this stage are…

**Proc Mixed** is going to use WORK.IMPORT (which means the same as import) as data, it's going to use a method that applies to all mixed models you'll ever use, reml (restricted maximum likelihood), and it's going to use an alpha level of 0.05 (i.e., 95% confidence limits).

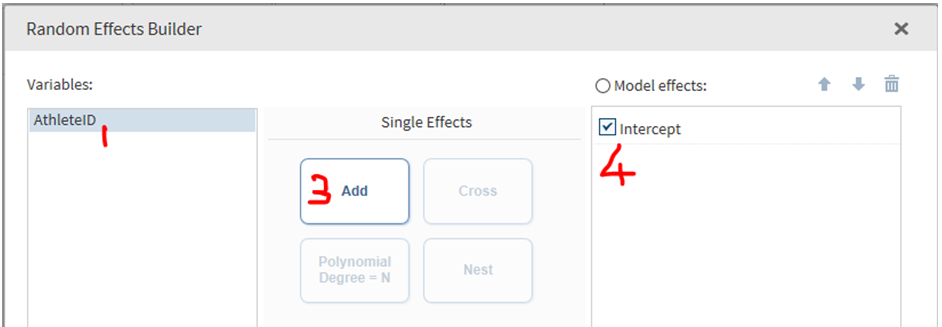
AthleteID is the only classification (or nominal or factor) variable.

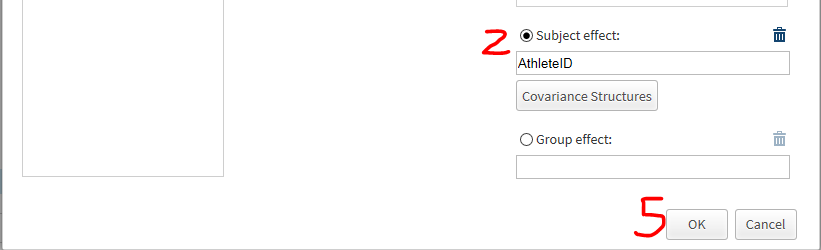
The model (which means the fixed-effects model) appears to be an equation in which the dependent variable (LnDepVar) is being predicted by nothing at all. (The slash "/" is an option delimiter, which is doing nothing at the moment.) Actually, what's not shown in the model is the usual constant in a linear equation, also known as the intercept. SAS doesn't show it, because it's always there unless you specify otherwise with noint after the slash.

1. Now click on the MODEL tab, then Add Random and click Edit for the Random Effects:

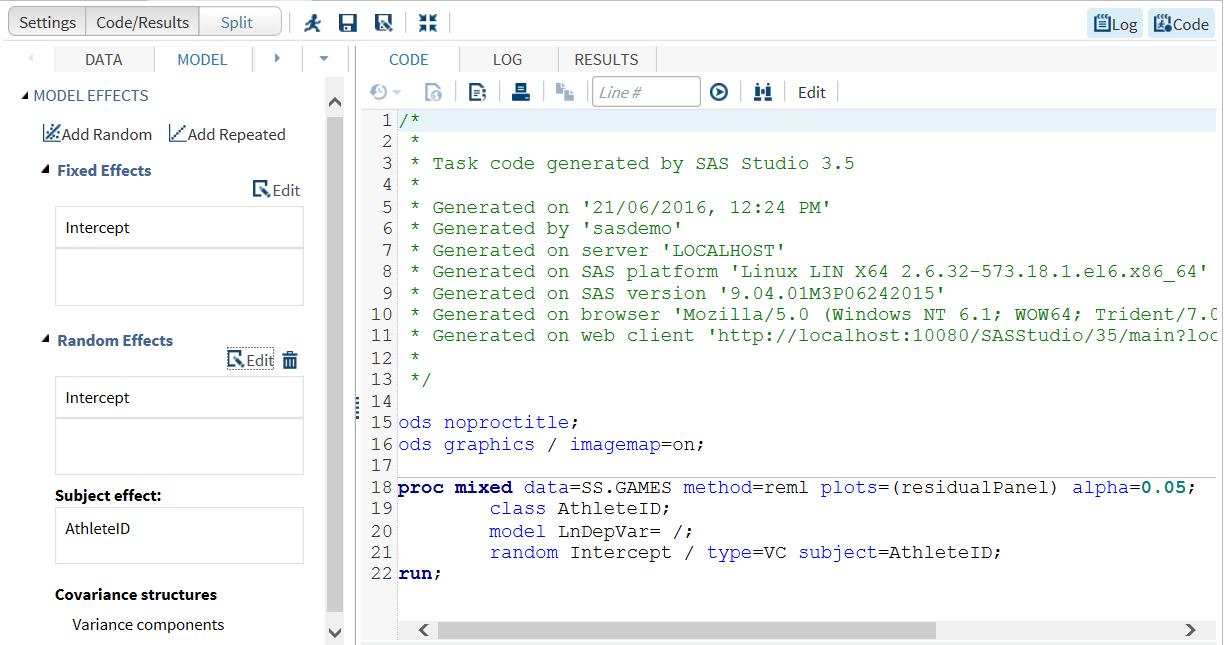


1. In Variables, click AthleteID to select it (1) click the Subject effect bullet (2), click Add (3), click Intercept to tick it (4), and finally scroll down to click OK (5)…





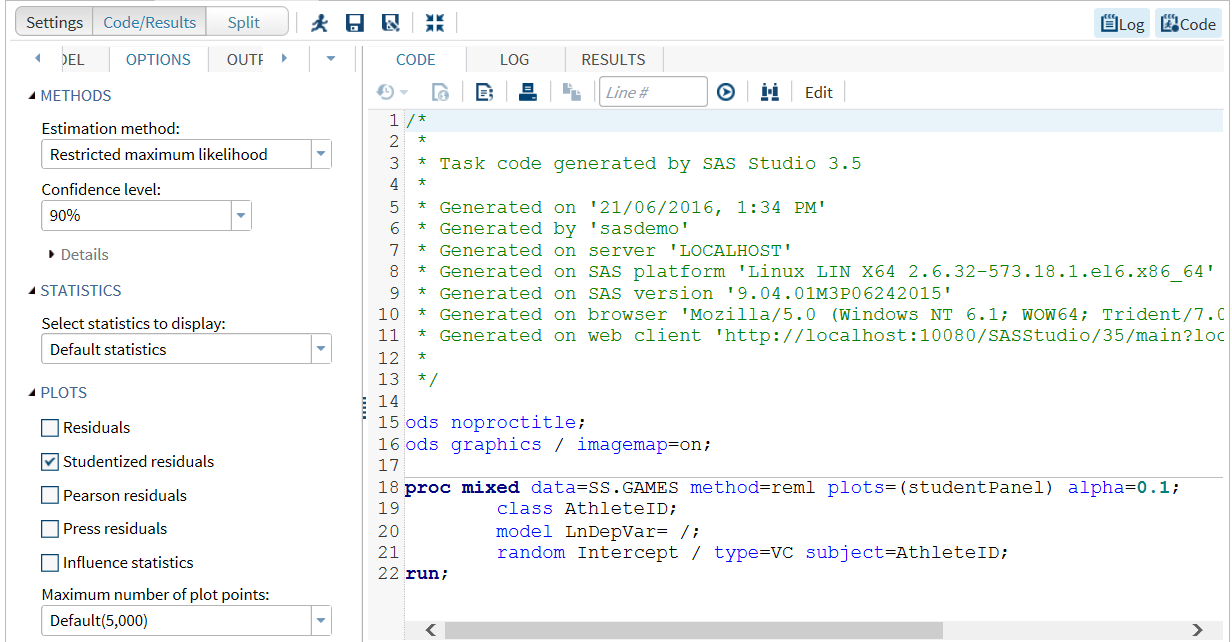
1. …which brings you back to this:



You now have a line of code for a random effect, which states that the Intercept with subject defined by AthleteID is a random effect. Type=VC refers to the type of relationship among the random effects, and VC stands for variance components, which means independent (uncorrelated), but since there is only one random effect here, the type is irrelevant.

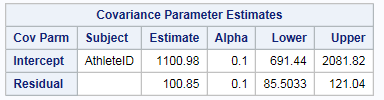
This whole random effect statement is just another way of saying that AthleteID is a random effect. You could get the same outcomes in the analysis with simply random AthleteID;. In fact, you could get this simpler-looking random-effect model by adding AthleteID to the Model effects window of the Random Effects Builder and by not including an intercept. Let's save that alternative approach for another random effect in the next analysis.

Now click the OPTIONS tab (to the right of the MODEL tab, if you can't see it), select 90%, and Studentized residuals rather than Residuals. The plots option in the proc mixed line of code changes to studentPanel, and the alpha changes to 0.1:

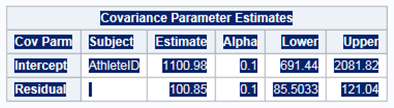


Studentized residuals are standardized residuals; that is, they have a standard deviation of 1.0 and a mean (like most random effects) of zero. They are great for looking for outliers: an observation with a standardized residual >3.5 is a candidate for sample sizes of ~50, >4.0 for sample sizes of ~500, >4.5 for sample sizes of 5000, etc. (Reference: the progressive stats article in Sportscience and MSSE.)

1. Now run the program. Scroll through the RESULTS window to the Covariance Parameter Estimates…



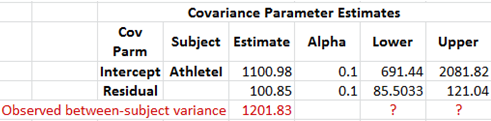
…which are the summary statistics for the random effects, expressed as variances with confidence limits. These need to be converted into standard deviations to interpret them. And they will be SDs derived from a log-transformed variable, so they need to be back-transformed to percents. The quickest way right now is to do it with a spreadsheet. Eventually you will be able to do it within SAS. So, highlight the table of covariance parameters…

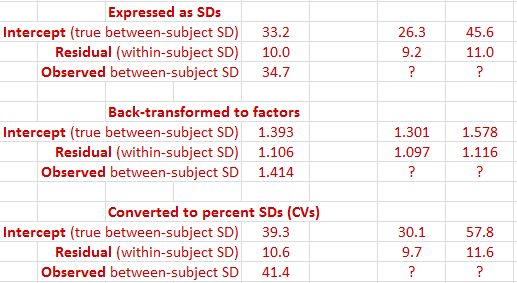


…copy to the clipboard, and paste into an Excel spreadsheet for processing. To save time, I have done it for you. Open the spreadsheet **convert SAS covparms to SDs.xlsx**. If you do it via the navigation frame in SAS, exit the maximized view…



…open Server Files and Folders in the navigation frame, right click on the spreadsheet, and select Download File. Open it and find this…



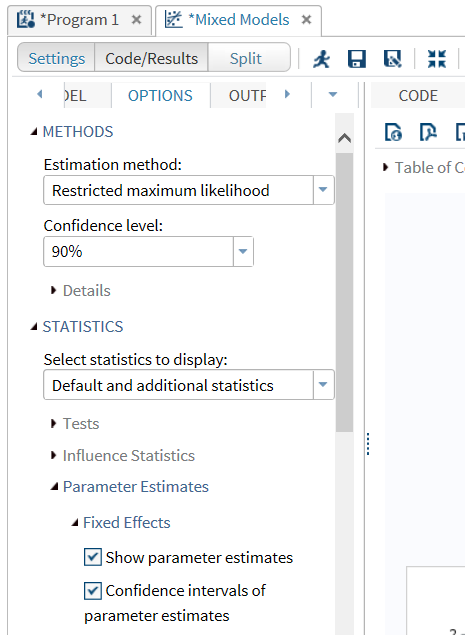


The stuff in black has come in from SAS. The stuff in red is what I have added. Click in the cells and figure out the formulae. Sorry about the lack of some confidence limits–too difficult for now.

