Analyzing time series data using the 'permutes' package

Cesko Voeten

21 September 2017

This vignette models EEG data, an example of *time series data*, using generalized additive mixed modeling (Wood, 2006) and permutation testing. Since both techniques rely fundamentally on the linear model of regression, we will first *very* briefly introduce the relevant mathematical foundations. The reader familiar with basic statistics can safely skip this section.

The linear model is summarized by the following equation:

$$Y = X\beta + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma^2 I)$$

where Y is a transposed vector (or a matrix, in the case of a multivariate response) known as the *outcome* (or alternatively, the *response*), X is the *design matrix*, β is a vector of *regression coefficients*, and ϵ is the unknown error, which is assumed to be normally distributed (in other words, i.i.d. Gaussian). Leaving out the unpredictable error gives us the modified regression equation below, which *estimates* the outcome by multiplying the predictors with estimated regression coefficients:

$$\hat{Y} = X\hat{\beta}$$

By construction, the residuals of this model sum to zero, but squaring them provides the *sum of squared residuals*:

$$r^2 = (\hat{Y} - X\hat{\beta})^2$$

The objective of linear regression is to find the estimates for $\hat{\beta}$ that minimize r^2 – in other words, to find the model that explains the most of the total variance. This is not difficult, because from defining the objective function as a squared quantity and assuming normal errors, it follows that the range of r^2 is *convex*, i.e. that there is no maximum and a single minimum. Finding the values for $\hat{\beta}$ that minimize the residuals is then simply a matter of differentiating r^2 with respect to $\hat{\beta}$ and solving for 0:

$$\begin{aligned}
r^2 &= (\hat{Y} - X\hat{\beta})^2 \\
&= (\hat{Y} - X\hat{\beta})^\mathsf{T}(\hat{Y} - X\hat{\beta}) \\
&= (\hat{Y}^\mathsf{T} - X^\mathsf{T}\hat{\beta}^\mathsf{T})(\hat{Y} - X\hat{\beta}) \\
&= \hat{Y}^\mathsf{T}\hat{Y} - \hat{Y}^\mathsf{T}X\hat{\beta} - X^\mathsf{T}\hat{\beta}^\mathsf{T}\hat{Y} + X^\mathsf{T}X\hat{\beta}^2 \\
\frac{\partial r^2}{\partial \hat{\beta}} &= 0 - \hat{Y}^\mathsf{T}X - X^\mathsf{T}\hat{Y} + 2X^\mathsf{T}X\hat{\beta} \\
0 &= 0 - 2X^\mathsf{T}\hat{Y} + 2X^\mathsf{T}X\hat{\beta} \\
-2X^\mathsf{T}X\hat{\beta} &= -2X^\mathsf{T}\hat{Y} \\
\hat{\beta} &= (X^\mathsf{T}X)^{-1}X^\mathsf{T}\hat{Y}
\end{aligned}$$

The covariance matrix is given by:

$$\hat{\sigma}(X^\mathsf{T}X)^{-1}, \quad \hat{\sigma} = s = \frac{r^2}{df}$$

The degrees of freedom for calculating the mean squared error are defined as $df = n - p$, where n is the number of observations and p is the number of predictors in the model. Subtracting p from n compensates for inaccuracy in the estimation of β by $\hat{\beta}$, which will have inflated the residuals $(Y - X\hat{\beta})^2$ away from their true values $(Y - X\beta)^2$ (remember that the residuals are squared, so this bias accumulates rather than canceling out). Accounting for this bias by artificially inflating the denominator results in s being an unbiased estimator of the 'true' mean squared error. Having the covariance matrix makes it possible to derive t-values by dividing the $\hat{\beta}$ by their standard errors, which are the square roots of the diagonal of the covariance matrix:

$$t_i = \frac{\hat{\beta}_i}{\sqrt{Cov_{i,i}}}$$

Analysis of variance squares these values and reports them as F instead, in the standard case of Type III ('marginal') SS. (For the 'sequential' types of SS, ANOVA works by performing the appropriate model comparison, in which case the difference in r^2 between the models is compared to the F distribution with the appropriate degrees of freedom.)

The model so far only contains *fixed effects*. The extension with *random effects* is:

$$Y = X\beta + Zb + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma^2 I)$$

where Z is the design matrix for the random effects and b is again an unknown vector of regression coefficients. The values of these coefficients are estimated using (restricted) maximum likelihood, which is computationally intractable if we don't assume some additional structure, namely:

$$Y = X\beta + Z\Lambda_\theta u + \epsilon, \quad u \sim \mathcal{N}(0, \sigma_1^2 I), \quad \epsilon \sim \mathcal{N}(0, \sigma_2^2 I)$$

where u and ϵ (and $\sigma_1^2 I$ and $\sigma_2^2 I$) are independent of each other. In this formulation b is decomposed into $\Lambda_\theta u$. Λ_θ is the relative covariance matrix for the random effects; it denotes those parts of the (co)variances that should be attributed to the random effects as opposed to the fixed effects or ϵ, relative to the overall model $\hat{\sigma}$. This matrix is block-diagonal, and is hence sparse; the non-zero entries are filled with θ-values, which have to be estimated. Which of the entries are non-zero depends on the parametrization; if we allow all entries to vary freely, the model will have to estimate $n \times (n+1)/2$ parameters (because $Cov_{x,y} \equiv Cov_{y,x}$) per random effect. `mgcv`, the package for generalized additive modeling that we will use, assumes that the off-diagonal entries (the correlations) are zero, and that each block in Λ_θ can be represented as a multiple of the identity matrix, therefore reducing the number of parameters to be estimated to one per random effect.

Note that only the θ parameters in Λ_θ have to be estimated via (restricted) maximum likelihood. Maximum likelihood is the iterative process of finding numerically the values for, in our case, θ, that maximize the likelihood of the data. For computational reasons, the packages available for calculating mixed-effects models do not maximize the likelihood ℓ, but rather minimize the deviance, which is just $-2log(\ell)$. It is apparently (Bates & DebRoy, 2004) not difficult to integrate out the unknown u during this optimization process. After convergence, this will yield the best linear unbiased estimators for β and θ from which it is only then necessary to calculate coefficients for u.

This is about as much detail as is needed to understand the `mgcv` analysis provided at the end, with one important addition. Because the values for θ and hence u are estimated so as to maximize the likelihood of the data, it is customary to add the squared sum of the estimates $\tilde{u}$ as a penalty term so that extreme values estimated for u artificially inflate the deviance. This promotes shrinkage of these coefficients towards zero, and is the backbone of `mgcv`'s approach to smooth estimation. This will be further detailed in the paragraph dedicated to the GAMM analysis.

