Variance Estimation - Second Level FSL

Han Bossier

15/6/2018

Contents

1 Introduction

2 Simulation

In this section, we discuss the data generation and the models we use to analyze the data. This will be using a mixed model approach with *lmer* (**R**), while the second approach is using the typical two-stage approach in fMRI and the mixed model *FLAME1* from **FSL**.

2.1 Data generation

Let us first define some global variables.

```r
# Location of raw data
RawDat <- '/Volumes/2_TB_WD_Elements_10B8_Han/PhD/Simulation/Results/Variance_2lvl/'
# During OHBM, I used the following path
#RawDat <- '/Users/hanbossier/Desktop/Results_VAR2LVL/'

# Number of batches simulated
IDs <- 200

# Number of simulations within a batch
numSimID <- 10

# Number of subjects
nsub <- 50

# Value for sigma in the model
sigma_eps <- 100

# Between subject variability (variance of random slope)
sigma_b2 <- c(0, 5, 10)

# Variance of random intercept
sigma_b1 <- c(0, 1, 1)
```

We generate time series for each subject for *one voxel*. We have 200 scans (T) with a TR of 2 seconds, a block design of 20 sec ON/OFF, a double-gamma HRF convolution and a true BOLD percent signal change of 3 percent. We generate T datapoints (Y) for each subject $i = 1, \ldots, N$ using the following GLM:

$$Y_i = (\beta_0 + b_{0i}) + (\beta_1 + b_{1i})X + \varepsilon_i,$$

with $b_{0i} \sim N(0, 1)$, $b_{1i} \sim N(0, \sigma_b^2)$ and $\varepsilon_i \sim N(0, \sigma_e^2)$. We see that we use the same design matrix X for each subject and we only generate Gaussian random (white) noise. Furthermore, we have $\sigma_b^2 \in \{0, 5^2, 10^2\}$ and $\sigma_e^2 = 10^2$. Finally, we have $\beta_0 = 100$.

```r
# Signal characteristics
TR <- 2
nscan <- 200
total <- TR*nscan
on1 <- seq(1,total,40)
onsets <- list(on1)
duration <- list(20)
```

```
###################
#### Generate a design: GROUND TRUTH DESIGN
###################

# true %BOLD change
BOLDC <- 3

# Base/intercept of signal
intcpt <- 100

#######################################
#### DESIGN AND SIGNAL TIME SERIES ####
#######################################

# Generating a design matrix: convolution of block design with double-gamma HRF
X <- neuRosim::simprepTemporal(total,1,onsets = onsets,
                               effectsize = 1, durations = duration,
                               TR = TR, acc = 0.1, hrf = "double-gamma")

# X vector for one subject = predicted signal
X_s <- neuRosim::simTSfmri(design=X, base=0, SNR=1, noise="none", verbose=FALSE)

# Now the model will be: (intcpt + b1) + (BOLDC + b2) * pred + epsilon

## Design parameters
# Extend the design matrix with the intercept
xIN <- cbind(intcpt, X_s)

# Contrast: not interested in intercept
CONTRAST <- matrix(c(0,1),nrow=1)

# Calculate (X'X)^(-1) with contrast
design_factor <- CONTRAST %*% (solve(t(xIN) %*% xIN )) %*% t(CONTRAST)
```

To generate random intercept and slopes, we first generate subject-specific values for b_{i0} and b_{1i} where we set $\sigma_b^2 = 5^2$ using a variance-covariance matrix:

```
var_cov_D <- rbind(c(sigma_b1[2]**2, 0), c(0, sigma_b2[2]**2))

# Generate the subject-specific values for intercept and slope using this D-matrix
B_matrix <- MASS::mvrnorm(nsub, mu=c(0,0), Sigma = var_cov_D)
```

Now looping over all subjects, we generate:

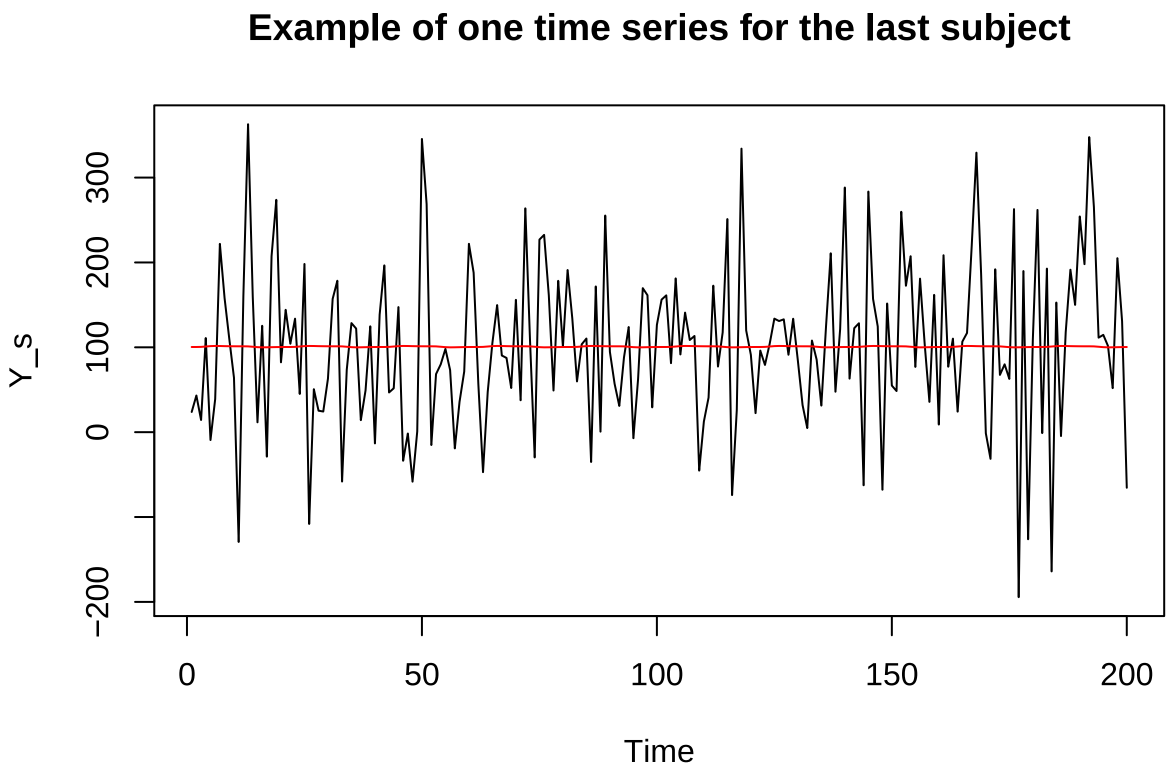
```
# Empty vector
Y <- data.frame() %>% as_tibble()

# For loop over all subjects
for(i in 1:nsub){
  # Generate nscan values, corresponding to time series of one subject
    # note: random intercept and random slope generated earlier
  Y_s <- (intcpt + B_matrix[i,1]) + ((BOLDC + B_matrix[i,2]) * X_s) +
    rnorm(n = nscan, mean = 0, sd = sigma_eps)

  # Add to data frame
  Y <- data.frame(Y = Y_s, X = X_s, sub = as.integer(i)) %>% as_tibble() %>%
    bind_rows(Y, .)
}
```

Let us plot the time series for one subject in one active voxel (red line is true response for this subject).

```
plot(Y_s, type = 'l', main = 'Example of one time series for the last subject',
     xlab = 'Time')
lines(x = ((intcpt + B_matrix[nsub,1]) + ((BOLDC + B_matrix[nsub,2]) * X_s)), col = 'red')
```



2.2 Models

The analysis of the time series of all N subjects using **R** is straightforward:

```
lmer(Y ~ 1 + X + (1 + X|sub), data = Y)
```

```
Linear mixed model fit by REML ['lmerMod']
Formula: Y ~ 1 + X + (1 + X | sub)
   Data: Y
REML criterion at convergence: 120482
Random effects:
 Groups   Name        Std.Dev. Corr
 sub      (Intercept)  0.000
          X            7.634    NaN
 Residual             99.961
Number of obs: 10000, groups:  sub, 50
Fixed Effects:
(Intercept)            X
     99.859        2.479
convergence code 0; 2 optimizer warnings; 0 lme4 warnings
```