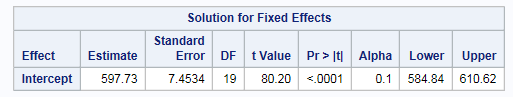
The whole point of mixed modeling is to take into account and estimate standard deviations. Here the within-subject SD is the typical variation in an athlete's score from game to game, the true between-subject SD is the typical difference between players when there is no within-player variation, and the observed between-subject SD is what you expect to see between players in any given game.

What about the fixed effects in this mixed model? There was only one, the intercept or overall mean value, and SAS hasn't even provided it by default! It's pretty boring, that's why, but let's output it anyway.

1. Maximize the view for the Mixed Models tab, and in the OPTIONS tab under Select statistics to display, select Default and additional statistics, and tick Show parameter estimates and Confidence intervals…, then Run:



1. Scroll down to Solution for Fixed Effects:

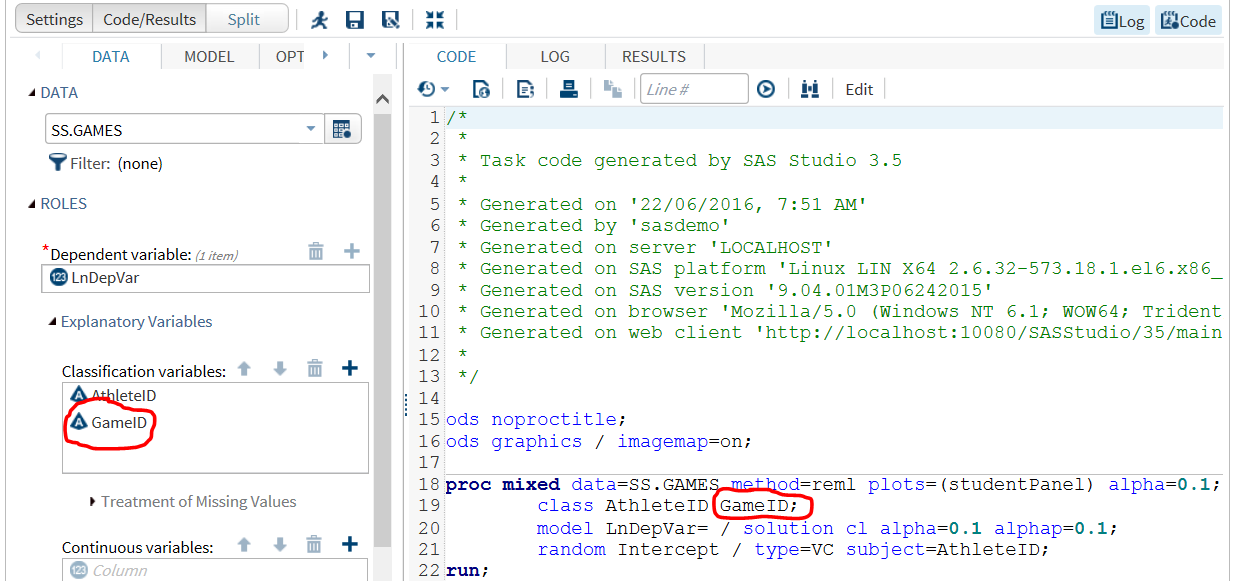


The Estimate (i.e., the mean) and its lower and upper confidence limits need back transformation, which I have also done in the spreadsheet. The mean is 394 meters, or whatever the units of the original variable were. You don't normally show confidence limits for the mean, but you do usually show the observed between-subject and the within-subject SDs, here 41% and 11%.

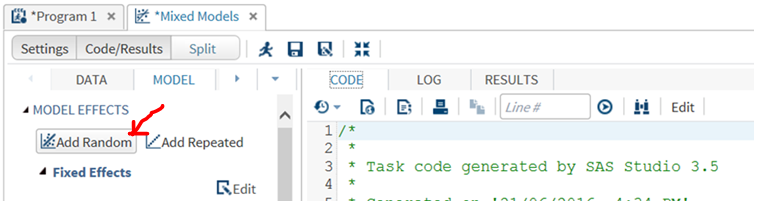
1. Ooops, I nearly forgot about the plot of the residuals. Check it out. No problems there.

**Two-way Reliability Model**

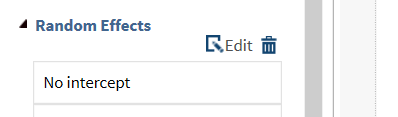
1. Now let's do the analysis with GameID included as a random effect. In other words, imagine that the games are also a random sample.
2. Click the CODE tab to bring back the program, find and click on the DATA tab in the Settings window, and add GameID to the Classification variables (you might have to resize the settings window to see the **+** for adding another variable). GameID will now appear in the class statement:



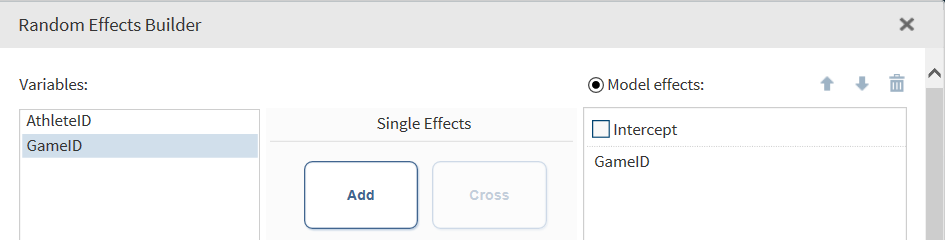
1. Now click on the MODEL tab in the Settings window, then click on Add Random:



You might have to scroll down in the Settings window to find that another random effect has been added:

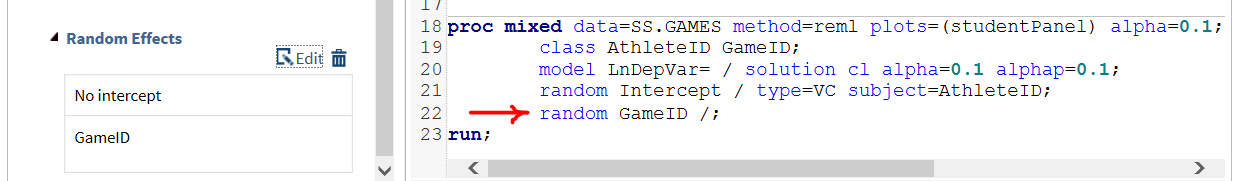


Click on Edit, scroll up (if necessary) in the Random Effects Builder, and add GameID to the Model effects. Do NOT click Intercept:



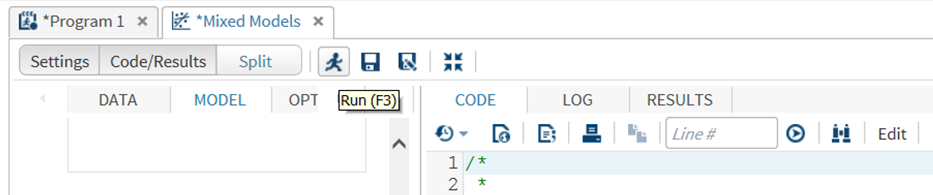
Then scroll down (if necessary) to find and click OK…

You will see that GameID has been added as a random effect:

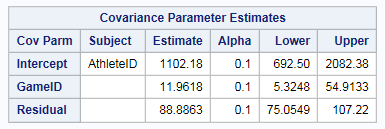


Recall that this code is equivalent to random Intercept/subject=GameID.

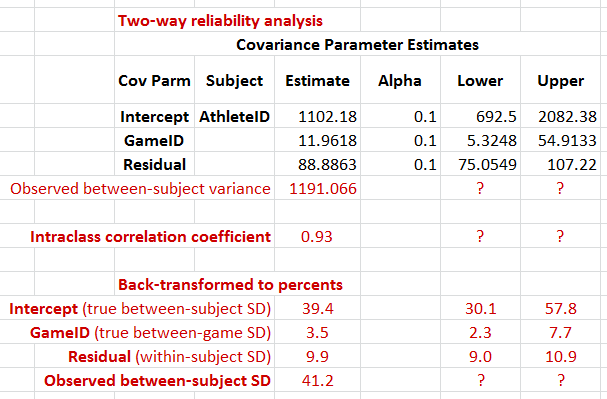
1. Now run the program…



…and scroll down the RESULTS window to the only thing we're interested in…



…which becomes this in a spreadsheet (click on the **Two-way** tab of **convert covparms to SDs.xlsx**):



1. Notice I have now added an intraclass correlation coefficient. The value represents the correlation you expect to get for the usual correlation between any two games. Check out the formula:   
   (pure between-subject variance)/(pure between-subject variance + within-subject variance).

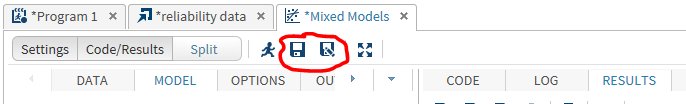
I could have derived a correlation from the one-way variances with the same formula. I didn't, because the resulting ICC would represent an impractical correlation for these kinds of data: it's what you would expect to get if you chose two games for the first player, two different games for the next player, and so on, then correlated the two columns of data. In other words, all the data would come from different games. Instead, the ICC from a two-way analysis is always what we are interested in: you choose two games, and the correlation is unaffected by the fact that the mean for one game will be different from the mean for the other.

Well, I'm assuming the mean for each game is different. It's what you would expect, and the GameID random effect tells us how different they are: its SD is the typical difference in the mean as you go from game to game. However, it is not quite what you would get if you derived the SD of the means of each of the games. That SD would be inflated by the random variation that each player shows in each game, so it would be slightly greater than the SD here. That's why I have called the SD derived from GameID random effect the *true* between-game SD. I sometimes use the term *pure* instead of *true*.

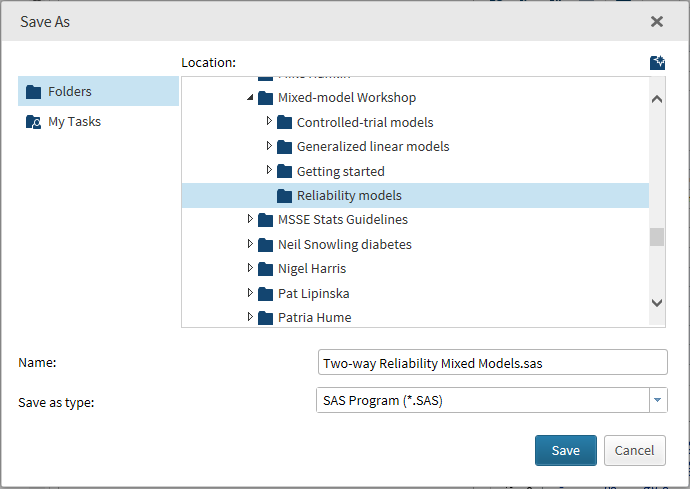
Compare the Intercept and Residual here with those from the one-way analysis: the intercepts are practically the same (differences between players should not depend on whether some games are harder than others), but the residual here is smaller. I hope it's obvious why: the variation in the mean of each game from game to game contributed to the within-player variability in the one-way analysis, but it has been taken out of the picture here.

**Two-way Reliability Model with Missing Values**

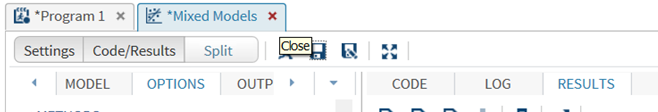
1. Now let's repeat the above analysis when data have missing values. At this stage we should save the program, via either of these two icons:

.