Why time series data are special

The dataset we will be working with is called `MMN`, which is an excerpt from a passive-oddball task conducted by Jager (in prep.). A proper method section with more details should eventually be found in that paper, but a brief and incomplete summary follows here in order to make it easier to work with the data. Jager (in prep.) presented Dutch learners of English with a stream of isolated vowels from English or from Dutch created by means of formant synthesis. Each of her six conditions (summarized in the below table; the package supplies data from two of these conditions) each having used four vowels: three of the same phonetic category (e.g. three realizations of the DRESS vowel with slightly different formant frequencies) and one of a different phonetic category (e.g. the TRAP vowel). The first set of vowels, termed 'standards', were presented frequently; the fourth vowel, the 'deviant', infrequently interrupted the stream of standards. Participants were instructed not to pay attention to this stream of sounds, and watched a silent movie to keep them occupied. EEG recordings of 30 electrodes (T7 and T8 were excluded) were recorded while the participants were exposed to the vowel stimuli. At the presentation of one of the deviant vowels, we expect to observe a negative deflection in the EEG signal about 200 milliseconds after stimulus onset, originating from frontal and central electrode sites. This effect is called the 'mismatch negativity'. A second effect called the 'P300' may also occur, but is ignored here. The vowel pairs tested in Jager (in prep.) are summarized in the below table; the data supplied with `permutes` are a subset consisting of S13 vs. S94 (DRESS as a standard vs. as a deviant) and S16 vs. S95 (ZAT as a standard vs. as a deviant).

Code	Corresponding standard	Corresponding deviant
S11 or S91	ZET	DRESS
S12 or S92	DRESS	ZET
S13 or S93	DRESS	TRAP
S14 or S94	TRAP	DRESS
S15 or S95	ZET	ZAT
S16 or S96	ZAT	ZET

The first few rows of the data look as follows:

```
library(permutes)
```

```
## Loading required package: plyr
## Loading required package: lmPerm
## Loading required package: ggplot2
## Loading required package: viridis
## Loading required package: viridisLite
```

```
head(MMN)
```

```
##         Fp1     AF3      F7      F3     FC1     FC5      C3     CP1     CP5
## 769 -1.8819  0.2637 -0.6339 -0.5369  0.3605  0.6546 -0.1360 -0.3824 -0.2467
## 770 -1.5811  0.2429 -0.5817 -0.5476  0.3723  0.4233 -0.2799 -0.3673 -0.3039
## 771 -1.2132  0.1748 -0.5282 -0.5543  0.3645  0.2103 -0.4002 -0.3030 -0.3402
## 772 -0.8486  0.0609 -0.4839 -0.5335  0.3388  0.0170 -0.4797 -0.2047 -0.3537
## 773 -0.5375 -0.0973 -0.4516 -0.4804  0.2994 -0.1490 -0.5150 -0.0923 -0.3529
## 774 -0.2892 -0.2683 -0.4246 -0.4078  0.2580 -0.2589 -0.5064  0.0189 -0.3474
##          P7      P3      Pz     PO3      O1      Oz      O2     PO4
## 769 -0.3253 -0.6873 -0.8139 -0.6393 -0.9595 -1.4620 -1.2554 -1.6028
## 770 -0.3180 -0.5955 -0.8088 -0.6527 -0.9664 -1.4524 -1.1569 -1.4998
## 771 -0.2925 -0.4985 -0.7544 -0.6393 -0.9325 -1.4179 -1.0198 -1.3673
## 772 -0.2459 -0.4092 -0.6773 -0.6126 -0.8756 -1.3698 -0.8718 -1.2345
```

```
## 773 -0.1823 -0.3422 -0.6095 -0.5931 -0.8252 -1.3267 -0.7427 -1.1249
## 774 -0.1128 -0.3093 -0.5731 -0.6006 -0.8122 -1.3060 -0.6530 -1.0477
##          P4      P8     CP6     CP2     C4    FC6    FC2     F4     F8
## 769 -0.9905 -0.3362 -0.0847 -0.6391 0.4028 2.3192 0.6892 1.4870 0.6433
## 770 -0.8125 -0.1618  0.1654 -0.4452 0.6595 2.6499 0.8777 1.3410 0.6399
## 771 -0.6238  0.0271  0.3529 -0.2325 0.8683 2.7869 1.0239 1.1184 0.6470
## 772 -0.4587  0.1942  0.4282 -0.0461 0.9820 2.7140 1.0996 0.8637 0.6663
## 773 -0.3384  0.3129  0.3924  0.0821 0.9904 2.4648 1.0976 0.6197 0.6986
## 774 -0.2608  0.3788  0.3083  0.1503 0.9329 2.1185 1.0392 0.4281 0.7312
##         AF4     Fp2     Fz     Cz      time ppn session cond dev
## 769 -0.4083 -1.0715 0.7902 0.4035  1.171875   1       0  S13   1
## 770 -0.2220 -0.5532 0.8840 0.4983  3.125000   1       0  S13   1
## 771 -0.0992  0.1172 0.9133 0.5953  5.078125   1       0  S13   1
## 772 -0.0532  0.8056 0.8751 0.6761  7.031250   1       0  S13   1
## 773 -0.0520  1.3555 0.7887 0.7273  8.984375   1       0  S13   1
## 774 -0.0574  1.6360 0.6891 0.7515 10.937500   1       0  S13   1
```

```
nrow(MMN) #how many observations?
```

```
## [1] 44814
```

```
length(unique(MMN$time)) #how many timepoints?
```

```
## [1] 231
```

The first 30 columns are the 30 sampled EEG electrodes. The `time` column denotes the moment from stimulus onset, in milliseconds, of each measurement type. `ppn` is the participant, and `session` is the session of the experiment minus one, to make it a proper dummy indicating whether the datapoint is from the first (0) or the second session (1). `cond` indicates the condition code, which can be decoded from the table. Finally, `dev` is a dummy indicating whether the stimulus was a standard (0) or a deviant (1), explained in the next paragraph. Note that, contrary to the results and recommendations in Brysbaert (2007), Jager (in prep.) averaged over all her items belonging to the same condition in this experiment.