Using **FSL**, we split the analysis in two stages. First we fit the time series to each subject (using OLS as we generate white noise) and save the estamated parameter with its variance in *.nifti* files which we write to a temporary folder. Note, we had to create a matrix of $2 \times 2 \times 2$ voxels, all containing the same values as FSL is not able to run its functions on just one voxel (I think it has something to do with the indices). Then we run *feat* (option *flame1*) on the 4D cope and 4D varcope maps with a given mask. Note that we first generate some auxilirary files (such as the design matrix), then run *feat* (not executed here) and finally read back the results in.

```
# For this, we need to first analyze each subject individually, save COPE and VARCOPE
# and then proceed.
# We call this object secLevel
secLevel <- Y %>%
  group_by(sub) %>%
  do(.,
     # For each subject, fit linear model with an intercept and X as predictors
     broom::tidy(
       lm(Y ~ 1 + X, data = .))) %>%
  # Filter on predictor
  filter(term == 'X') %>%
  # Now select the estimate and standard error
  dplyr::select(sub, estimate, std.error) %>%
  # Create variance
  mutate(varCope = std.error^2)

# Create 4D images (all voxels in first 3 dimensions are the same), otherwise FSL crashes!
# Then convert the estimates and variance to nifti images
COPE4D <- nifti(img=array(rep(as.numeric(secLevel$estimate), each = 8),
                          dim=c(2,2,2,nsub)),
                dim=c(2,2,2,nsub), datatype = 16)
VARCOPE4D <- nifti(img=array(rep(as.numeric(secLevel$varCope), each = 8),
                             dim=c(2,2,2,nsub)),
                   dim=c(2,2,2,nsub), datatype = 16)

# Write them to DataWrite
writeNIfTI(COPE4D, filename = paste(DataWrite,'/COPE',sep=''), gzipped=FALSE)
writeNIfTI(VARCOPE4D, filename = paste(DataWrite,'/VARCOPE',sep=''), gzipped=FALSE)

#########################
## ETC ETC SEE var_2lvl.R
#########################

# Example of run in FSL
command <- paste(fslpath, 'flameo --cope=COPE --vc=VARCOPE --mask=mask --ld=FSL_stats --dm=design.mat
Sys.setenv(FSLOUTPUTTYPE="NIFTI")
system(command)

#########################
## ETC ETC SEE var_2lvl.R
#########################
```

Note, we simulate all actual data in the var_2lvl.R script.

2.3 True Values

Before we go to the results, we first calculate the true values. We will do this for the two-stage model formulation as the variance estimation of the random effects model is complicated (it involves taking the inverse of the full observation matrix which is too large). The two-stage model for a simplified (one predictor) GLM is defined as:

$$Y_{it} = \beta_0 + \beta_1 X + \varepsilon_{it}, \quad i = 1, \dots, N \quad \text{and} \quad t = 1, \dots, T. \tag{1}$$

In the second stage, we get:

$$Y_G = \beta_1^* X_G + \varepsilon^*, \tag{2}$$

where Y_G is the vector of estimated first level parameters ($\hat{\beta}_1$) and X_G equals a column of 1's with length N. In this case, $\varepsilon^* \sim N(0, \sigma_b^2 + \text{Var}(\widehat{\beta_1}))$. Denote σ_G^2 as $\text{Var}(\varepsilon^*)$ and note that is a mixed error component containing both variability of the estimation at the first level and a between-subject variability component σ_b^2.

Furthermore, we have:

$$\text{Var}(\widehat{\beta}_1) = \frac{\widehat{\sigma}_e^2}{\sum_{t=1}^{T}(X_t - \overline{X})^2} \tag{3}$$

Finaly, we find the variance of the estimated group level parameters ($\text{Var}(\beta_1^*)$) combining the equations from above:

$$\text{Var}(\beta_1^*) = \frac{\sigma_b^2 + \frac{\sigma_e^2}{\sum_{t=1}^{T}(X_t - \overline{X})^2}}{\sum_{i=1}^{N}(X_{Gi} - \overline{X_G})^2} \tag{4}$$

$$= \frac{\sigma_b^2 + \frac{\sigma_e^2}{\sum_{t=1}^{T}(X_t - \overline{X})^2}}{N} \tag{5}$$

In **R**, this is:

```r
X_G <- matrix(1, nrow = nsub, ncol = 1)
trueVarBetaG <- c((sigma_b2^2 + (sigma_eps^2 / sum((X_s - mean(X_s))^2))) / nsub)
trueVarBetaG
```

```
[1] 5.087648 5.587648 7.087648
```

```r
#(sigma_b2^2 + sigma_eps^2 ) / nsub
#diag(sigma_b2^2, nsub)
```

3 Results

Here, we read in the simulation results.

```
allDat <- data.frame() %>% as.tibble()
# For loop over the batches
for(i in 1:IDs){
  allDat <- bind_rows(allDat,
  readRDS(file = paste(RawDat, 'Results_bsub_',sigma_b2[1],'/VAR2LVL_',i,'.rda', sep = '')) %>%
    mutate(True_SD_bsub = sigma_b2[1]),
  readRDS(file = paste(RawDat, 'Results_bsub_',sigma_b2[2],'/VAR2LVL_',i,'.rda', sep = '')) %>%
    mutate(True_SD_bsub = sigma_b2[2]),
  readRDS(file = paste(RawDat, 'Results_bsub_',sigma_b2[3],'/VAR2LVL_',i,'.rda', sep = '')) %>%
    mutate(True_SD_bsub = sigma_b2[3])
  )
}
```