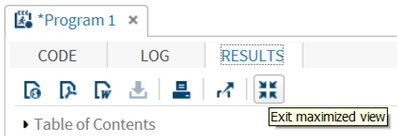
You can save it either as a SAS program (.sas) or a task template (.ctk). When you re-open a SAS program, your only approach to modifying the program is via the code itself. When you re-open a template, your only approach to modifying the program is via the settings window. Unfortunately the task template is useless: I thought it would save variable names etc., but I discovered that as soon as you change the dataset, you lose everything. So let's work directly with the code by saving as a SAS program with some suitable name in the Reliability models folder, as shown here…



1. Close the Mixed Models window via the x on the tab…

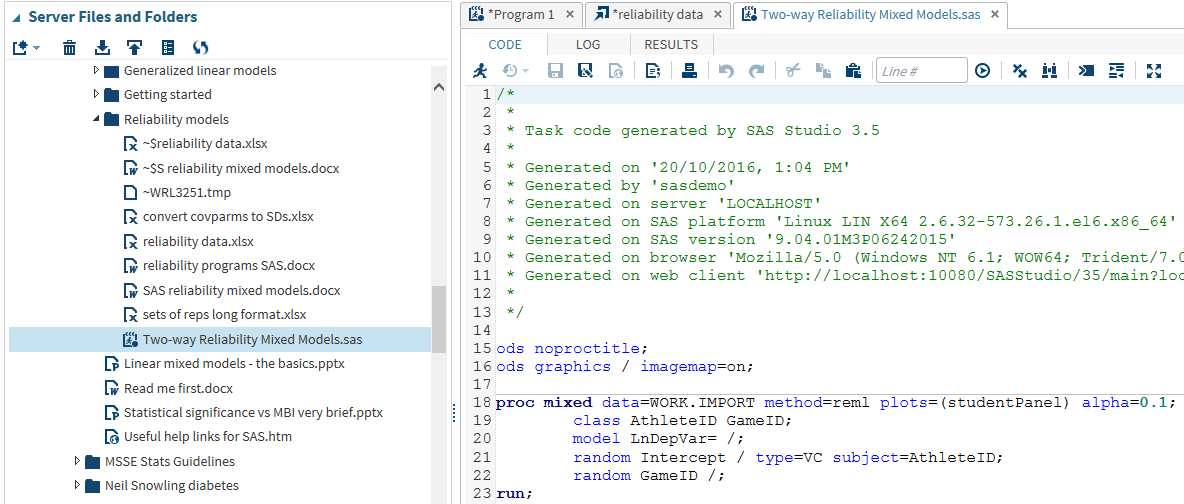


…and exit maximized view (if necessary)…

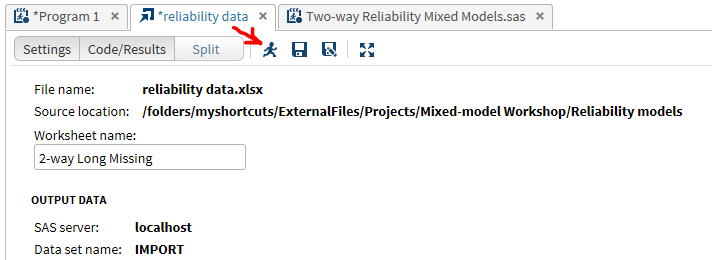


…so you can access the sas program and the dataset in the Server Files and Folders:

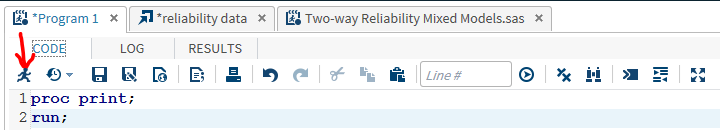
1. Find and open (double-click) the program to get this:



1. Now import the dataset with missing values. Double-click **reliability data.xlsx**, add missing to 2-way long in Worksheet name, then click the Run icon (and click Replace in the warning window):



1. The dataset is available as a new temporary set called IMPORT. We'd better check it out first. Click on the Program 1 tab, which should still show proc print;run; Click the Run icon:



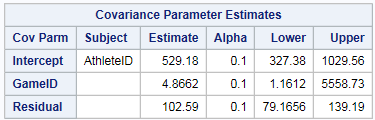
1. Scroll down the RESULTS window and you will see that missing data for LnDepVar are shown as decimal points:



This is the way SAS stores and refers to missing data for numeric variables.   
For example, you could have a line of code in a data step that states   
if LnDepVar = . then <whatever>;   
For nominal, class or factor variables, missing values are blanks, so a line of code could be  
if AthleteID = "" then <whatever>;  
You can also write " " instead of "".

Note that missing values can be present as observations with actual missing values, or the observation can be completely absent—it makes no difference to Proc Mixed.

1. Now click on the Two-way Reliability Mixed Models.sas tab. The program will run with the new dataset, because the new dataset is called import. So click the Run icon. If it runs correctly, you find this in the RESULTS window:



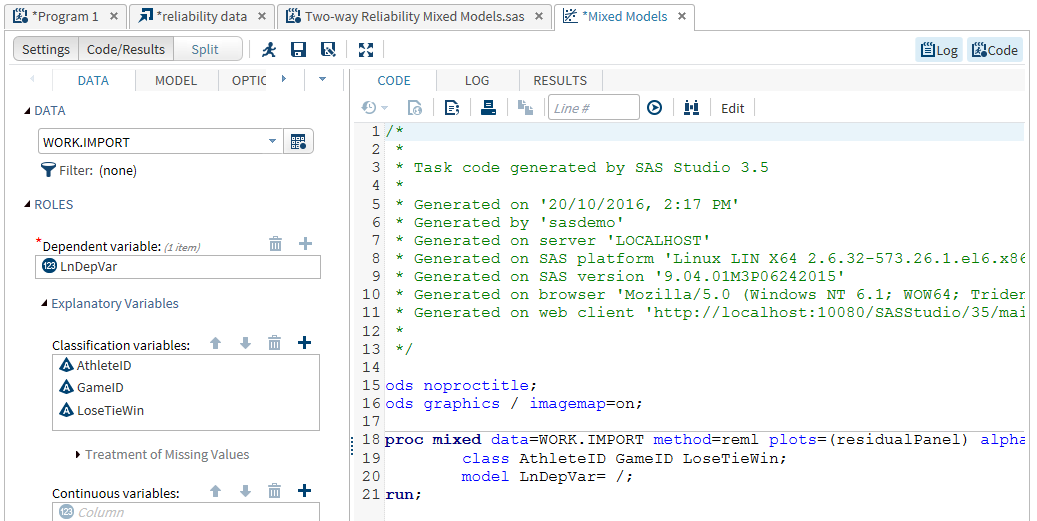
The values have all changed, but that's to be expected, because it's a different dataset: a subsample of the previous data. Notice also that the confidence interval for GameID is grossly asymmetric (1.16 to 5558). The problem here is that SAS is using by default an inappropriate method for generating the variances and their confidence limits. For most of the analyses we are likely to do with modest sample sizes, variances other than the residual can be negative. Here's why. Suppose the population or large-sample value of a variance is close zero (relative to the magnitude of the residual variance). You never get the population value with a sample, owing to sampling variation. OK, so with small samples, the sampling variation will be large; that is, the value of the variance derived from sample is going to vary in a big way from sample to sample. Hence the confidence interval has to be wide. But if you assume that the variance cannot be negative, the lower confidence limit will be positive and close to zero, while the upper confidence limit has to "balloon" out to some huge value, under the assumption that the mixed model makes for the sampling distribution of a variance. Hence that huge value for the upper confidence limit of the GameID variance. But that huge value is ridiculous, because there is no way that the game-to-game variation would be this large in any sample you are ever likely to draw. (Back transform the upper confidence limit and you get 111%, or a factor variation from game to game of ×/÷2.11!) If you aren't convinced, consider what happens if the observed GameID is only a tiny bit greater than zero. Now, the upper confidence limit will be on the order of millions, which is totally unrealistic.

The solution is to allow negative values of the variances and their confidence limits, a feature of SAS that SPSS and R currently do not offer. Let's leave it for the next analysis, in which we include a fixed effect.

1. We could add the code directly for the fixed effect, but you need to see how to develop the model with the point-and-click approach. That means developing the model from the beginning again, but it's good practice. It's the same model, but with the addition of WinTieLose as a fixed effect.

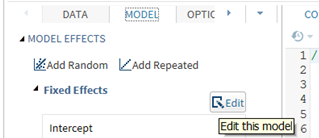
**Two-way Reliability Model with Missing Values and a Fixed Effect**

1. Find Mixed Models in Tasks and Utilities and double-click it. Choose LnDepVar as the dependent, and AthleteID, GameID and now LoseTieWin as classification variables:

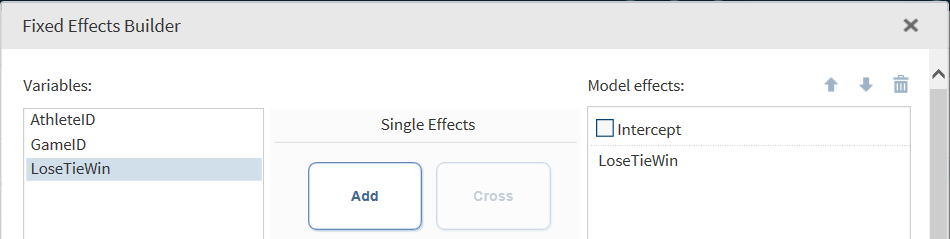


Notice that LoseTieWin is now included in the class statement in the CODE window.

1. Click on the MODEL tab and Edit the Fixed Effects:

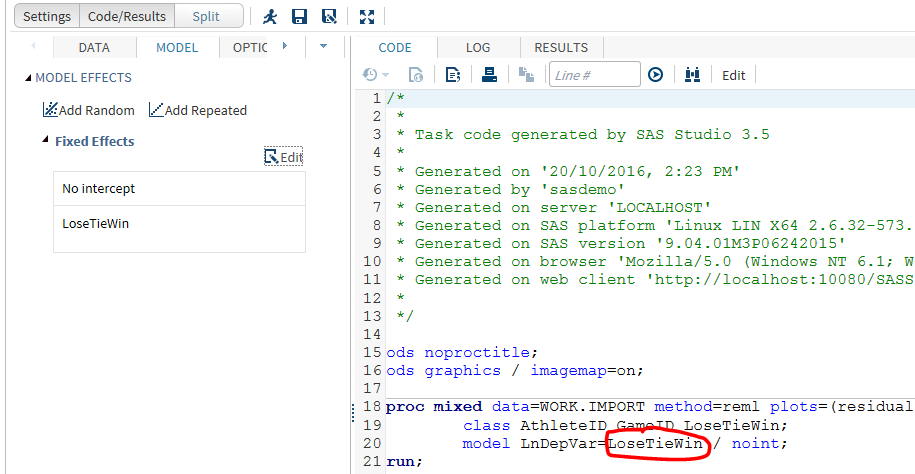


1. Add LoseTieWin to the Model Effects in the Fixed Effects Builder. I've unclicked the Intercept, but if you've got a nominal fixed main effect, it doesn't make any difference to the final estimates if you include or exclude the intercept:

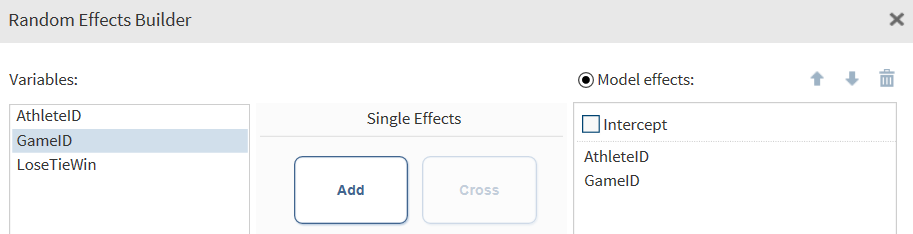


Scroll down to the bottom of the window and click OK.