Time series data such as these are special because the data consist of multiple, *strongly correlated*, measurements of the same signal. This generates two problems. The first is a research problem: which portion of this time series do we want to analyze? The `permutes` package was designed to help researchers answer precisely this question. The second problem is of a statistical nature: how do we handle the strong autocorrelation present throughout this sampled window? Note that in the case of EEG data, these same two problems are additionally encountered in the *spatial* domain: what combination of electrodes ('region of interest') do we want to analyze, and how do we deal with the spatial correlation present in data measured from neighboring sites? This issue is dealt with in the `permutes` package by running separate analyses for each site, as is shown below.

Determining the window and ROI

The first problem equates to what is known in the EEG literature as determining the *window*. The normal way to do this is to run a MANOVA (with the 30 electrodes as the dependent variables) on every individual timepoint, and take as the window the point where a large sequence of significant p-values occurs. If we plot time horizontally and the electrode site vertically, we can use this approach to select both a time window and an ROI at the same time.

It should be noted that this approach suffers from an extreme multiple-comparisons problem: we will be running 30×231 ANOVAs! When compared to an asymptotic null distribution, the p-values will be spuriously significant in, expectedly, 347 random combinations. The permutation testing approach to time series data helps with this problem in two ways. Firstly, and most importantly, the results from a permutation analysis are *never* interpreted as actual findings; rather, they only serve as a guideline to empirically determine the

analysis window and ROI. Secondly, the null distribution is not taken as the asymptotic F distribution, but is rather inferred from the data itself by *permutation testing*: the null distribution is obtained by randomly permuting the entries of Y and assessing how badly this perturbs the model fit; if a permutation incurs a large deterioration of the fit, then the portions of the design matrix associated with that permutation apparently contributed rather significantly to obtaining the proper fit. It should be noted that this is certainly not the *best* way to derive an empirical null distribution: permutation testing could be considered nothing more than a poor man's approach to bootstrapping. Bootstrapping approaches, however, are not feasible when dealing with large datasets such as the `MMN` data. While bootstrapping 2310 ANOVAs on, on average, 58 rows of data each will be too slow for the average user, permutation testing will be much faster, by virtue of not having to generate a large number of random draws from a posterior distribution.

The below code runs a permutation test series on the `MMN` data and plots the results as a heatmap. Note that permutation testing is not deterministic (the permutations are chosen randomly), so your results may be slightly different from mine.

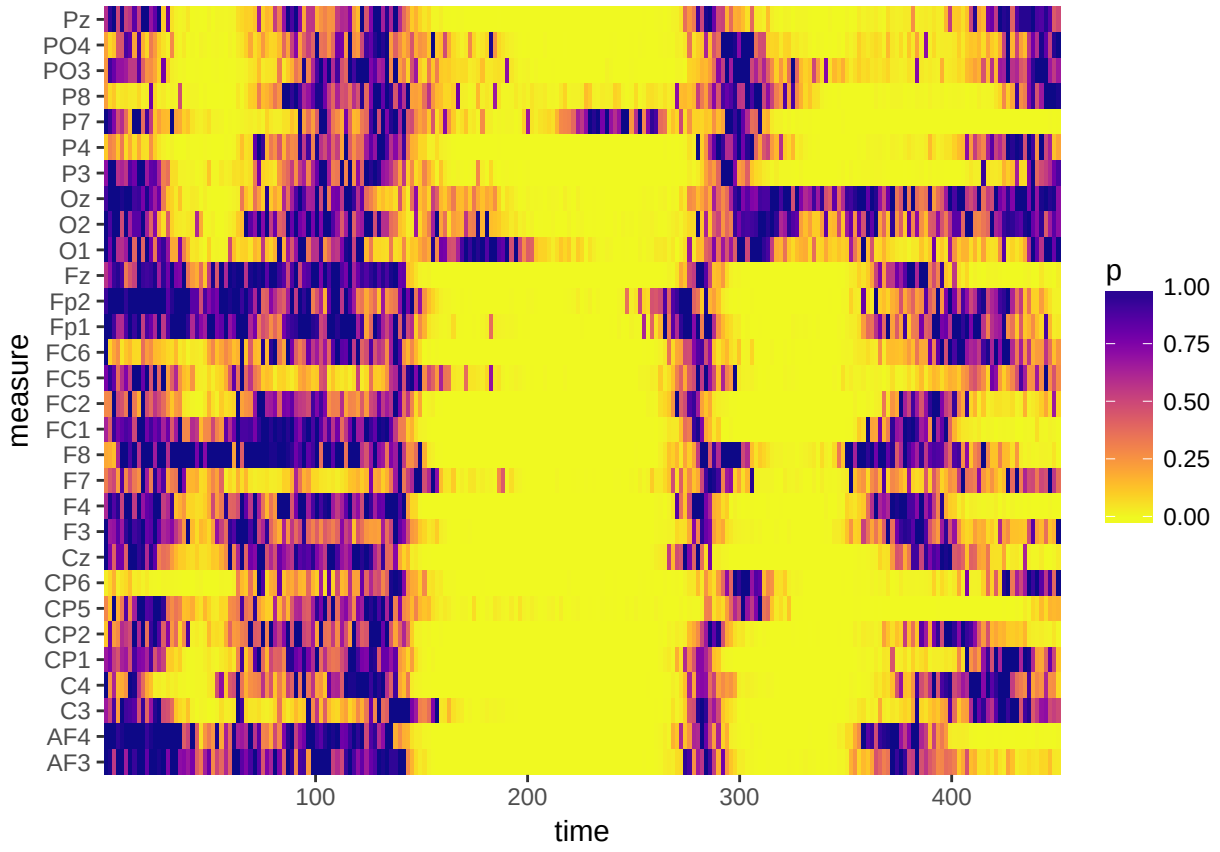
```
perms <- permu.test(cbind(Fp1,AF3,F7,F3,FC1,FC5,C3,CP1,CP5,P7,P3,Pz,PO3,O1,Oz,O2,PO4,
    P4,P8,CP6,CP2,C4,FC6,FC2,F4,F8,AF4,Fp2,Fz,Cz) ~ dev | time,data=MMN)
## (output not shown)
```

This takes a few seconds to run. We can speed it up by parallelizing the testing of the individual timepoints:

```
library(doParallel)
cl <- makeCluster(2) #or more
registerDoParallel(cl)
perms <- permu.test(cbind(Fp1,AF3,F7,F3,FC1,FC5,C3,CP1,CP5,P7,P3,Pz,PO3,O1,Oz,O2,PO4,
    P4,P8,CP6,CP2,C4,FC6,FC2,F4,F8,AF4,Fp2,Fz,Cz) ~ dev | time,data=MMN,parallel=TRUE)
```

