Paper CT11

A programmer's introduction to modelling with proc mixed, focusing on how to debug and validate, common mistakes and how to interpret the output

George Gibson, AstraZeneca, Cambridge, England

ABSTRACT

A lot of complex analysis in the pharma industry relies on the proc mixed statement to create models that are used to estimate and predict key endpoints in studies. Programmers may not have a mathematical background and so the concept of modelling can appear daunting, with help provided via forums and SAS itself often using a lot of mathematical terms that won't help a stuck programmer.

When using proc mixed a lot of things can go wrong such as adding incorrect weighting on covariates, adding the covariates in the wrong order and even more complex issues such as the model not converging.

In this presentation I aim to go over the key elements of the proc mixed statement, how to create model estimates including a brief discussion on how to create 2- and 3-way model interactions, some debugging tips/common mistakes programmers make and finally how to interpret the model outputs.

Introduction

Proc mixed is an incredibly powerful and flexible tool that is crucial in the world of statistical analysis. To an inexperienced programmer, or someone who doesn't come from a mathematical background, the concept of proc mixed can appear daunting and simple mistakes can arise. These mistakes however are often very easily fixed and with the right experience, can easily be spotted. Within this paper I intend to go over the definition of proc mixed, discuss some of the key syntax and summarise what they mean, how to write estimate statements and then how to break down and manage interaction terms. Throughout the paper there will be some key things to consider when programming with proc mixed and helpful tips that should aid both in the creation of proc mixed code and the validation of it.

What is proc mixed?

Proc mixed is used within SAS to create mixed linear models which are useful in many common statistical analyses. Mixed linear models are incredibly useful and can be used to model both fixed and random effects, with a fixed effect being a driver that we know will have a certain impact (such as which treatment arm a subject is on) and a random effect being drivers that might have a small, unpredictable impact that should be considered (such as site location).

Syntax

Within SAS, proc mixed has several statements that can be used to customise your model, ranging from a by statement that groups based on a pre-defined variable and creates an output for each group to the weight statement which allows the user to add weighting to each covariate in your model. The statements that I will be focusing on are:

- By: used to obtain separate analysis based on a grouping variable
- Class: Defines the categorial variables within your model
- Model: Defines the model itself (including the fixed effect)
- Random: Defines the random effects
- Repeated: Defines a time variable where data is collected. This is used to analyse data collected at multiple time points or under different conditions for the same subjects, helping us understand how measurements change over these repeated observations.
- Estimate: This is used to generate model estimates for specific "scenarios" of interest (through weighting)
- Lsmeans: Creates least squares means of the fixed effect. This is a way of finding a "balanced" average that gives each group a fair representation in the overall average ie it will put equal weighting on each of the model covariates.

When all the above statements are combined, you would get a chunk of code that looks like this:

```
proc mixed data=input;
  by imputenm;
  class usubjid trt01pn avisitn;
  model chg = trt01pn avisitn trt01pn*avisitn;
  repeated avisitn / type=un subject=usubjid;
  lsmeans trt01pn|avisitn/ pdiff cl;
Run;
```

Breakdown

This mixed model is looking at what effect the treatment arm, visit and the interaction term between treatment arm and visit have on a subject's change from baseline (as shown by the model line). Expanding on this, the by statement tells SAS that we would like to run the analysis for each group of imputation and that the variables usubjid, trt01pn and avisitn are all categorical. Note that the class line is very important, especially when dealing with numeric variables. By default, SAS will assume that any numerical variable listed in the model statement is continuous and so will only produce one estimate in the output. Since, in this example, we would like our variables to be categorical (and thus have an estimate for each level of the variable) we need to tell SAS that it should be treated as such via the class statement.

The repeated statement tells SAS that avisitn is a repeated variable, and so, our analysis should account for the fact that multiple measurements are taken for the same subject (specified by the subject= option). Within this line we also include the type=un option, this tells SAS what type of convergence we would like to use, and this information should be found in your SAP (if it isn't then speak to your statistician).

Finally, the lsmeans statement tells SAS that we would like to produce least squares mean estimates for the interaction term between treatment and visit. The extra options pdiff and cl tell SAS that within this output we would also like to include the differences between these means with confidence intervals.