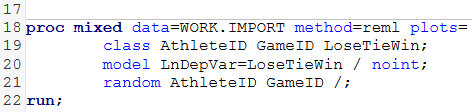
1. LoseTieWin is now in the model statement:



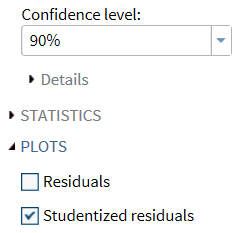
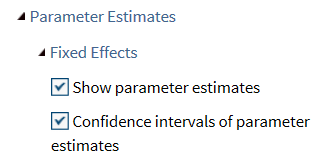
1. Now add the random effects we had previously: AthleteID and GameID. Do it in one hit by adding them both in as Model effects, without an intercept, then OK:



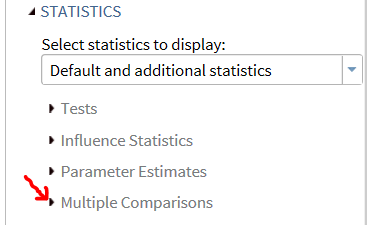
And here's the model so far. Notice the much simpler random statement:



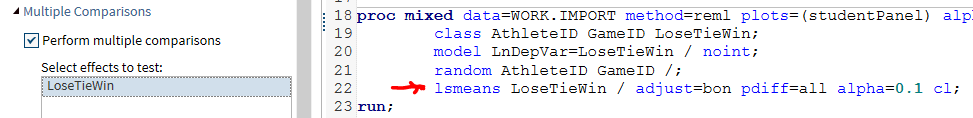
1. Now click on the OPTIONS tab and select the options we had previously: 90% confidence level, PLOTS of Studentized residuals, Show parameter estimates and their confidence intervals:

1. Click Statistics, choose Default and additional statistics, and click Multiple Comparisons:

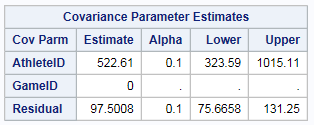


SAS forces us to choose a method for adjustment of Type I error. I've opted for Bonferroni, but we will ignore it in the output. Choose a 0.10 significance level, and notice that a new line of code has appeared:

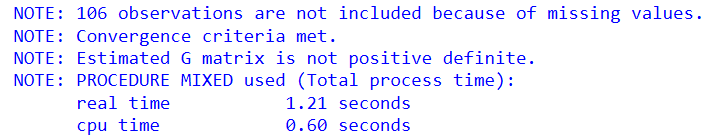


lsmeans stands for *least-squares means*, which is the name used for *estimated marginal means* when they are estimation by analysis of variance. SAS has kept the old name.

1. It's time to click the Run icon. Scroll down the RESULTS window and you'll find this:

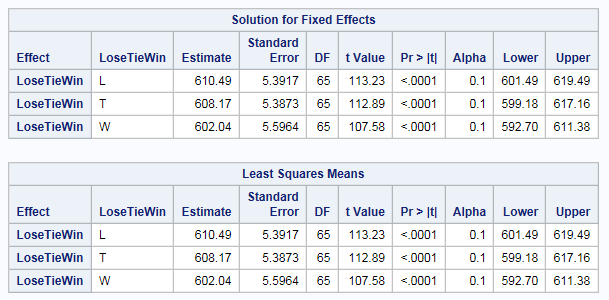


Oh no, zero variance for GameID!? Did something go wrong? Check out the LOG:



It's only a note, but when you get zero variance for a random effect, you will get "Estimated G matrix is not positive definite." The problem here is that SAS would like to make the variance negative, but by default it can't. Let's come back to this problem shortly.

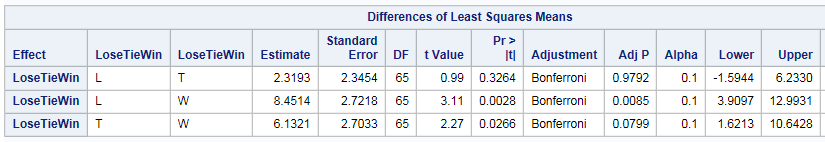
1. Go back to the RESULTS window and scroll down to these:



The solution for fixed effects is exactly the same as the least-squares means. This happens with simple models, but they differ when there are one or more numeric covariates in the model. In that case, the least-squares means are what you want, because they are adjusted to the mean value of the covariates. That is, they are the mean values predicted when the covariates are at their mean values (and you can choose other values). They also differ if you have an intercept in the model, but let's not worry about that.

The above estimates and confidence limits need to be back-transformed to raw values, using exp(Estimate/100). It's done for you in **convert covparms to SDs.xlsx** in the tab **Two-way missing**.

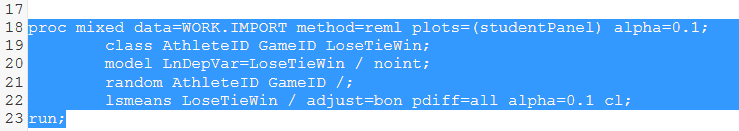
1. Next, the differences between the least-squares means:

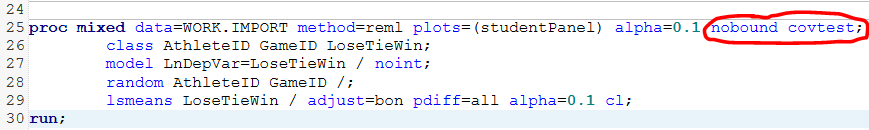


Notice that the values for Estimate are for the first column of LoseTieWin minus the second column; that is, the estimate 2.3193 is the value for L minus T, and so on.

These estimates also have to be back-transformed, but to percents, not raw values. See the spreadsheet. The back-transformation is the same as for back-transforming SDs, except that there's no square root inside the EXP(). You can see that the percent effects are approximately the log values. Hence the factor of 100 in the log transform: effects and SDs of the log-transformed values "make sense" without back-transformation. You still have to back-transform to get the correct percent values, especially for effects and SD >10%.

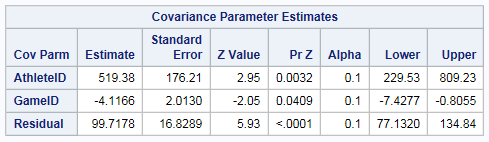
Now that we have a fixed effect, we can do some inferences about magnitudes. I've made a table of magnitude thresholds in the spreadsheet, based on standardization with the observed between-subject SD. The inferences should be done with the log-transformed values. To do a full probabilistic magnitude-based inference, you would copy the effect, the threshold for small, and the p value into the spreadsheet for confidence limits and clinical chances at Sportscience (or you could copy the confidence limits instead of the p value into the spreadsheet for combine/compare effects). But it's a simple matter to determine by eye that all the effects are clear here. L-T is trivial, and the other differences are small.

1. Let's now return to the problem of zero variance for GameID. There is no option for allowing negative variance in the Mixed Model task, so we'll have to do it directly with the code. Highlight and copy the program to the clipboard…  
   

…paste it at the end of Two-way Reliability Mixed Models.sas, and add the two terms nobound covtest shown in the Proc Mixed line:  


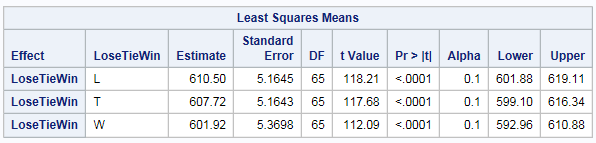
Nobound stands for no (lower) bound on the variances, and covtest produces standard errors for the variances (which we could have selected in the Mixed Model OPTIONS / STATISTICS / Tests).

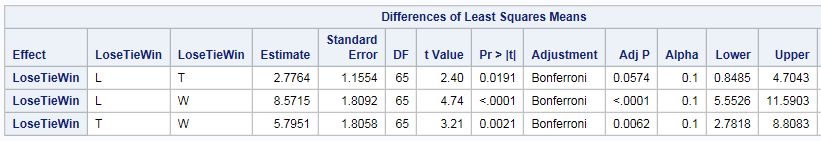
1. Highlight the proc mixed step down to run; and run it. You should see this for the covariance parameters:



Now the estimate is negative, and even the upper confidence limit is negative. Exactly what does this all mean? Obviously there is an *observed* SD for the game mean between games, and even if the true between-game SD were zero, you would expect to see a negative variance in half of all samples. But in that case you would expect the upper confidence limit to be a reasonable positive value. Here the upper confidence limit shows that the game-to-game variation in the mean of games is *less* than you would expect by chance, given the variability of the players from game to game and the differences in the means due to winning, tying and losing. If these were real data, there would probably be a good explanation.

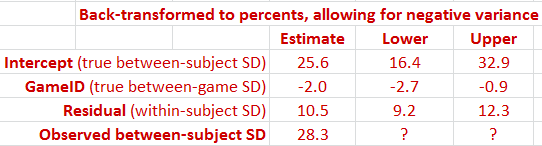
1. We'll come back to the random effects shortly. Meantime, the data are the data, and when you allow negative variance, something interesting happens to the least-squares means for WinTieLose. The means themselves are hardly different, but the confidence intervals for the differences are a lot narrower:





The differences between the least-squares means were already clear, but they are even clearer now. They're back-transformed in the spreadsheet.

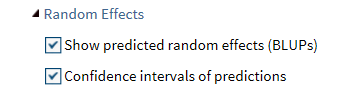
1. Now let's get back to the random effects. Converting a negative variance to a standard deviation is problematic, because you can't take the square root of a negative number. Many years ago I decided that it's OK to change the sign, take the square root, and call it a negative SD. There is no such a thing, but it tells the reader than the variance was negative, and the estimate along with the confidence limits gives the reader an idea of the possible magnitude of the random effect. Here are the back-transformed values, done with the IFERROR function in Excel to automatically change the sign, when the variance is negative:



I included covtest in the proc mixed statement to output the standard errors for the variances so you can see that, with the exception of the residual variance, SAS computes the lower and upper confidence limits as Estimate ± 1.65\*StandardError. (It would be ±1.96 for 95% confidence limits.) In other words, SAS is assuming that the sampling distribution of the variance is normal. You will sometimes see the expression "asymptotically normal", which means the sampling distribution is normal when the sample size is large enough. We have no way of knowing if the sample is large enough, so we just assume it is normal anyway. It is certainly better to assume normality than the default, where the confidence limits are derived from the chi-squared distribution.

Why doesn't SAS include nobound by default? One good reason is that the iterative process that fits the model to the data sometimes can't get started with nobound (the error is "infinite likelihood"), so you have to help it with a parms statement to supply initial reasonable values of the variances: an extra complication for the casual user. Another reason is longer computation times with nobound: it can take hours or days for a program to run. Maybe even the creators of Proc Mixed were suspicious about the reality of negative variances, but as I stated above, when the true variance is near zero, sampling variation should be allowed to produce negative variance.

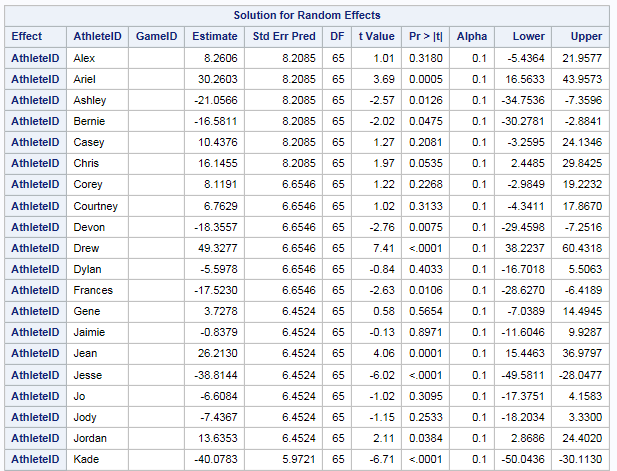
1. One final cool feature of mixed models is the "solution" for the random effects. You output these by ticking these boxes in OPTIONS:



SAS calls them predicted random effects here, but in the code the option appears as solution. Add it into the random statement as follows:

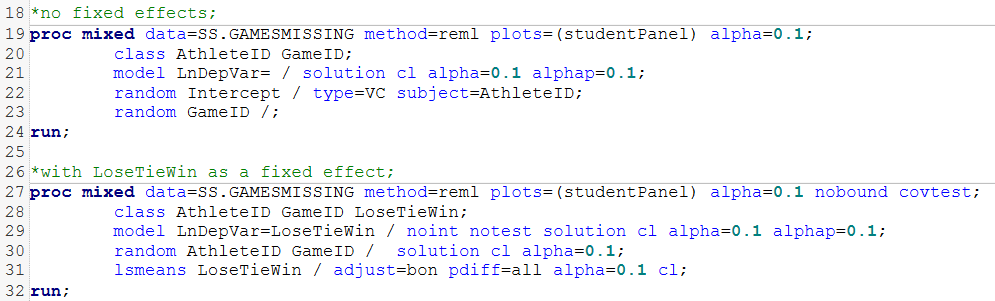


Now run the program again, and here's what you get:

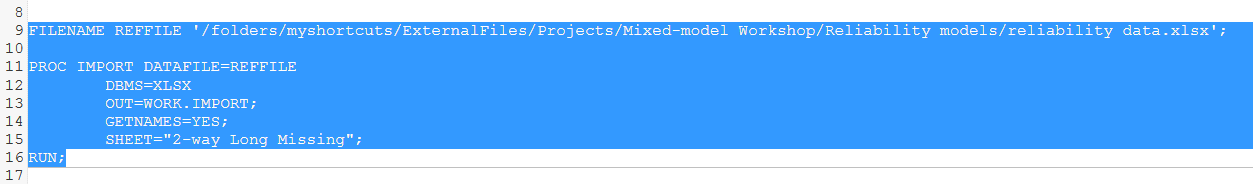


The Estimate, with its confidence limits, tells you how good or bad the athlete is relative to all the other athletes: the values back-transform to percents like any other effect (no square root) and represent the athlete's overall mean value above or below the average athlete, who has a value of zero. You can output the solution to a dataset and rank order it within SAS, but it will be easier for you to paste into a spreadsheet and sort them there.

