Yevgeniya Okuneva
STAT 242
Final Project

Power and Sample Size Estimation for 3-level Mixed Effects Models

Introduction

The main objective of this project is the power and sample size estimation for a 3-level mixed effects model using fake-data simulation. My first thought was to make the functions very general so that we can readily use them for any 3-level mixed effects model; however, every model is so unique that it made more sense to master the ideas and methods behind the approach to be able to adapt the functions to other models and designs when needed. The nature of the process makes the computations embarrassingly parallel, therefore, I looked into executing these functions in parallel on 8 cores on the laptop used for the completion of the project. For the comparative purpose, the computations are also done using traditional iterative approach on a single machine/core.

Approach

As was already mention in the previous section,the estimation is done using fake-data simulation. The idea is to first run the model on the pilot data to get the estimated fixed and random effects. Next, the new data set is generated n times based on these estimates. The model is applied to each of the generated data sets followed by the contrasts of variables of interest. The coefficient estimate and the p-value are extracted from each contrast object and saved in the matrix. After all n iterations (data generation, model, contrast) are completed, the number of significant p-values is counted, divided by the number of iterations (n) and multiplied by 100. The obtained number is the power, in other words, it is the percentage of correct rejections of null hypothesis when the null hypothesis is false (the probability of not making Type II error), assuming we know that there is a true effect and the null hypothesis should be rejected. All of the above is discussed more thoroughly in the subsequent sections.

Pilot Data and Model

lme() function from NLME package was used to fit the models. Contrasts were obtained with the contrast function from the CONRASTS package that excepts and knows how to deal with “lme” objects. The actual pilot data is not presented, instead the model is ran in a different session and the model estimated fixed effects coefficients are extracted and stored for future use. 3 types of errors are also extracted and stored. Please note that the functions for extracting the variance components were obtained from the web. Both (varRan() and varWithin()) were written by Jose Pinheiro, one of the developers of the NLME package and the co-author of the “Mixed-Effects Models in S and S-plus” book, therefore, the source is reputable enough(!) to reuse these functions instead of trying to rewrite them. The variance components are simply the within variance and the random effects at levels 1 and 2.

The modeled response variable is called “value”. Other variables included in the model as covariates are two continuous variables named “cov1” and “cov2”, and two categorical variables, one with 24 levels (points per day) and the other one with 3 levels. We also have random effect for day with nested random effect for the subject. Points per day are nested withing subjects. The data matrix for the continuous covariates is used in the “fake-data” generation process (cov.vals.csv file in the repository).

Note: the focus of this project is not on the model building so the autocorrelation and other diagnostic problems are not discussed and ignored for the particular exercise.

"simulateData()" function to generate data

First, 3 sets of errors are generated using the approach below. Please note that tau, sigma2, and sigma1 are obtained from Jose Pinheiro varRan() and varWithin() functions.

1. Subject-level random-effects are generated from the random normal distribution with mean = 0 and sd = sqrt(tau). N of them are generated where N is the number of participants, and then each is replicated 24*8 times (note from the previous step that we have points per day (24) nested within subjects (N) which in turn are nested within days (8).
2. Day-level random-effects are generated from the random normal distribution with mean = 0 and sd = sqrt(sigma2). N*8 of them are generated and then each is replicated 24 times.
3. Level-1 errors are generated from the random normal distribution with mean = 0 and sd = sigma1. N*8*24 of them are generated.

Next, data for first treatment called "one" is generated. Fixed coefficient for cov1 and cov1 values from the cov.vals.csv mentioned before, fixed coefficient for cov2 and cov2 values, as well as fixed coefficient for treatment "one" with coefficients for treatment "one" being replicated 8 times are all added together. This gives mean estimate at each time point throughout 8 days. The data set is replicated N times (sample size) after which the vectors of errors obtained earlier are added to the data. Similarly, we get the values for treatments "two" and "three".

Next , treatments are binded together and cov1 and cov2 columns are added. The factor variables are all added to the data set. Plots (not presented in the write up) were created to compare the pilot and the generated data to ensure the approach described above works well giving us similar to pilot data sets.

The "Power" Iterative Function

First, a matrix is preallocated with as many rows as the iterations planned and 2 rows, one for contrasts coefficients and the second one for p-values. Please note that the arguments to the simulateData() functions are the explicit values obtained from the model but they will be substituted with small functions getting these values directly from the model avoiding hard-coding.

Next, the model is applied to each data set. Try() function is used to make sure the process continuous even if the model encounters a problem. Another common problem with mixed-effects models is "Non-positive variance covariance matrix", and if that is the case, NA for the output is generated, if not, it goes on to the contrast function. The number of iteration is printed to keep track of where the functions is at. Please note that the p-value for each contrast is multiplied by 3 to correct for the number of contrasts (only 1 contrast presented but all possible treatment contrasts will be added in at a later point). Power is obtained as the sum of the number of significant p-values divided by the number of iterations.

First, the function is tested with N = 25 with 100 iterations. The time for how long the whole process took is recorded.

```
 user  system elapsed
269.133   0.668 269.980
```

Before going into the exploration of different sample sizes and power for those sample sizes the same process is repeated in parallel to compare speed to determine whether parallel computations take less time and if makes sense to run them in parallel.

The "Power" Iterative Function in Parallel

The function is very similar to the one described in the previous section. The only difference is in the way we store data. Instead of preallocating a matrix, we save the output as the list of contrasts from each iteration and then extract the variables of interest, the contrast coefficients and the p-values.

We use random number generator to make sure that each worker is independent and that the results are reproducible. As noted earlier, we use 8 cores present on the local machine using "forking". The cluster is created after coef.csv and cov.vals.csv as well as simulateData() function are loaded to make sure that each worker inherits what it needs to successfully do the job.

ClusterApplyLB() is used to make sure each worker is constantly busy in case some workers turn out to be faster then the other workers (though it may not be the case as all jobs are identical and don't differ in the level of difficulty, they just get different data to do the same job).

```
user  system elapsed
 0.824   0.988 217.352
```

Interestingly, clusterApplyLB() took much more time than non-parallelized computations. Leaving that for further exploration at some other point, clusterApply() was used instead.

The time it took to complete 100 iterations for N = 25 is

```
user  system elapsed
 0.784   0.768 250.158
```

clusterApplyLB() is slightly faster then the non-parallelized computations, however, this changes when the number of iterations increases (i.e. doubled).

Discussion

The main objective of this project was to build functions for sample size and power estimation for 3-level mixed-effects model. This part was successfully completed leaving us with the answer that a sample size of 25 subjects in each treatment should be sufficient. Parallel computations did not go as well and require further exploration (perhaps parApply() may work). Also, turning to snow package may be helpful as it provides nice methods for obtaining visual cues about what is happening between the master and each worker by plotting snow.time objects.