Within the model creation step there are several things that a programmer should consider and be aware of. These are often little things that can easily be overlooked, but if they aren't considered issues could arise when it comes to validating your model, especially if your validator has considered them. These include:

- Your input dataset having a different sorting order to a validator. This can cause the model to produce different estimates and so, when creating models, make sure the input datasets are identical between the developer and the validator.
- Numerical variables that should be considered as categorical, not continuous, aren't defined in the class statement.
- The order of the variables in the model statement are different, this is especially important when using interaction terms, the details of why this is can be found in the 2-way/3-way interaction section.
- Warnings appearing in your log relating to convergence. This is often resolved by changing the type option and your study statistician should be able to help with this
- Weighted vs estimate. As mentioned, the lsmeans statement will produce estimates that assume equal weighting on all covariates. This is rarely true and so estimate statements are often used to allow for greater flexibility with variable weighting. This issue can be resolved by using a weighted dataset but that will not be covered here.

Now that we know the basic syntax of proc mixed, lets focus more on estimates, interactions, and interpretations.

Estimates

Once your basic model has been setup, you can expand it further by adding estimate statements that allow you to generate model outputs for specific scenarios of interest. Sticking to our basic model that looks at treatment arm and visit we could model:

- Change from baseline for Drug 1 at Visit 3
- Change from baseline for Drug 1 over Visit 3
- Change from baseline at Visit 4: Drug 1 vs Drug 2

To do this, you need to add weighting to each of the model variables. For continuous variables (such as height, weight etc) you only need to add one weighting value and this would commonly be the average. For categorical variables, you need to add a weighting to each level of your variable, with the level being how many unique values the variable has. The weighting on a categorical variable must add up to 1 unless you are looking at the difference between 2 levels, then it must sum to 0.

An example of how to do this is below. For this example, our categorical variables are trt01pn (treatment arm), avisitn (visit) and race and these have 4,5 and 2 levels respectively.

```
Estimate 'Treatment 1 at Visit 3'
       height &average_h.
       weight &average_w.
       race 0.72 0.28
       trt01pn 1 0 0 0
       avisitn 0 0 1 0 0 / CL;
```

Here we have used the average height and weight for our continuous variable weighting, these have been created in a previous step and placed in a macro variable. For the categorical variables we have assigned a weighting of 0.72 for the first level of race and 0.28 for the second. While these numbers chosen were arbitrary, it is best practise to follow the approach used for the continuous variables, that is use the proportions seen in your study. The best way to do this is to use a proc means to calculate the proportions of the categorical variables levels and then place them into a macro variable. If you are using proc means (or any other method to derive the weighting) then the level of precision should be consistent between the developer and the validator. If the developer uses 2 decimal places and a validator uses 3, then the model estimate will be slightly different and your proc compare won't be clean. Another item to consider, especially if you are using proc means to generate your weightings, is to ensure that a subject only has one observation in the proc means dataset. If the number of observations within the dataset varies from subject to subject, then using proc means will not generate a true proportion for each variable.

Notice that for the final two variables (trt01pn and avisitn) we have assigned several levels the weighting of 0. This is because, as denoted by the label seen on the estimate line, we are only interested in the value of treatment 1 at visit 3.

If, instead of looking at the estimate for treatment 1 at visit 3, you wanted to look at the estimate of treatment 1 over visit 3 (ie the effect at visit 1, 2 and 3) you would use the below syntax.

```
avisitn 0.333333333 0.333333333 0.333333333 0 0 0
```

Note that this breaks our earlier rule of summing to 1. This is fine as in the background SAS knows you are trying to apply a 1/3 weighting to each of the levels and so will not give you a warning.

Finally, if you wanted to look at the difference between treatment 1 and treatment 2, you would use this syntax noting that this sums to 0 and not 1:

```
trt01pn 1 -1 0 0
```

Interactions

The final part of this paper will focus on interactions and will provide some tips on how to best write them. An Interaction term explores the combined effect of two or more variables on our outcome. So, because of the inclusion of the interaction term, we can explore how the effectiveness of the treatment changes over time. It helps us to understand if the treatment has a different effect on patients depending on whether they are early in their treatment journey or further along.

The syntax for an interaction term is commonly denoted using an * in the model statement between the two variables we would like to see the interaction effect of, as shown in the proc mixed example.

When creating estimate statements for interaction terms it is important to note that SAS is expecting a string of x*y length, where x is the number of levels of the first variable of the interaction term and y is the number of levels in the second one. So, for our previous examples, if we are modelling the interaction term between treatment arm and visit, SAS is expecting a string 20 characters long.

```
Trt01pn*avisitn 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

But where do you put your weighting values? If you want to estimate the effect of treatment 1 at visit 3, how do you know where to put the 1 in this interaction term string. The simplest way to do this is to break the string down into blocks. SAS reads this string as 4 blocks of 5 (since treatment arm was the first variable defined in the model, it is the variable that creates the blocks), so we can break this string down into the following:

```
Trt01pn*avisitn 0 0 0 0 0   0 0 0 0 0   0 0 0 0 0   0 0 0 0 0
                Treatment 1 Treatment 2 Treatment 3 Treatment 4
```

Within these blocks we can then clearly see 5 values, with each one corresponding to the visit of interest. So, if we want to model the effect of treatment 1 at visit 3, we should put our weighting value in the third space of the first block:

```
Trt01pn*avisitn 0 0 1 0 0   0 0 0 0 0   0 0 0 0 0   0 0 0 0 0
                Treatment 1 Treatment 2 Treatment 3 Treatment 4
```

The most important thing to consider when using interaction terms is that the order is defined by the order of the variables in the model statement, not in the interaction term. This is best described using the below example.