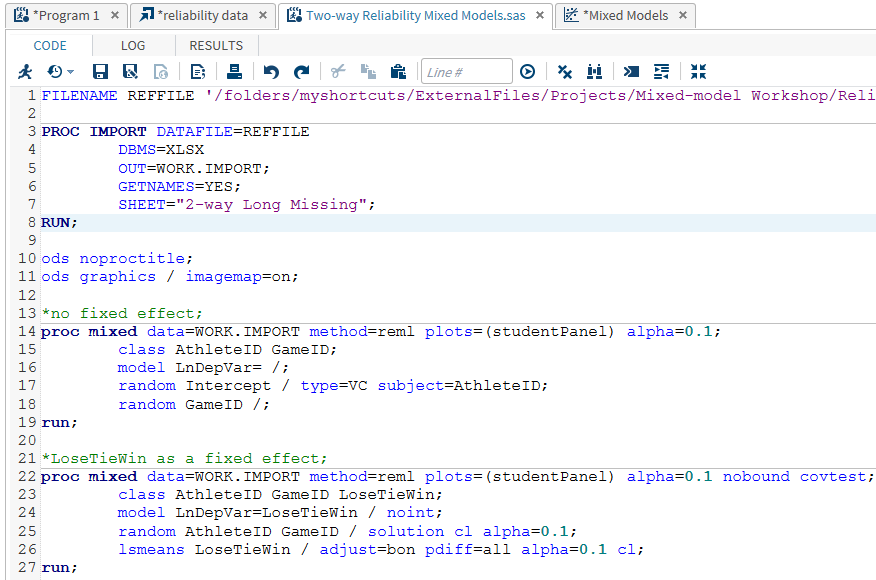
1. We need to tidy up and close some of the windows in SAS before proceeding. Add an explanatory comment above each of the proc mixed steps:



Delete the comment text at the top of the program. To keep a copy of the code that imported the data, click on the reliability data tab, highlight and copy this text…



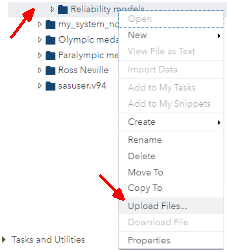
…and paste it into the top of the SAS program. This will allow you to run the program again in a new SAS session without manually importing the data. The final program should look like this:



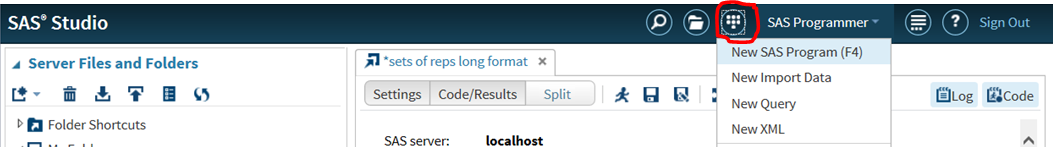
Save it, then close it and close all the other windows without saving them.

**Extending the Reliability Mixed Model to Monitoring and Clustering**

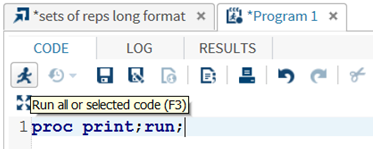
1. By including the LoseTieWin fixed effect, we have already started to develop a mixed model for monitoring. Imagine that, in addition to LoseTieWin, you have a variable for date of each game, and you have all the games in a season. You could include Date as a numeric fixed effect and model a seasonal trend in whatever dependent variable. Or instead of numeric Date, you could have a nominal fixed effect with levels representing different phases of the season. Because these extra effects have values that change within each subject, you could also model individual differences in their effects, by including them interacted with the subject random effect.
2. Here's a fairly simple example of real data with clusters and trends. The data come from a study of the effects of caffeine on repeated sprints, done as a crossover with placebo and several doses of caffeine, each a week apart. On each of the three testing days, the athletes performed four sets of six all-out 30-m sprints. To keep it simple, I've included the data only for the placebo. The aim of the analysis is to quantify how much the subjects fatigue in each set of six sprints, taking into account and estimating individual differences in the fatigue. We'll also account for any consistent changes in mean sprint time between sets (i.e., the extent of any fatigue between sets), and any variability in each subject's mean sprint time from set to set (which probably won't make sense to you until we develop the model). Oh, and most subjects make a bigger effort on the last sprint of a set, so we'll account for and estimate that, too, along with its individual differences!
3. Right-click on the Reliability models folder and select Upload files:



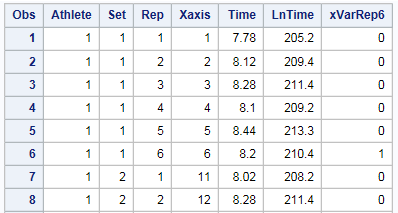
1. Find **sets of reps long format.xlsx** the workshop folder, open (upload) it, double-click it in the Server Files and Folders window to set up the import program, then Run to import it.
2. It came in as IMPORT2 for me. For you it might be IMPORT. Whatever, view the data with proc print in a new program window…



…type proc print;run; and run it:

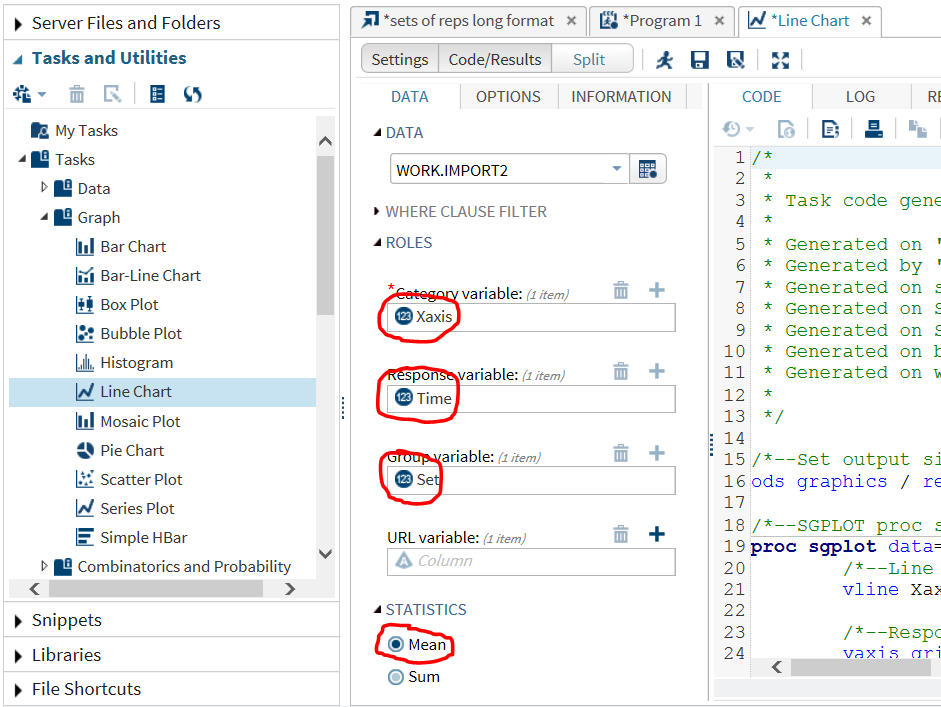


Here's the first few observations in the RESULTS window:

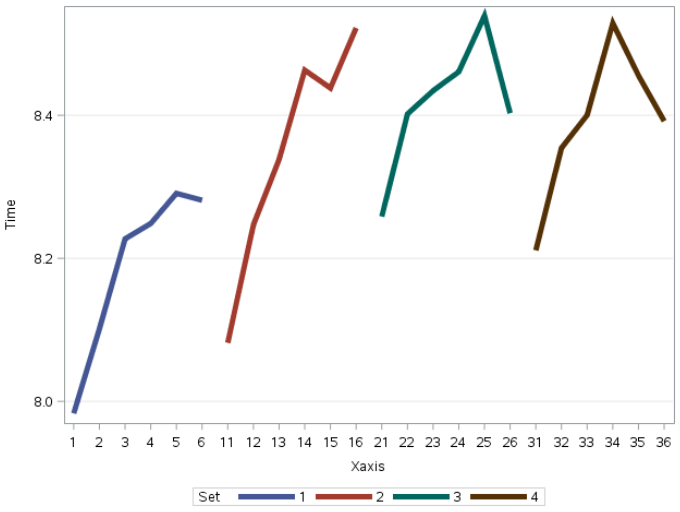


The variables Athlete, Set, Rep, Time and LnTime should be obvious enough. xVarRep6 is a dummy variable I generated in Excel to model extra effects on the last (6th) sprint of each of the four sets. (A simple name like Rep6 would be OK.) Check out its values. Xaxis is another variable I generated in Excel to make a chart (graph) so we can see what's happening in each set.

1. Let's make a chart first. In the navigation frame, find Task and Utilities/Tasks/Graph,   
   double-click Series Plot, and choose these DATA values, and run it…



1. Maximize the window and make the Settings window narrower, if necessary, to see the graph properly:



There's obvious fatigue over the first five reps in each set, but in all but the second set the athletes have gone relatively faster on the last rep. So we'll develop a model that estimates linear fatigue in each set, apart from the last rep.