We can then plot the results:

```
plot(perms)
```

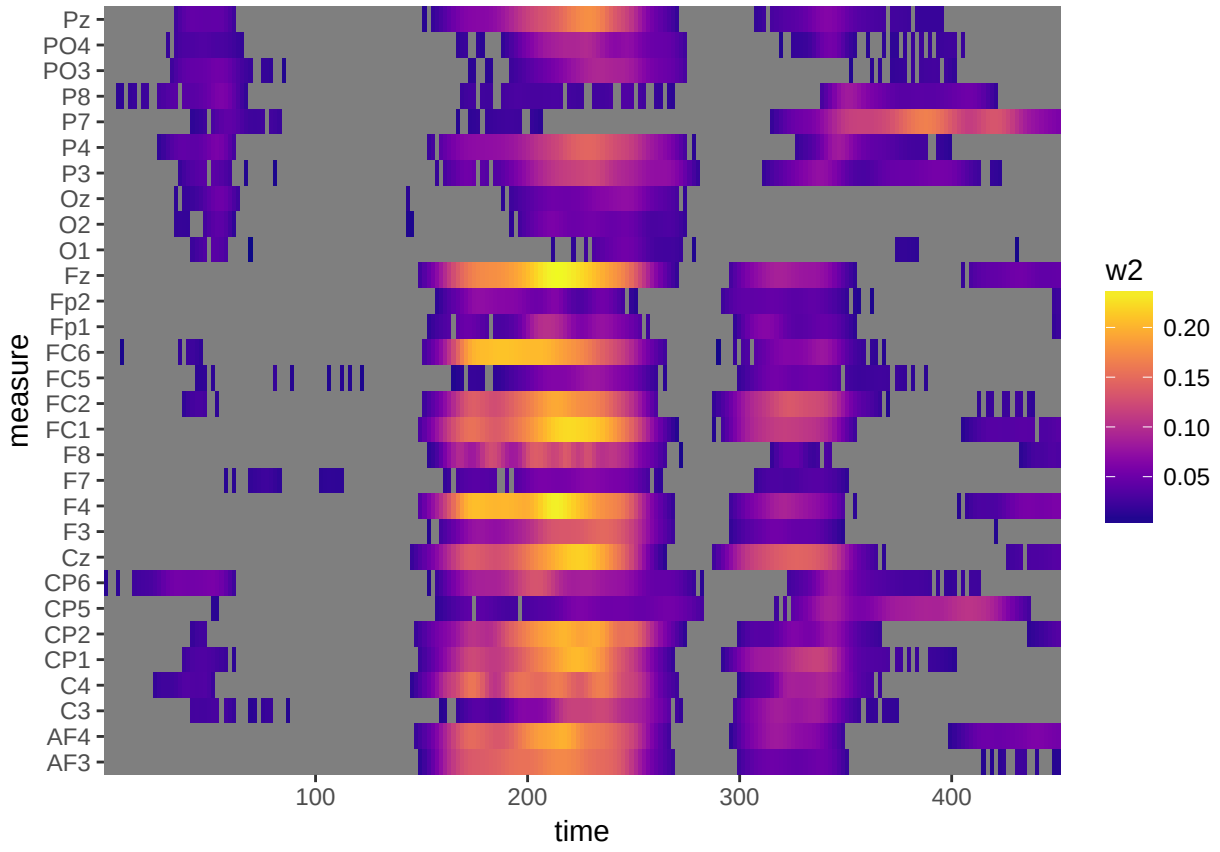


The plot suggests two windows: one window located around 200 ms post-stimulus-onset, and a second window located around 330 ms post-stimulus-onset. The first window corresponds to the expected mismatch-negativity effect; the second window is likely a P300, which we will ignore here. A region of interest is not easily identified: it appears that all sites are giving significant results. This is expected given the spatial correlation inherent to EEG data, but at the same time we should realize that the significant p-values do not really tell us all that much: the *effect sizes* might vary by ROI in a meaningful way, but a binary criterion $p < .05$ eclipses this variation. Once we know that a measurement is significant, its *effect size* (ω^2) would be a more useful measure. We can ask for a heatmap of the effect sizes by passing `plot='w2'` (as opposed to the default `plot='p'`) to our `plot` command. However, we first gray out the calculated effect sizes for those cases whose p-values did not reach the $< .05$ cutoff point, because effect sizes for non-significant effects are dubious.

```
head(perms) #take a look at the data structure
```

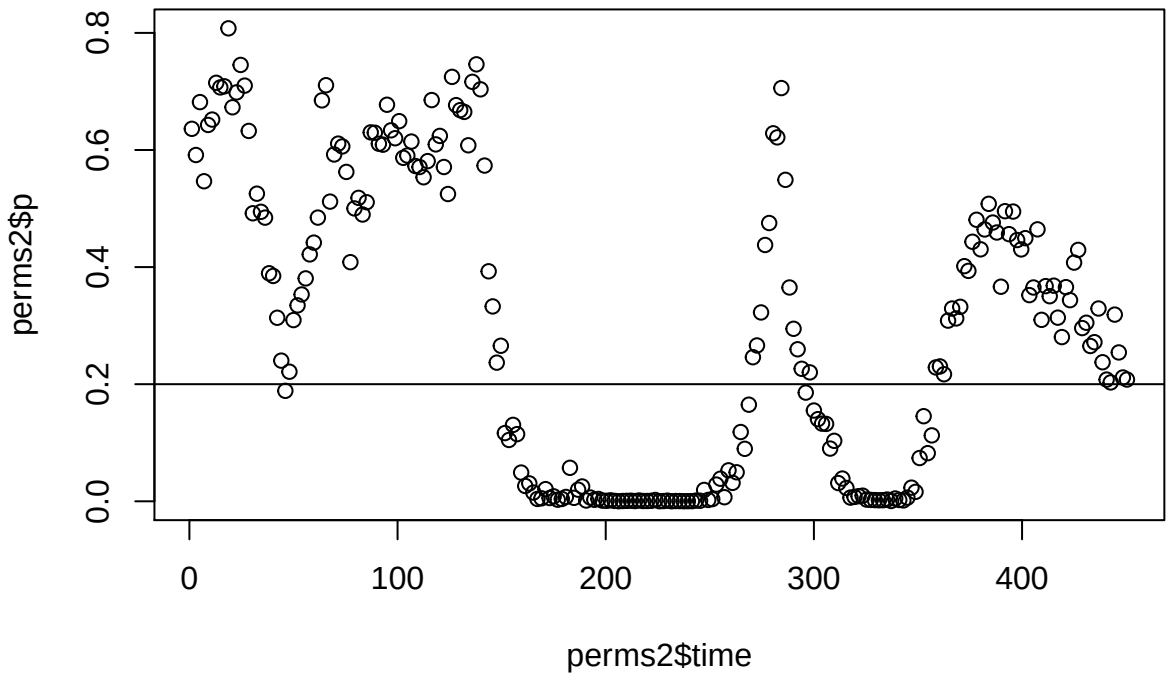
```
##    measure     time factor         p w2
## 1      Fp1 1.171875    dev 1.0000000  0
## 3      AF3 1.171875    dev 1.0000000  0
## 5       F7 1.171875    dev 0.3194444  0
## 7       F3 1.171875    dev 1.0000000  0
## 9      FC1 1.171875    dev 0.6333333  0
## 11     FC5 1.171875    dev 0.7647059  0
```

```
perms[perms$p >= .05,'w2'] <- NA
plot(perms,plot='w2')
```