We can now check:

- the average of the β_1 estimates, should be `3`
- the empirical variance of the β_1 estimates (variance over Monte-Carlo simulations)
- the average of the estimated variance of β_1 (average over simulations of $Var(\beta_1)$). This should approach `5.0876479, 5.5876479, 7.0876479`, depending on the true value of σ_b^2.
- the average estimated between-subject variability. In FSL this is the file *mean_random_effects_var1*, with **R** this is the estimated term from *lmer*. It should approach `0, 5, 10`.
- the empirical coverage of the 95% CI around the true value of β_1.

```
allDat %>% group_by(type, True_SD_bsub) %>%
  summarise(Avg_beta = mean(estimate),
            Obs_var_beta = var(estimate),
            Avg_var_beta = mean(variance),
            Avg_sd_bsub = mean(SD_bsub),
            EC = mean(EC)) %>%
  mutate(TrueVar_beta = trueVarBetaG) %>%
  #Re-arrange
  ungroup() %>%
  dplyr::select(type, True_SD_bsub, Avg_sd_bsub, Avg_beta, TrueVar_beta, Obs_var_beta,
                Avg_var_beta,EC) %>%
  # Print to table
  kable(., caption = 'Results of Monte-Carlo simulation study', digits = 3)
```

Table 1: Results of Monte-Carlo simulation study

type	True_SD_bsub	Avg_sd_bsub	Avg_beta	TrueVar_beta	Obs_var_beta	Avg_var_beta	EC
FLAME	0	3.144	2.921	5.088	5.383	5.501	0.956
FLAME	5	4.789	2.944	5.588	5.858	5.830	0.956
FLAME	10	9.380	2.966	7.088	7.333	7.130	0.950
LMER	0	4.111	2.923	5.088	5.321	5.630	0.957
LMER	5	5.776	2.945	5.588	5.811	5.970	0.962
LMER	10	9.838	2.968	7.088	7.305	7.184	0.956

Not sure if the true variance is correct. . .

4 Session

```r
sessionInfo()
```

```
R version 3.4.3 (2017-11-30)
Platform: x86_64-apple-darwin15.6.0 (64-bit)
Running under: macOS High Sierra 10.13.5

Matrix products: default
BLAS: /Library/Frameworks/R.framework/Versions/3.4/Resources/lib/libRblas.0.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/3.4/Resources/lib/libRlapack.dylib

locale:
[1] nl_BE.UTF-8/nl_BE.UTF-8/nl_BE.UTF-8/C/nl_BE.UTF-8/nl_BE.UTF-8

attached base packages:
[1] tcltk     stats     graphics  grDevices utils     datasets  methods
[8] base

other attached packages:
 [1] bindrcpp_0.2       fMRIGI_0.1.0       NeuRRoStat_0.1.0
 [4] neuRosim_0.2-12    devtools_1.13.4    metafor_2.0-0
 [7] mvmeta_0.4.7       RColorBrewer_1.1-2 MASS_7.3-47
[10] lme4_1.1-14        Matrix_1.2-12      reshape2_1.4.3
[13] tidyr_0.7.2        knitr_1.17         tibble_1.4.2
[16] dplyr_0.7.4        ggplot2_2.2.1.9000 oro.nifti_0.9.1
[19] gridExtra_2.3      lattice_0.20-35    AnalyzeFMRI_1.1-16
[22] fastICA_1.2-1      R.matlab_3.6.1

loaded via a namespace (and not attached):
 [1] purrr_0.2.4       splines_3.4.3     colorspace_1.3-2
 [4] htmltools_0.3.6   yaml_2.1.14       rlang_0.1.6.9003
 [7] R.oo_1.21.0       pillar_1.1.0      nloptr_1.0.4
[10] withr_2.1.1.9000  glue_1.2.0        R.utils_2.6.0
[13] plyr_1.8.4        bindr_0.1         stringr_1.2.0
[16] munsell_0.4.3     gtable_0.2.0      RNifti_0.7.1
[19] R.methodsS3_1.7.1 evaluate_0.10.1   memoise_1.1.0
[22] highr_0.6         Rcpp_0.12.15      scales_0.5.0.9000
[25] backports_1.1.1   abind_1.4-5       digest_0.6.14
[28] stringi_1.1.6     grid_3.4.3        rprojroot_1.2
[31] tools_3.4.3       bitops_1.0-6      magrittr_1.5
[34] lazyeval_0.2.1    pkgconfig_2.0.1   assertthat_0.2.0
[37] minqa_1.2.4       rmarkdown_1.8     R6_2.2.2
[40] nlme_3.1-131      compiler_3.4.3
```