1. This is an appropriate point to show you some more sophisticated graphs that you can publish and/or put into a Powerpoint slide. Graphs like the above usually have SD bars on the points, which are available only in the Scatter Plot, but not as a point-and-click option. Here's the code that generates the means and the upper and lower values of the SDs, then the code for proc sgplot to plot the means and error bars:

\*Import the spreadsheet sets of reps long format.xlsx as IMPORT;

proc sort data=IMPORT;

by Set Rep;

title "Means and SDs";

proc means noprint data=IMPORT;

var Time;

by Set Rep Xaxis;

output out=meantime n=NoOfSprints mean=TimeMean std=TimeSD min=TimeMin max=TimeMax;

proc print data=meantime noobs;

var Set Rep Xaxis NoOfSprints TimeMean TimeSD TimeMin TimeMax;

format TimeMean TimeSD TimeMin TimeMax 6.1;

run;

data meantime1;

set meantime;

MeanPlusSD=TimeMean+TimeSD;

MeanMinusSD=TimeMean-TimeSD;

run;

title height=1.5 "Means and SDs, series side-by-side";

ods graphics / reset width=16cm height=20cm imagemap attrpriority=none;

proc sgplot data=meantime1 noborder;

styleattrs

datacolors=(black blue red green)

datalinepatterns=(solid)

datacontrastcolors=(black)

datasymbols=(circlefilled squarefilled diamondfilled trianglefilled);

\*reg x=Xaxis y=TimeMean /degree=1 nomarkers lineattrs=(thickness=1) group=Set;

\*scatter x=Xaxis y=TimeMean /yerrorupper=MeanPlusSD yerrorlower=MeanMinusSD

errorbarattrs=(color=black) group=Set

filledoutlinedmarkers markerattrs=(size=18) name='abc';

\*unstar the above two lines and star off the next two lines to fit regression polynomials;

scatter x=Xaxis y=TimeMean /yerrorupper=MeanPlusSD yerrorlower=MeanMinusSD

errorbarattrs=(color=black) group=Set;

series x=Xaxis y=TimeMean /markers filledoutlinedmarkers markerattrs=(size=18)

lineattrs=(pattern=solid) group=set name='abc';

keylegend 'abc' /title="Set:" noborder titleattrs=(size=16) valueattrs=(size=16);

xaxis label="Repetition+10\*Set" labelattrs=(size=16) valueattrs=(size=16);

yaxis label="Sprint time (s)" labelpos=top labelattrs=(size=16) valueattrs=(size=16);

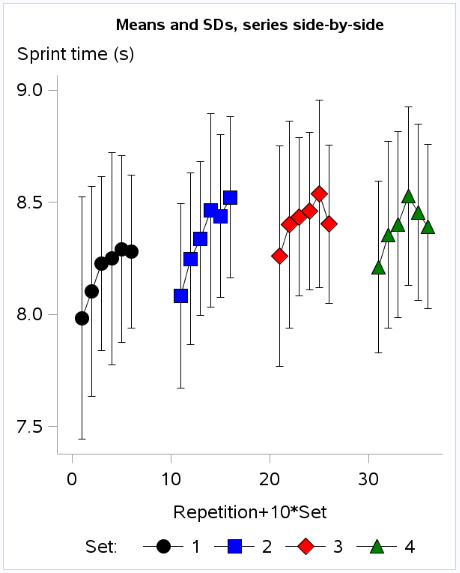
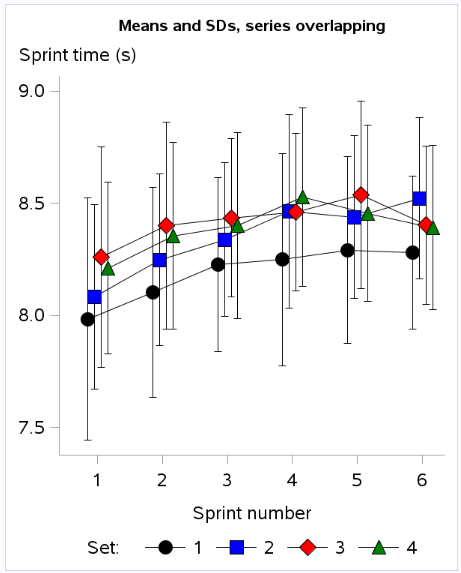
\*refline 0;

run;

ods graphics / reset;

The details in this code will make more sense when you have to plot your own data, so I will deal only with the most important issues.

I have chosen options for graph, font and symbol sizes that will look good when the graph is cleaned up for publication or presentation in a slideshow. (The choice was dictated mainly by the fact that there is no way to change the size of the caps on the error bars.) Here's the result on the left, with the graph on the right showing the series overlapping (for which the code appears immediately below).

title height=1.5 "Means and SDs, series overlapping";

ods graphics / reset width=16cm height=20cm imagemap attrpriority=none;

\*proc sgplot data=meantime1 noborder uniform=all;

\*by Set;

\*unstar the above two lines and star off the next to get separate graphs for animation;

proc sgplot data=meantime1 noborder;

styleattrs

datacolors=(black blue red green)

datalinepatterns=(solid)

datacontrastcolors=(black)

datasymbols=(circlefilled squarefilled diamondfilled trianglefilled);

\*reg x=Rep y=TimeMean /degree=1 nomarkers lineattrs=(thickness=1) group=Set;

\*scatter x=Rep y=TimeMean /yerrorupper=MeanPlusSD yerrorlower=MeanMinusSD

errorbarattrs=(color=black) groupdisplay=cluster clusterwidth=0.4 group=Set

filledoutlinedmarkers markerattrs=(size=18) name='abc';

\*unstar the above two lines and star off the next two lines to fit regression polynomials;

scatter x=Rep y=TimeMean /yerrorupper=MeanPlusSD yerrorlower=MeanMinusSD

errorbarattrs=(color=black) groupdisplay=cluster clusterwidth=0.4 group=Set;

series x=Rep y=TimeMean /markers filledoutlinedmarkers markerattrs=(size=18)

lineattrs=(pattern=solid) groupdisplay=cluster clusterwidth=0.4 group=set name='abc';

keylegend 'abc' /title="Set:" noborder titleattrs=(size=16) valueattrs=(size=16);

xaxis label="Sprint number" labelattrs=(size=16) valueattrs=(size=16)

values=(1 to 6 by 1) offsetmin=0.1 offsetmax=0.06;

yaxis label="Sprint time (s)" labelpos=top labelattrs=(size=16) valueattrs=(size=16);

\*refline 0;

run;

ods graphics / reset;

You wouldn't necessarily use the symbols and colors I have chosen for publication, but you can change them easily in the code. Even the SD bars on every point aren't necessarily appropriate; separate single stand-alone bars representing between- and within-subject SDs might be better here. However, I have provided the code as a template for plotting data that do require SD bars.

1. You can fit regression lines with the reg statement shown, but I have starred it off here. Why? Because the regression line would be based on all six reps, but the last rep in each set is clearly off the linear trend. Another option I have starred off is refline 0, which would make a horizontal "reference" line at the value of 0 on the Y axis; this option is useful when you are plotting change scores.
2. To go from SAS Studio to Powerpoint, it is possible in principle to use an ods powerpoint command. Unfortunately it produces a plot that has an unpredictably modified aspect ratio, and the image itself can't be modified. So you have to save the output as an RTF file…

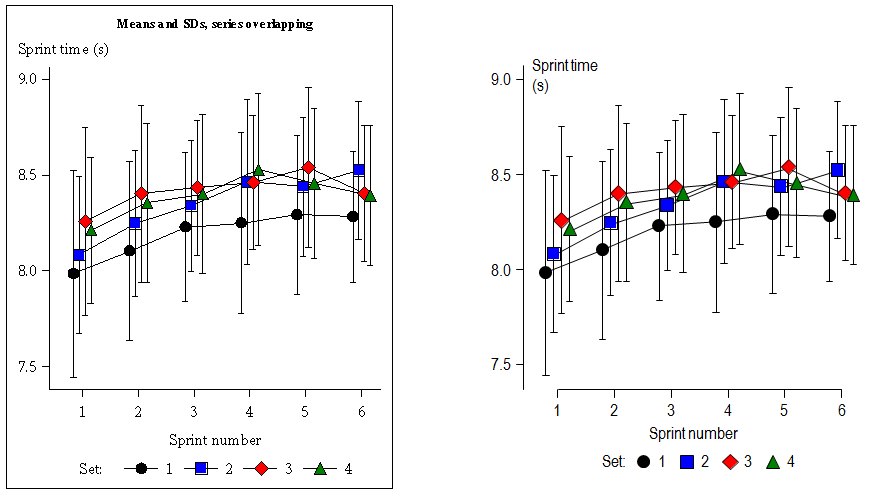


…then copy the graph to the clipboard, open Powerpoint, Paste Special/Picture (Windows Metafile), resize to a half-column width (~85 mm) for publication, or keep it at its existing size for a slide. Ungroup twice. To facilitate selection and editing of graph elements, select and clear all the invisible white backgrounds. Then by judicious selecting (usually via click-drag or shift-click), right-justify the text for the tick labels on the Y axis, center-justify the text for the tick labels on the X axis, and so on; next, change all text to Arial Narrow 10 pt, select all the tick labels on the Y axis and move them so they line up properly with the ticks; ditto the tick labels on the X axis; ditto the text for the key legend. This kind of editing is a lot easier if you customize your toolbar in Powerpoint by adding appropriate icons and deleting those used infrequently:

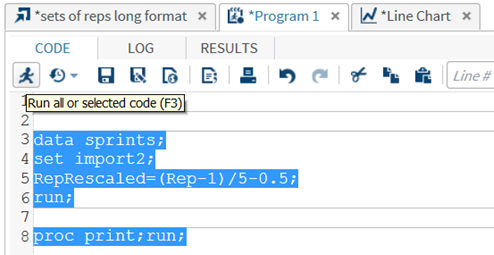




Here's the graph as it appears in the RTF file on the left, and after cleaning up in Powerpoint on the right. Notice that some blue squares aren't completely filled. You can fix manually, if necessary.



1. If you want to animate the individual series in a Powerpoint slide, you will need to group each series with its error bars. That would be almost impossible to do in the overlapping plot, so the trick is to produce a separate plot for each series, save as RTF, paste each of them into a separate Powerpoint slide, ungroup without changing the size, then select each series, group it, copy it, paste it into the first series, and then animate each series. See the comment near the beginning of the above code.
2. Save the above code as **plot sets of sprints.sas** or similar, then open a new SAS program. Now, back to the mixed model… For reasons that will become clear, we need a new Rep variable that goes from -0.5 to 0.5 rather than 1 to 6. We could do that in Excel, but let's do it with a data step in SAS. Click on the Program 1 tab, enter the following code after the proc print (the crucial line is RepRescaled=(Rep-1)/5-0.5; copy it from here, if you like). Make sure you use the right name for your imported data. Highlight it and run it:



Check that the values do indeed go from -0.5 to +0.5.

1. Now let's build the model. For those of you who already might like to think in code, here's what we'll end up with (minus the unnecessary bits):

ods noproctitle;

ods graphics / imagemap=on;

proc mixed data=sprints plots=(studentPanel) alpha=0.1 nobound;

class Athlete Set;

model LnTime=Set Set\*RepRescaled xVarRep6/solution cl alpha=0.1;

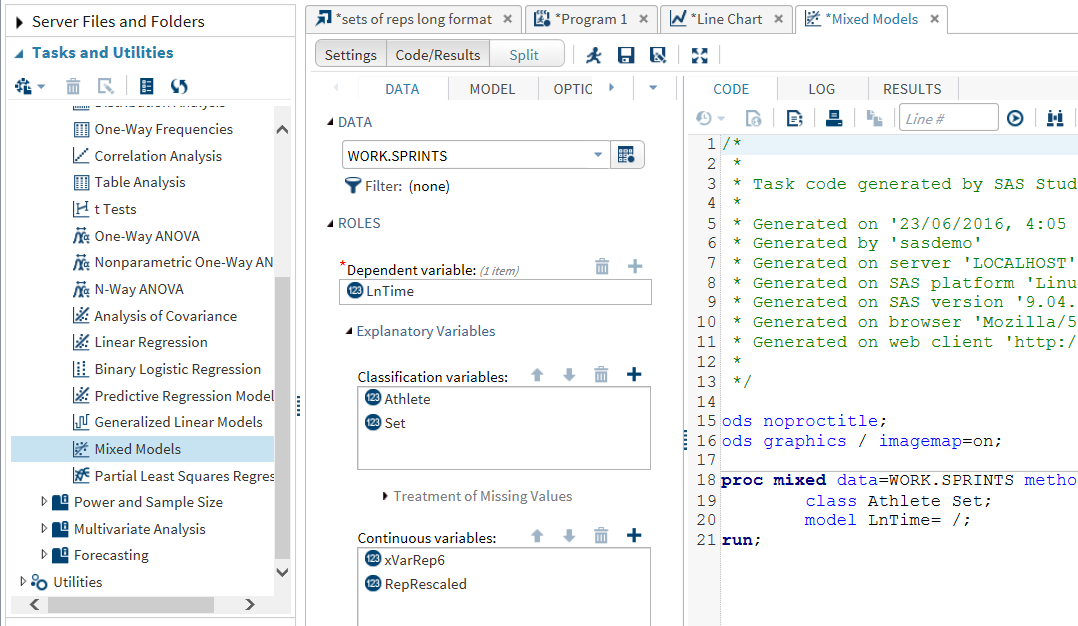
random Intercept RepRescaled/type=un subject=Athlete;

random Set xVarRep6/subject=Athlete;

lsmeans Set/diff alpha=0.1 cl;