From this plot, the effects look to be largest at the frontal and central regions. This is again good news for Jager (in prep.), as it is in line with prior literature on the MMN component. We can now exploit the fact that `permutes` objects are also normal dataframes to precisely determine our temporal window. We remove the channels outside of the frontal/central ROI and then aggregate the p-values that remain. We can then define arbitrary cutoff points for including/excluding a part of the larger time window.

```
perms2 <- perms[perms$measure %in% .(Fp1,AF3,F7,F3,FC1,FC5,C3,CP1,CP5,CP6,CP2,C4,FC6,FC2,
    F4,F8,AF4,Fp2,Fz),]
perms2$measure <- 'Aggregate'
perms2 <- aggregate(p ~ measure + time + factor,perms2,mean)
plot(perms2$time,perms2$p)
abline(h=.2)
```



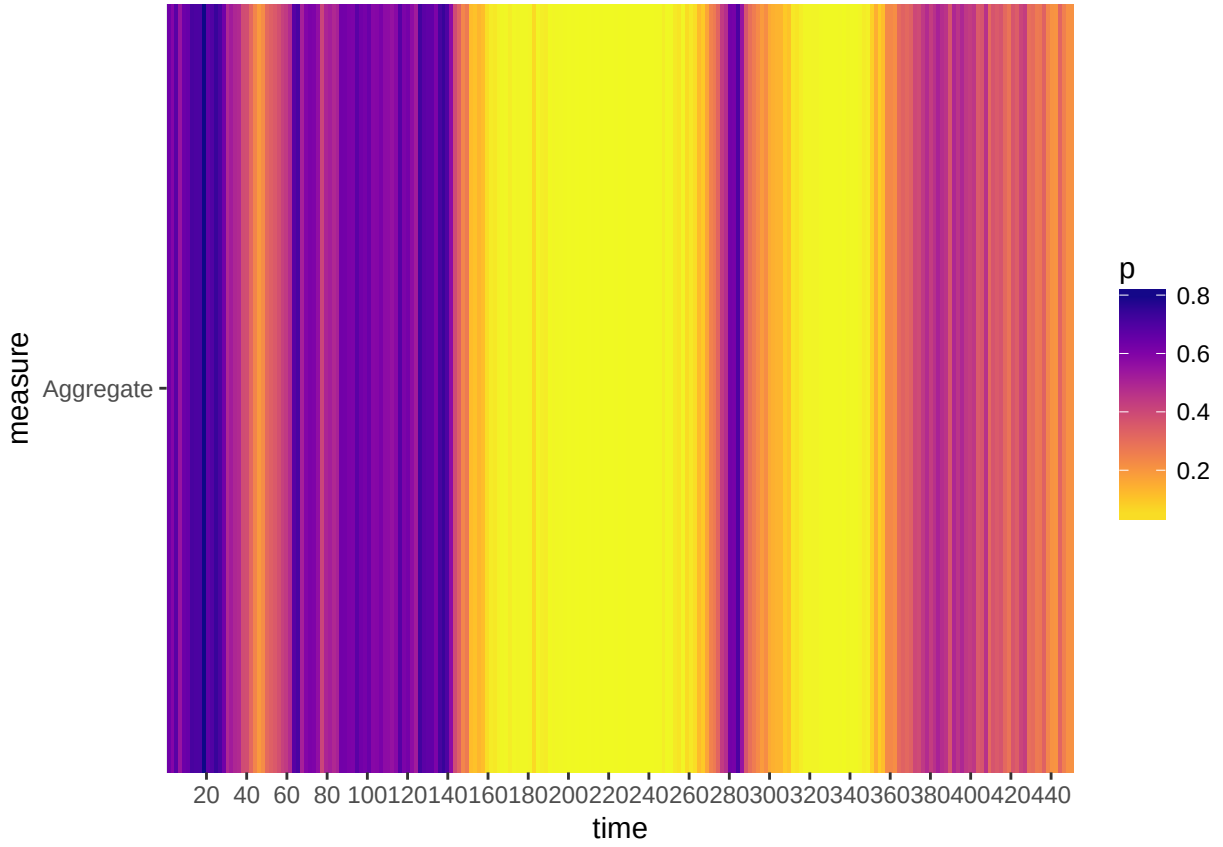
Keep in mind that our aggregating of p-values is statistically completely nonsensical, so we cannot use $p < .05$ as a criterion for determining our window here. Based on the plot, the author would arbitrarily suggest picking a value of .2. The included `abline(h=.2)` shows that this looks quite reasonable. Let's see what time window corresponds to this value:

```
unique(perms2[perms2$p < .2,'time'])
```

```
##  [1]  46.09375 151.56250 153.51562 155.46875 157.42188 159.37500 161.32812
##  [8] 163.28125 165.23438 167.18750 169.14062 171.09375 173.04688 175.00000
## [15] 176.95312 178.90625 180.85938 182.81250 184.76562 186.71875 188.67188
## [22] 190.62500 192.57812 194.53125 196.48438 198.43750 200.39062 202.34375
## [29] 204.29688 206.25000 208.20312 210.15625 212.10938 214.06250 216.01562
## [36] 217.96875 219.92188 221.87500 223.82812 225.78125 227.73438 229.68750
## [43] 231.64062 233.59375 235.54688 237.50000 239.45312 241.40625 243.35938
## [50] 245.31250 247.26562 249.21875 251.17188 253.12500 255.07812 257.03125
## [57] 258.98438 260.93750 262.89062 264.84375 266.79688 268.75000 296.09375
## [64] 300.00000 301.95312 303.90625 305.85938 307.81250 309.76562 311.71875
## [71] 313.67188 315.62500 317.57812 319.53125 321.48438 323.43750 325.39062
## [78] 327.34375 329.29688 331.25000 333.20312 335.15625 337.10938 339.06250
## [85] 341.01562 342.96875 344.92188 346.87500 348.82812 350.78125 352.73438
## [92] 354.68750 356.64062
```

Based on this output combined with (common?) sense, the author would recommend a window ranging from 151 to 269 milliseconds. If the analyst trained in the NHST framework is afraid of the large degree of personal judgement in this approach, it is possible to coerce our `perms2` object – which became a `data.frame` after the `aggregate` call – back into a `permutes`-class object, so that the more familiar (and colorful) plot provided by `permutes` can be used to make the same decision:

```
class(perms2) <- c('permutes','data.frame')
plot(perms2,breaks=20*0:(450/20)) #x-axis labeling
```



Having determined a window and an ROI, we can now proceed to the actual modeling step.