```
class trt01pn avisitn;
  model chg = trt01pn avisitn height weight trt01pn*avisitn;
Estimate 'Treatment 1 at Visit 3'

Trt01pn*avisitn 0 0 1 0 0   0 0 0 0 0   0 0 0 0 0   0 0 0 0 0
                Treatment 1 Treatment 2 Treatment 3 Treatment 4
```

```
class trt01pn avisitn;
  model chg = avisitn trt01pn height weight trt01pn*avisitn;
Estimate 'Treatment 1 at Visit 3' 'Treatment 3 at Visit 1'

Trt01pn*avisitn 0 0 1 0   0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0
                Visit 1   Visit 2   Visit 3   Visit 4   Visit 5
```

From this, we can see that the first model has trt01pn first while the second one has avisitn. When it then comes to writing the model estimates, this fundamentally changes the way SAS interprets the string. Instead of looking for 4 blocks of 5 (since trt01pn is the variable that creates the blocks) it is looking for 5 blocks of 4 (since the blocks are now defined by avisitn). If we placed the 1 in the 3rd position, we then aren't modelling the estimate of treatment 1 at visit 3, but instead we are modelling the estimate of treatment 3 at visit 1. This is why the ordering of the variables in the model statement is so important, it can change the way SAS interprets your interaction terms and you might not be estimating what you intended to.

In very rare circumstances, you may be asked to model 3-way interactions and the process of simplifying it is very similar. SAS is now expecting an x*y*z string (where z is now the number of levels of your third interaction term variable). So, if we add race (which has 2 levels) to our interaction term, SAS is expecting a string of 40 characters.

```
Trt01pn*avisitn*race 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

Like with 2-way interactions, this can easily be broken down. First, instead of breaking it down into x blocks, we instead break it down into x rows:

```
Trt01pn*avisitn*race 0 0 0 0 0 0 0 0 0 0
                     0 0 0 0 0 0 0 0 0 0
                     0 0 0 0 0 0 0 0 0 0
                     0 0 0 0 0 0 0 0 0 0
```

4 rows, one for each treatment

This can then be broken down into y blocks:

Each of which contain 2 characters (one for each level of our race variable). So now we can clearly see that if we wanted to estimate the effect of treatment 1, at visit 3 for race 1, our weighting value should go in the 1st position of the 3rd block on row 1:

```
Trt01pn*avisitn*race 0 0   0 0   1 0   0 0   0 0
                     0 0   0 0   0 0   0 0   0 0
                     0 0   0 0   0 0   0 0   0 0
                     0 0   0 0   0 0   0 0   0 0
```

You can also create estimates for over a visit range (such as over visit 3) using the same rules as before:

```
Estimate 'Treatment 1 over Visit 3 for race group 1'
…
Trt01pn*avisitn*race 0.333333 0   0.333333 0   0.333333 0   0 0   0 0
                     0 0          0 0         0 0          0 0   0 0
                     0 0          0 0         0 0          0 0   0 0
                     0 0          0 0         0 0          0 0   0 0
```

Or even for treatment differences:

```
Estimate 'Treatment 1 vs Treatment 2 at Visit 3 for race group 1'
…
Trt01pn*avisitn*race 0 0   0 0   1 0   0 0   0 0
                     0 0   0 0  -1 0   0 0   0 0
                     0 0   0 0   0 0   0 0   0 0
                     0 0   0 0   0 0   0 0   0 0
```

Conclusions

Proc mixed is an incredibly powerful tool and while mistakes can happen, these are often simple things that a programmer may not think to consider. If you are struggling to validate your model, then a few basic things to check are:

- Your datasets are sorted.
- Model statements have the same order.
- Weighting of covariates in the estimate statement have the same degrees of accuracy.

Interactions can add a new level of complexity to programming, but if you break it down using the tips above then a programmer should have no problem with creating 2-way and even 3-way interaction terms, if they keep their code clean and organised.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Author Name: George Gibson

Company: AstraZeneca

Email: George.gibson1@astrazeneca.com