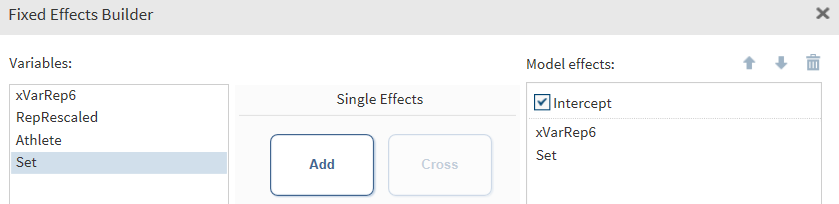
run;

1. Go to Tasks and Utilities/Tasks/Statistics, double-click Mixed Models, and choose the following DATA and ROLES:

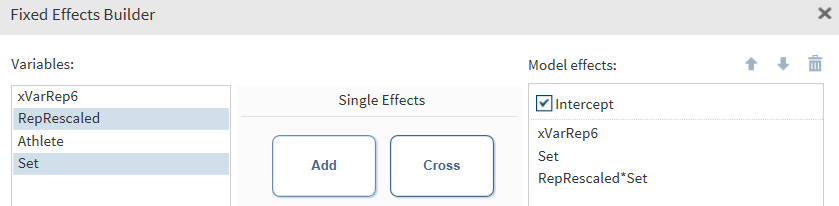


LnTime is the dependent. Athlete is a class variable, because it will be a random effect. Set is a class variable, because we'll look at the mean for each set. RepRescaled in a covariate, because fatigue usually develops in a linear fashion with repeated sprinting, and we want an overall estimate of what I call linearized fatigue from first to last sprint. That is, we will estimate the linear component of fatigue over six sprints. If we had used Rep rather than RepRescaled, we'd have to multiply its coefficient by 5 (yes, 5, not 6) to get the overall fatigue in six sprints. OK that's not a big deal, but it makes the individual differences in the slope a lot easier to interpret, as we will see. xVarRep6 is a dummy continuous variable that will estimate the extra effort the athletes make on the last sprint of each set.

1. Click on the MODEL tab, Edit the Fixed effects, add xVarRep6 and Set…

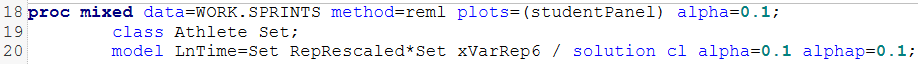


…then add RepRescaled\*Set by highlighting both variables (click, then Ctrl-click) and clicking Cross:



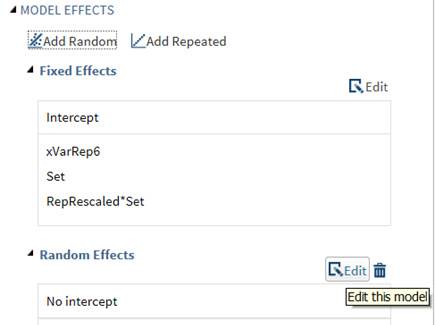
The order of these terms doesn't matter. SAS provides up and down arrows to change the order, so I moved xVarRep6 down to the bottom (to deal with its estimate last). Once again the intercept doesn't matter. Let's include it this time. Then click OK.

1. Here's the model so far:

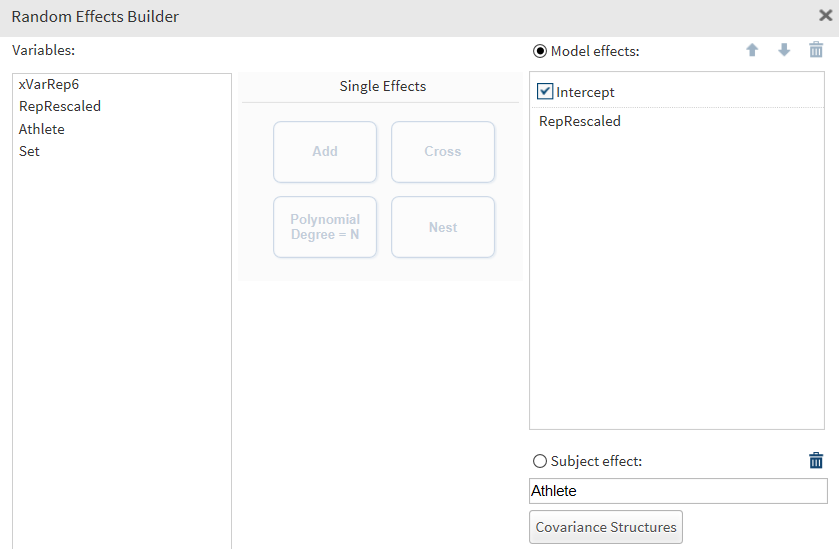


This model implies we're going to estimate a different mean for each set, a different mean slope for each set, and a different mean time for the last rep averaged across all the sets. Because we're already estimating mean time in each rep in each set with the other two terms, the effect for this dummy will be the amount by which the athletes on average go faster than a linear fatigue model predicts for the last rep.in each set.

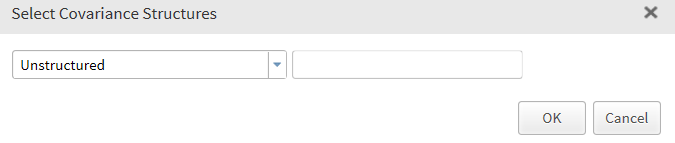
1. Click Add Random, Edit…



…Add Athlete as a Subject effect. RepRescaled as a Model effect, tick the Intercept…



…then click on Covariance Structures, select Unstructured, and click OK…



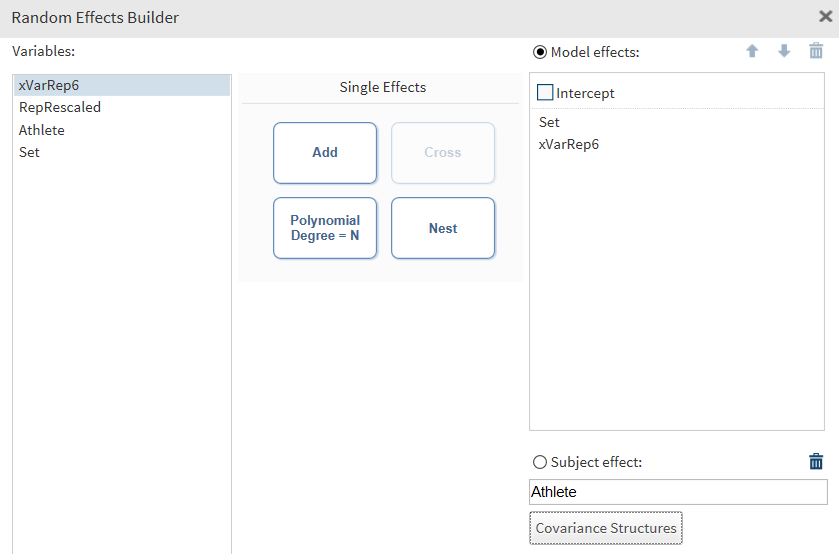
…and finally OK for the random effect. Check that the CODE window now shows this random effect:



This random-effect model generates a different random value for athlete (they differ in how fast they sprint), and a different random value for Athlete\*RepRescaled (they differ in how much they fatigue). How cool is that?! "Unstructured" allows these two effects to be correlated: faster athletes might fatigue more, for example. It's called an *individual-slopes* model.

1. But wait, we need another random effect. Click Add Random, and scroll down to Edit it…

Add Athlete as a Subject effect, add Set and xVarRep6 as Model effects, but do NOT include an intercept…

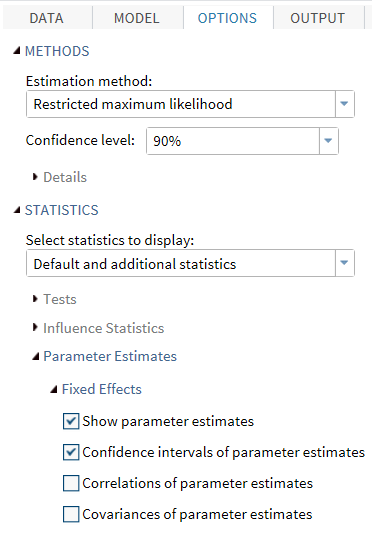
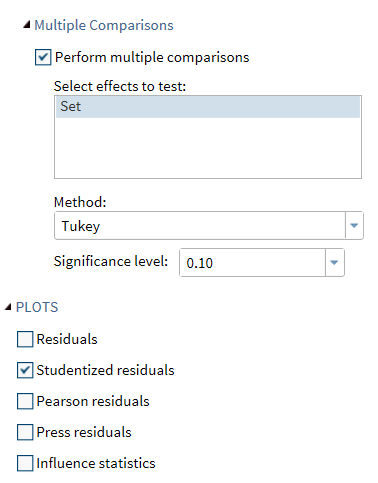


…then click OK. Check that you now have these two random effects:

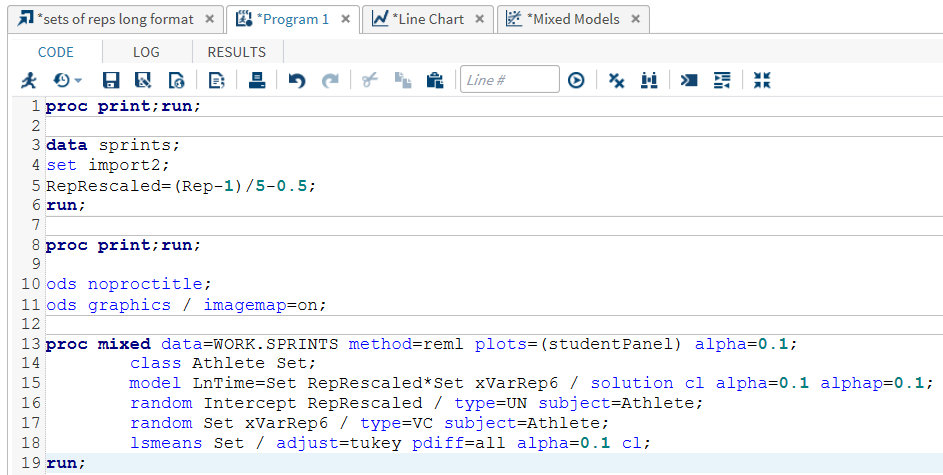


Set, with /subject=Athlete, means the same as Athlete\*Set; it represents the variability in an athlete's mean time between sets, which might not be much here, but it would be more if there were days rather than minutes between sets. xVarRep6, which means Athlete\*xVarRep6, represents individual differences (consistent over the 4 sets) from the mean value provided by the fixed effect for xVarRep6.