Modeling EEG data using generalized additive mixed modeling

As stated in the beginning, generalized additive mixed models are capable of dealing with the temporal correlation and the spatial correlation that are both still present in our data. We first need to select the subset of our data corresponding to our chosen window. We then need to center our timepoint predictor (important for our temporal smooth later on) and to melt the dataset and assign coordinates to the electrode positions (important for our spatial smooth later on). Finally, we extract only our chosen ROI.

```
part <- MMN[MMN$time > 151 & MMN$time < 269,]
part$time <- part$time - mean(part$time)

library(reshape2)
part <- melt(part,id.vars=.(time,ppn,session,cond,dev),
    variable.name='electrode',value.name='amplitude')
part$electrode <- as.character(part$electrode)
part <- ddply(part,~electrode,function (data) {
    me <- unique(data$electrode)
    data$x <- list(Fz=0,Cz=0,Pz=0,Oz=0,FC1=-1,CP1=-1,Fp1=-2,AF3=-2,F3=-2,C3=-2,P3=-2,
        PO3=-2,O1=-2,FC5=-3,CP5=-3,F7=-4,T7=-4,P7=-4,FC2=1,CP2=1,Fp2=2,AF4=2,F4=2,
        C4=2,P4=2,PO4=2,O2=2,FC6=3,CP6=3,F8=4,T8=4,P8=4)[[me]]
    data$y <- list(T7=0,C3=0,Cz=0,C4=0,T8=0,CP5=1,CP1=1,CP2=1,CP6=1,P7=2,P3=2,Pz=2,P4=2,
```

```
        P8=2,PO3=3,PO4=4,O1=5,Oz=5,O2=5,FC5=-1,FC1=-1,FC2=-1,FC6=-1,F7=-2,F3=-2,Fz=-2,
        F4=-2,F8=-2,AF3=-3,AF4=-3,Fp1=-4,Fp2=-4)[[me]]
    data
})
part <- part[part$electrode %in% .(Fp1,AF3,F7,F3,FC1,FC5,C3,CP1,CP5,CP6,CP2,C4,FC6,FC2,F4,
    F8,AF4,Fp2,Fz),]
part <- part[complete.cases(part),] #remove cases with missing electrodes
```

Generalized additive mixed models work by assuming that a user-specified set of predictors can be modeled by a user-specified smooth function. As a simple (and rather pointless) example, consider a one-dimensional linear smooth of a hypothetical independent variable x whose values $x_{1\ldots10}$ are the numbers one to ten and whose true effect on the response is to increase it by $\mathcal{N}(3x, 1)$. The generalized additive mixed model would fit this effect in two places. First of all, it will perform a smooth-specific transformation to the data points $x_{1\ldots10}$ and store the result as a predictor in X. In our example, the smooth is linear (and is hence not really a smooth at all), so this will just result in a column containing the numbers one to ten. This models the fixed part of the smooth; in our example, this effect will be fitted as $\hat{\beta} = 3$. It will also add the same column to Z, modeling the wiggly part of the smooth; for our example, its variance should be estimated as 1.

Of course, a linear smooth is exactly the same as the ordinary combination of a fixed effect and random slope over a dummy grouping factor, by definition. In our analysis of Jager's (in prep.) EEG data, we will use three different smooths that are much more useful. The first smooth type is known in `mgcv` as `re` (for 'random effect'). This smooth is a little different from the general case described in the previous paragraph, because it only operates on Z, and does not actually add its parameters to X at all. It is discussed first because this is also the only type of smooth (to the author's knowledge) that, like our linear-smooth example, does not transform the data. These two properties mean that this smooth is exactly equivalent to a simple random effect. This smooth is used four times in our analysis: once to denote a random intercept by participants (in which case the effect's block in Z will consist of a column of ones for each participant), and three times to denote a random slope by participants (in which case it will be represented by an actual part of the `MMN` dataset).

We will also use two 'real' smooths. The first is a Duchon spline, which is useful for modeling spatial data. This smooth is of dimension 2 (because it will smooth over the $\langle x, y\rangle$ coordinates of the electrodes on the cap) and is fitted using nineteen knots (i.e. nineteen columns in X and in Z). This number was not chosen arbitrarily: it is the number of unique combinations of $\langle x, y\rangle$ pairs present in the data. As we will see below, this number is probably a fair bit too high, but the only consequence of that is that the model will take much longer to fit than if a more optimal number had been chosen.

The second smooth is a thin plate regression spline, which will smooth over the 'time' predictor. The number of knots is arbitrarily but reasonably (given the already closely-specified temporal interval) chosen as 5. Note that what we actually ask the model to calculate is the *tensor product* of the Duchon spline and the thin plate regression spline: this is conceptually (but not mathemathically) similar to an interaction. Mathematically, interactions between two smooths are impossible, hence the name 'generalized *additive* model', but the tensor product combines the two smooths together into a single large smooth matrix. Because we use `te()` rather than `ti()`, this matrix contains both the 'main effects' of our Duchon spline and thin plate regression spline, as well as their combination (i.e. how the spatial distribution of the EEG activity changes over time).

In addition to this tensor product of two smooths, we include the same tensor product combined with a random effect by participants (modeling how the smooths differ across individuals). Our use of `t2` and `full=TRUE` make this tensor product coincide with what for simplex random smooths is called a 'factor smooth', and we pass `m=1` to penalize the first-order derivatives of the smooths (when `m` is a single number, it is applied to all smooths inside a tensor product). This is a stricter penalty than, for instance, enforcing smoothness of the second-order derivatives, which would allow the smooth itself to vary freely in slope, as long as the *changes* in the slopes are relatively small; for non-random smooth terms, this would be sensible to obtain effective smoothing, but for a random smooth it is better to enforce a stronger penalty. This causes the estimated slopes to shrink towards zero, which penalizes overfitting in a similar way to how linear mixed-effects regression penalizes larger values of u. For the Duchon spline and the thin plate regression

spline, we pass the same number of knots as we did before (which, we will see below, is gravely suboptimal in terms of computational efficiency), and for the random effect we pass a nonsensical number of 10, which is ignored anyway because a random effect *has* no knots.