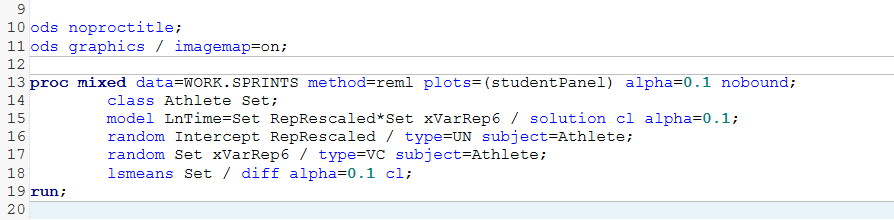
1. Click the OPTIONS tab and select the following:

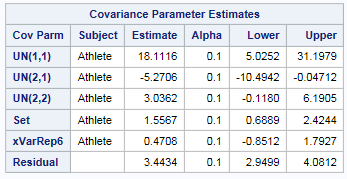
1. We're almost ready to run the analysis. We should allow for negative variance, so copy all the code to Program 1…



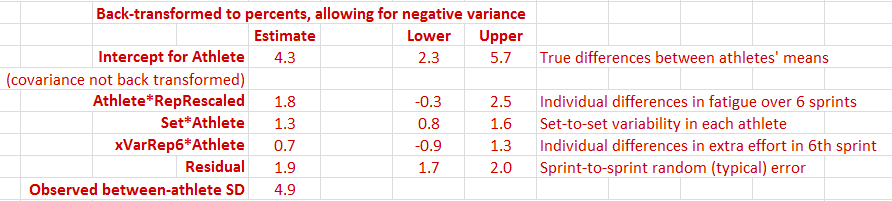
…and add nobound to the proc mixed statement. Delete alphap=0.1 from the model statement, and replace adjust=Tukey pdiff=all with diff in the lsmeans statement:



1. Now highlight the ods statements and all the proc mixed code, then click the Run icon.
2. Let's deal with the random effects first, because they also provide us with magnitude thresholds for interpreting their magnitude and the magnitude of the fixed effects. I also check the random effects first, because that's where I make the most mistakes! Here they are:



They need to be square rooted and back-transformed. Open **convert covparms to SDs.xlsx** and click on the tab **Sets of reps**, where it's done for you. I've also changed their names a bit:



**UN(1,1)** is the way SAS and other stats packages label the variance of the first random effect in the list of effects that are included in the unstructured set of effects. Here that's the athlete random effect. Think: each athlete gets their very own unique number out of this hat, representing how much faster or slower they are than the average.

**UN(2,2)** is the variance for the second in the list, here Athlete\*RepRescaled. Think: RepRescaled is a slope, and a different value has to come out of the hat for each athlete. It therefore represents individual differences from each of the mean slopes in the four reps. Note that we are estimating a single consistent difference from each of the four different means. It would be possible to estimate a different individual difference for each rep, but what's the point? We want some indication of how consistently an athlete differs from other athletes across all four reps.

**UN(2,1)** is the covariance of these two effects, basically a term like a variance indicating how much they go together. (It could also be written UN(1,2).) When it's negative, as here, it means they go in opposite directions: the faster an athlete, the bigger the drop-off.

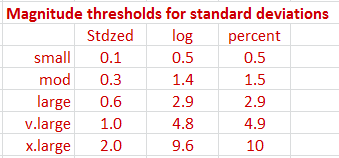
**Set [subject = Athlete]** is the Set\*Athlete random effect. Think: if you change the athlete or you change the set, another random number has to come out of this random-effect hat, so this represents the set-to-set variability an athlete shows in the mean of the athlete's reps of each set. This is a classic within-subject random effect when you have clusters of repeated measurements (here sets of repeated sprints).

**xVarRep6 [subject = Athlete]** is the xVarRep6\*Athlete random effect. Think: xVarRep6 is zero for all but Rep 6, when it is 1, so each athlete gets a single value out of this hat, and it represents the athlete's way of dealing with Rep 6, either faster or slower than the mean reduction we've seen earlier.

**Residual** is the random error that accompanies every sprint. Think: for every sprint, a number has to come out of this hat. It's analogous to the typical error in a straightforward reliability study. By the way, the residual is equivalent to a random effect specified by Athlete\*Set\*Rep.

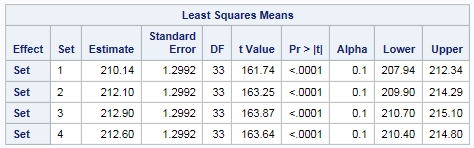
1. To properly assess these SDs, we need the smallest important difference or change. For team-sport athletes (which these were), the default for differences in means is 0.2 of the *observed* between-athlete SD in any given testing occasion. This SD is given by adding up all the variances that contribute to a single sprint in any given testing session; here it is the variance for the stable (or true or pure) between-athlete differences = UN(1,1), plus the variance an athlete shows in any given set = Set\*Athlete, plus the variance an athlete shows in any given sprint = Residual. After back-transformation, the value is 4.9%, as shown in the spreadsheet. (Subtle issue… With RepRescaled deliberately having a value of 0 in the middle of the set of sprints, the individual responses in the fatigue don't contribute to the SD in the middle of the sprints, but they would on either side. The negative covariance means that athletes with faster sprints have greater fatigue, so that apparently means there would be a bigger observed between-subject SD at the start of the sprints and a smaller one at the end. The SD in the middle is therefore a reasonable value to use, and 0.2 of 4.9% is 1.0%.)

The other thresholds for differences in means as fractions or multiples of the between-athlete SD are 0.6, 1.2, 2.0 and 4.0 for moderate, large, very large and extremely large. But wait! You have to double SDs to interpret their magnitudes, or equivalently halve the thresholds. I have halved the thresholds (to 0.1, 0.3, 0.6, 1.0 and 2.0), and calculated the corresponding log and percent effects, as shown in the spreadsheet:



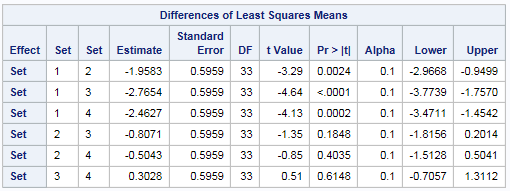
You don't evaluate the Athlete random effect or the UN(2,1), but you evaluate everything else. I can see that the observed effects range from small to moderate, and only the xVarRep6 is unclear. I don't normally do a full probabilistic magnitude-based inference on an SD. I think it's enough to interpret first the observed value, then the upper confidence limit to see how big it could be, then the lower confidence limit to see if it could be trivial or negative. Some negative SDs occur simply because of sampling variation when the true value is actually trivial or small. Other negative SDs can represent real negative variance, especially individual responses to treatments. See later.

1. The least-squares means for Set are the means you would want to use, if you were to report them…

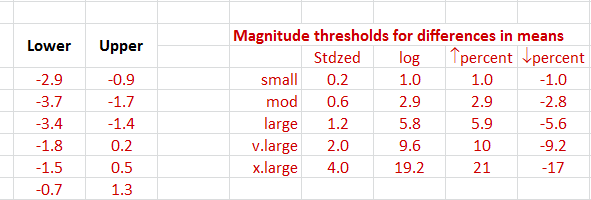
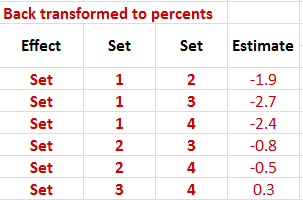


They back-transform to means of exp(210.1/100) = 8.18, 8.34, 8.41 and 8.38 s. See the spreadsheet.

1. The pairwise comparisons are more interesting…



These are all <3%, so they don't change much following back-transformation. I've included cells to show magnitude thresholds for differences (here *changes*) in means:

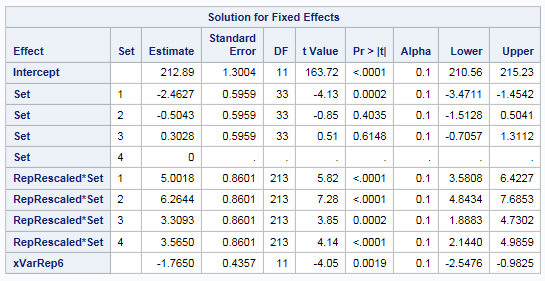


Note that with percents, the thresholds for reductions are not exactly minus the thresholds for increases. Click in the cells to see what's going on.

You can see that the observed magnitudes are trivial to small. You can see that there was 2.0% fatigue between the first and second set, then another 0.8% (trivial) between the second and third, but they revived a little (trivially) in the fourth set, by 0.3%. With a smallest important of 1%, all of these effects would be clear at different levels of likelihood for trivial or substantial.

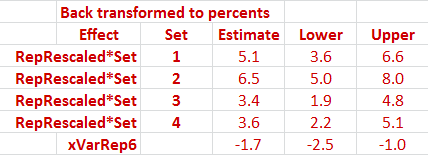
If they were interesting enough, you could do full magnitude-based inference on these effects via the log-transformed mean difference and its p value with the [Confidence limits & clinical chances](http://www.sportsci.org/resource/stats/xcl.xls) spreadsheet at Sportscience. The smallest important log-transformed value goes into the cell under "benefit or ???". Check that the spreadsheet gives the same confidence limits as shown here. If it doesn't, you've made a mistake. You can also do the MBI via the mean difference and its confidence limits with the [Combine/compare effects](http://www.sportsci.org/resource/stats/xCombineGroups.xls) spreadsheet. Click on the tab at the bottom of this spreadsheet to select the sheet for **1 or more groups or statistics**, and use a single custom weight with a value of 1.

1. Now the fixed effects…



Theestimates for Intercept and Set show what happens when you have a nominal fixed effect and leave the intercept in the model. The intercept gets assigned the last level of the nominal, and the estimates of the other levels of the nominal are the differences from this level. Which is OK, if you want estimates of these differences, but what about all the other differences? See the least-squares means and their differences.

Here are the back-transformed values and confidence limits for the other effects, from the spreadsheet:



**RepRescaled\*Set 1, 2, 3 and 4** etc. are the fatigue slopes evaluated over the six sprints, and they are small enough to be almost exactly percent increases in time: 100\*exp(5.00/100)-100 = 5.1%. So there is a ~5-6% fatigue in the first two sets, but only around 3.5% in the last two.

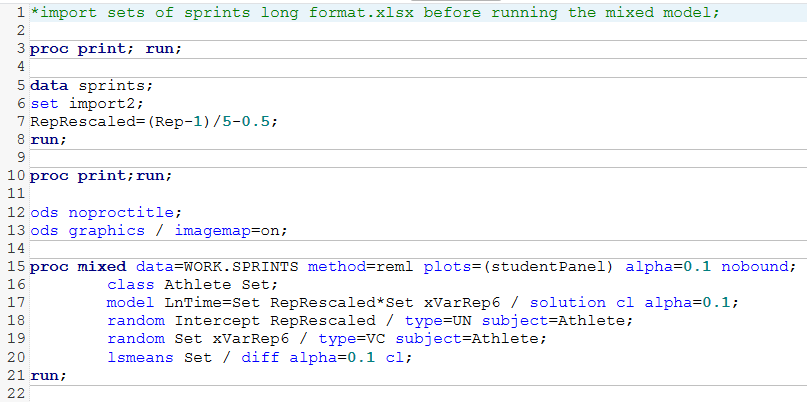
**xVarRep6** is -1.7%, which means the athletes went faster than predicted in the last rep overall by 1.7%.

With the smallest important difference or change in sprint time of 1.0%, the fatigue effects and the faster effect in the last rep are all clearly substantial. The 6.3% is actually large.

1. Let's modify the program a bit before saving it. Just insert this comment in the first line:

\*import sets of sprints long format.xlsx before running the mixed model;

Here's the final program:



1. Now save the program as **sets of sprints.sas**, close all the tabs and Sign Out. (Any tabs you leave open will re-open next time you start SAS.)

This program and the others we developed are all available in **all reliability programs SAS.docx.**

**Using "Nesting" to Specify Repeated Measurements**

1. There is an alternative way of stating the clustering of repeated measurements within subjects. To make it simple, imagine that you have some repeated measurements within each of the four sets in the above dataset, and there is no fatigue or anything else happening with the reps: they are just repeated measurements to improve the precision. Here are the model and random statements using the same approach as above:

model LnTime=Set/solution cl alpha=0.1;

random Intercept/subject=Athlete;

random Set/subject=Athlete;

Or the even simpler versions…

model LnTime=Set/solution cl alpha=0.1;

random int Set/subject=Athlete;  
…or…

model LnTime=Set/solution cl alpha=0.1;

random Athlete Athlete\*Set;

1. The alternative way is to imagine that within each athlete there are four measurements of Set; that is, Set is clustered or *nested* within Athletes. Here are the random effects for the simple model:

random Intercept/subject=Athlete;

random Intercept/subject=Set(Athlete);

And here are the random effects for the previous fatigue model:

random Intercept RepRescaled/type=un subject=Athlete;

random Intercept/subject=Set(Athlete);

random xVarRep6/subject=Athlete;

The output of the random effects looks slightly different, but the values of the covparms are identical.

I find the "nesting" approach and terminology confusing, but if it works for you, go for it.