It is important to keep in mind that *none of these smooth terms are of actual interest*. They *only* serve to compensate for the random differences between participants and for the autocorrelated space and time that are inherent to EEG data. The real predictors of interest are the parametric terms `dev`, `session`, and their interaction. To ease interpretability, we run a separate analysis for both of the two conditions included in the supplied dataset. We then expect significant MMN effects in only some of the conditions and not in others, for theoretical reasons beyond the scope of this vignette. Note that this is a cluster-class workload: an ordinary desktop computer will not be able to fit models of this complexity to a dataset this large (~7.7 million rows). Since we will assume that this analysis will be run on a computer cluster, we will liberally make use of the parallelization capabilities in `mgcv` and of the fact that fitting two separate models is an obviously parallelizable task. The code we use is presented below:

```
part$both <- part$dev * part$session #construct an interaction term to be used as by=...
                                     #argument
clusterExport(cl,'part')
pairs <- list(c('S13','S94'),c('S16','S95'))
fun <- function (C) {
    library(parallel)
    cl <- makeCluster(8)
    library(mgcv)
    data <- part[part$cond %in% C,]

    form <- amplitude ~ dev*session + s(ppn,bs='re') + s(ppn,by=dev,bs='re') +
        s(ppn,by=session,bs='re') + s(ppn,by=both,bs='re') +
        te(x,y,time,bs=c('ds','tp'),d=c(2,1),k=c(19,5)) +
        t2(x,y,time,ppn,bs=c('ds','tp','re'),d=c(2,1,1),k=c(19,5,10),m=1,full=TRUE)
    model <- bam(form,data=data,method='REML',cluster=cl)

    # Figure out the autocorrelation in the residuals. We cannot use acf(resid(m)),
    # because that will not take into account event boundaries (the whole dataset will be
    # treated as a single huge EEG signal)
    AR.start <- data$time == min(data$time)
    res <- resid(model)
    bins <- findInterval(res,which(AR.start))
    res.binned <- split(res,bins)
    rhos <- sapply(res.binned,function (x) acf(x,plot=FALSE)$acf[2])

    m.corrected <- bam(form,data=data,method='REML',cluster=cl,
        AR.start=AR.start,rho=mean(rhos))
    list(model.ac=model,model.corrected=m.corrected)
}
models <- parLapply(cl,pairs,fun)
```

A few bits of this code require some explanation. First, we make a list of data combinations that are of interest to us; it is by this list that we subset the data into our two comparisons. This is not the most straightforward way to segment the data into our two bins of interest, but this is how Jager (in prep.) formatted her data. Then, for each of our two comparisons (which will be run in parallel), we create an eight-node subcluster for use by `mgcv`, subset our data, and then formulate and fit our model. We pass `method='REML'` to our call to `bam` instead of using the default, `fREML`, because fREML can be unreliable at times.

It is very important to mention that our use of the REML criterion (i.e. restricted maximum likelihood), rather than the conventional maximum likelihood, means that it is illegitimate to perform model comparisons

between models differing in X. Without going into too much detail, the REML criterion can be defined as:

$$\ell_{REML}(\theta, \sigma|Y) = \int_{-\infty}^{\infty} \ell(\beta, \theta, \sigma) d\beta$$

where ℓ is the likelihood of the data given the current parameter estimates for the model (but in practice, ℓ is always replaced by the deviance). The innovation over the traditional likelihood is that the β parameter is integrated out just like u was earlier (see the final paragraph of the introduction). This compensates for bias inherent to regular maximum likelihood estimation, which is the same problem we saw in the first paragraph when we formed $\hat{\sigma}$ by dividing the r^2 by $n - p$ rather than by n. Maximum likelihood finds the values for θ that maximize the likelihood of the data given β, θ and σ, but it fails to take into account that we have *estimated* β by $\hat{\beta}$, which will have introduced error (that again accumulates when squared and takes away variance from the residual error σ). This is something that should again be compensated just like it is when calculating $\hat{\sigma}$ (or rather s) as an unbiased estimator of σ. This is done by maximizing not the likelihood, but the *restricted* likelihood. As shown above, this REML criterion is calculated by integrating out the fixed effects. In other words, the model maximizes the likelihood of observing the data *for all values of* β, rather than only the estimated $\hat{\beta}$. Crucially, this means that if two models do not have the same structure for X, and hence have different βs, integrating with respect to β amounts to adding a different constant to the likelihood function. *It is then no longer valid to perform model comparisons by comparing the change in deviance to a* χ^2 *distribution*, since the change in deviance between the two models will also include the change in this integrated constant. If we wanted to do model comparisons (we see no reason to, although a scrutinous linguist might consider the `dev:session` term for exclusion), we would have to refit the models with `method='ML'`.

There is one part of this code that still needs explanation. We actually fit the model not once, but twice. This is because EEG data is often temporally autocorrelated in the residuals, even after including a temporal smooth (e.g. De Cat, Klepousniotou, & Baayen (2015)). The first time we fit the model, we only use the resulting fit to determine how large this autocorrelation is. We extract the lag-1 autocorrelation present in the model's residuals, and then fit the model again, this time informing `bam` that it should take an autocorrelation of that size into account. Note that the true autocorrelation is likely to be much more complex than an AR(1) process, and might be better modeled using an ARMA process instead; however, currently `bam` is only able to fit lag-1 autocorrelated models. It is possible to fit more complex correlation structures using `gamm`, but this routine is much slower, less numerically stable, and, at least when run on these data, often produces singular fits during the optimization process, which is a fatal error in `nlme`. (This is because Pinheiro & Bates (2000) parameterize Λ_θ not in terms of relative covariance, but rather in terms of its reciprocal, viz. relative precision. A singular fit then corresponds to infinite parameter values, from which numerical calculation cannot continue.)

The results of the two (corrected) fits – which are not distributed along with the package due to the large size of the `models` object (the `.RData` file is 2GiB in size) – are shown below:

```
for (i in 1:2) {
    cat('\n---',pairs[[i]],'---\n')
    print(summary(models[[i]]$model.corrected))
}
```

```
##
## --- S13 S94 ---
##
## Family: gaussian
## Link function: identity
##
## Formula:
## amplitude ~ dev * session + s(ppn, bs = "re") + s(ppn, by = dev,
##     bs = "re") + s(ppn, by = session, bs = "re") + s(ppn, by = both,
```

```
##     bs = "re") + te(x, y, time, bs = c("ds", "tp"), d = c(2,
##     1), k = c(19, 5)) + t2(x, y, time, ppn, bs = c("ds", "tp",
##     "re"), d = c(2, 1, 1), k = c(19, 5, 10), m = 1, full = TRUE)
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.4372     0.1645   2.657  0.00788 **
## dev          -0.9273     0.2289  -4.051  5.1e-05 ***
## session      -0.3770     0.1677  -2.248  0.02456 *
## dev:session  -0.2180     0.3499  -0.623  0.53335
## ---
## Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1    1
##
## Approximate significance of smooth terms:
##                       edf  Ref.df         F p-value
## s(ppn)              25.58   26.00     86.94  <2e-16 ***
## s(ppn):dev          25.92   26.00  58198.01  <2e-16 ***
## s(ppn):session      19.03   20.00  47701.99  <2e-16 ***
## s(ppn):both         19.75   20.00 126259.46  <2e-16 ***
## te(x,y,time)        72.60   77.75     24.41  <2e-16 ***
## t2(x,y,time,ppn) 1594.82 2538.00  13258.98  <2e-16 ***
## ---
## Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1    1
##
## R-sq.(adj) =   0.78   Deviance explained = 78.3%
## -REML = -56934  Scale est. = 0.13832   n = 111325
##
## --- S16 S95 ---
##
## Family: gaussian
## Link function: identity
##
## Formula:
## amplitude ~ dev * session + s(ppn, bs = "re") + s(ppn, by = dev,
##     bs = "re") + s(ppn, by = session, bs = "re") + s(ppn, by = both,
##     bs = "re") + te(x, y, time, bs = c("ds", "tp"), d = c(2,
##     1), k = c(19, 5)) + t2(x, y, time, ppn, bs = c("ds", "tp",
##     "re"), d = c(2, 1, 1), k = c(19, 5, 10), m = 1, full = TRUE)
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.6868     0.1679   4.091 4.30e-05 ***
## dev          -0.9621     0.2207  -4.359 1.31e-05 ***
## session      -0.5602     0.1749  -3.203  0.00136 **
## dev:session  -0.0888     0.2474  -0.359  0.71963
## ---
## Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1    1
##
## Approximate significance of smooth terms:
##                       edf  Ref.df        F  p-value
## s(ppn)              11.16   26.00    0.894 1.61e-10 ***
## s(ppn):dev          25.54   26.00 1271.480  < 2e-16 ***
## s(ppn):session      18.61   20.00  987.790 3.94e-11 ***
## s(ppn):both         18.96   20.00 1448.366 1.57e-11 ***
```

```
## te(x,y,time)       68.41   75.26   11.431  < 2e-16 ***
## t2(x,y,time,ppn) 1344.11 2545.00  389.414 1.65e-13 ***
## ---
## Signif. codes:  0  ***  0.001  **  0.01  *  0.05  .  0.1    1
##
## R-sq.(adj) =  0.641   Deviance explained = 64.5%
## -REML = -38299  Scale est. = 0.44324   n = 111203
```

Note that the results show that our settings for the `k` parameters for the smooths were not entirely optimal. A value of `k` that is too low will result in undersmoothing, and hence in underfitting of the data. This can be detected by looking for smooth parameters in the summary whose `edf` (estimated degrees of freedom) are very close to the reference degrees of freedom (i.e. the maximum permitted by our choice of `k`). On the other hand, a value for `k` that is too large is statistically entirely unproblematic. One might expect that a value that is too large will lead to oversmoothing and hence overfitting, but because the splines are penalized this problem resolves itself: the model artificially disfavors smooths that use more knots than justified by the data. Thus, if a `k` value is set too large, the analyst only pays in terms of computational complexity, i.e. the amount of time needed to fit the model. This is because by telling `gam` to use a large number of knots, we instruct it to search a large space of possible solutions to the model, whereas the 'true' solution is apparently located in a much smaller interior space. For the first spatiotemporal smooth, the difference between the `edf` and `Ref.df` suggest that we have number of knots we have allowed the model to use is adequate; we could perhaps reduce the spatial smooth by one or two knots, but in general it looks like the model has been estimated relatively efficiently. For the second, random, smooth, however, the number of knots is nearly twice as much as the models actually needed to use, so significant computational efficiency can be gained by reducing the number of knots here. Exploring this possibility is beyond the scope of this vignette, however.

This concludes our analysis of Jager's (in prep.) EEG data. On a final note, the summary statistics reported here should probably be corrected for multiple comparisons, but my supervisor does not seem to think so, so I'll leave it at that. If the analyst wants to correct for multiple comparisons, the Sidak correction is a good option:

$$\alpha_{Sidak} = 1 - (1 - \alpha)^{\frac{1}{n}}$$

If we assume $\alpha = .05$ and divide it by the corrected α_{Sidak}, we conclude that our p-values should be multiplied by ~1.97. Note that the actual implementation should be something like:

```
corrected.summary <- function (summary) {
    correct <- function (table) {
        corr <- .05 / (1 - (1-.05)^(1/2))
        table[,4] <- pmin(table[,4]*corr,1)
        table
    }
    summary$p.table <- correct(summary$p.table)
    summary$s.table <- correct(summary$s.table)
    return(summary)
}
```

to clamp the p-values to the $[0, 1]$ range. (p-values exceeding 1 are an error in `R`, and will prevent the summary from being printed.)

References

Bates, D. M., & DebRoy, S. (2004). Linear mixed models and penalized least squares. *Journal of Multivariate Analysis*, *91*(1), 1–17.

Brysbaert, M. (2007). The language-as-fixed-effect-fallacy: Some simple SPSS solutions to a complex problem. *London: Royal Holloway, University of London*.

De Cat, C., Klepousniotou, E., & Baayen, R. H. (2015). Representational deficit or processing effect? An electrophysiological study of noun-noun compound processing by very advanced L2 speakers of English. *Frontiers in Psychology*, *6*.

Jager, L. (in prep.).

Pinheiro, J., & Bates, D. M. (2000). *Mixed-effects models in S and S-PLUS*. Springer Science & Business Media.

Wood, S. (2006). *Generalized additive models: An introduction with R*. Chapman; Hall/CRC.