fastFMM Vignette: Fast Functional Mixed Models using Fast Univariate Inference

Gabriel Loewinger, Erjia Cui

2023-06-09

Contents

Introduction

`fastFMM` is a fast toolkit for fitting functional mixed models using the fast univariate inference (FUI) method proposed in Cui et al. (2022). Assume the multi-level/longitudinal functional data are of the type $Y_{ij}(s)$ on a grid $\{s_1, ..., s_L\}$ of the compact functional domain $\mathcal{S}$. Denote by $i = 1, 2, ..., I$ the index of subject, and $j = 1, 2, ..., J_i$ the index of longitudinal visit at time t_{ij}. In addition to $Y_{ij}(s)$, for each visit of each subject, we also observe $\boldsymbol{X}_{ij} = [X_{ij1}, X_{ij2}, ..., X_{ijp}]^T \in \mathbb{R}^p$ as the fixed and $\boldsymbol{Z}_{ij} = [Z_{ij1}, Z_{ij2}, ..., Z_{ijq}]^T \in \mathbb{R}^q$ as the random effects variables. Denote by $g(\cdot)$ a pre-specified link function and $EF(\cdot)$ the exponential family distribution. A longitudinal function-on-scalar regression model has the following form:

$$
\begin{aligned}
&Y_{ij}(s) \sim EF\{\mu_{ij}(s)\}, \\
&g\{\mu_{ij}(s)\} = \eta_{ij}(s) = \boldsymbol{X}_{ij}^T \boldsymbol{\beta}(s) + \boldsymbol{Z}_{ij}^T \boldsymbol{u}_i(s).
\end{aligned}
$$

The model above is also called Functional Mixed Model (FMM) in the Functional Data Analysis (FDA) literature. Many statistical methods have been proposed to fit this model, which in general fall into two categories: joint methods and marginal methods. FUI is a marginal method that is computationally fast and achieves similar estimation accuracy compared with the state-of-the-art joint method, such as the `refund::pffr()` function.

FUI consists of the following three steps:

1. At each location $s_l \in \mathcal{S}, l = 1, 2, ..., L$, fit a pointwise generalized linear mixed model (GLMM):

$$Y_{ij}(s_l) \sim EF\{\mu_{ij}(s_l)\},$$
$$g\{\mu_{ij}(s_l)\} = \eta_{ij}(s_l) = \boldsymbol{X}_{ij}^T \boldsymbol{\beta}(s_l) + \boldsymbol{Z}_{ij}^T \boldsymbol{u}_i(s_l).$$

Here $\boldsymbol{\beta}(s_l)$ is a $p \times 1$ dimensional vector of fixed effects and $\boldsymbol{u}_i(s_l)$ is a $q \times 1$ dimensional vector of random effects. Denote the estimates of fixed effects from L separate univariate GLMMs as $\tilde{\boldsymbol{\beta}}(s_1), \tilde{\boldsymbol{\beta}}(s_2), ..., \tilde{\boldsymbol{\beta}}(s_L)$.

2. Smooth the estimated fixed effects $\tilde{\boldsymbol{\beta}}(s_1), \tilde{\boldsymbol{\beta}}(s_2), ..., \tilde{\boldsymbol{\beta}}(s_L)$ along the functional domain. The smoothed estimators are denoted as $\{\hat{\boldsymbol{\beta}}(s), s \in \mathcal{S}\}$.

3. Obtain the pointwise and joint confidence bands for fixed effects parameters using analytic approaches for Gaussian data or a bootstrap of subjects (cluster bootstrap) for Gaussian and non-Gaussian data.

The key insight of the FUI approach is to decompose the complex correlation structure into longitudinal and functional directions. For more details on the analytic and bootstrap approach, please refer to the paper.

In this vignette, we provide a tutorial on how to use the `fui` function to fit FMM. The toolkit is currently implemented in R, but may incorporate C++ to speed up the code in the future development. Notice that the model function assumes that the data are stored in a `data.frame`, where the predictors are stored as separate columns and the outcome is stored as a matrix; see examples for the required format.

Installation

The development version of `fastFMM` can be downloaded from github by executing:

```
library(devtools)
install_github("gloewing/fastFMM")
```

Package Setup

Downloading the development version of the package requires one to install package dependencies. We have included the code below to install any `CRAN` packages not already downloaded on your computer.

```
# install packages (will only install them for the first time)
list.of.packages = c("lme4", "parallel", "cAIC4", "magrittr","dplyr",
                     "mgcv", "MASS", "lsei", "refund","stringr", "Matrix", "mvtnorm",
                     "arrangements", "progress", "ggplot2", "gridExtra", "here")
new.packages = list.of.packages[!(list.of.packages %in% installed.packages()[,"Package"])]
if(length(new.packages)){
  chooseCRANmirror(ind=75)
  install.packages(new.packages, dependencies = TRUE)
```

```
}else{
  # load packages if already installed
  library(lme4)
  library(parallel)
  library(cAIC4)
  library(magrittr)
  library(dplyr)
  library(mgcv)
  library(MASS)
  library(lsei)
  library(refund)
  library(stringr)
  library(Matrix)
  library(mvtnorm)
  library(arrangements)
  library(progress)
  library(ggplot2)
  library(gridExtra)
}
```

```
library(fastFMM) # load our package
```

Tutorial Guide

To illustrate the main functions in our `fastFMM` package, we start by analyzing synthetic data in which the functional domain is defined by time: each functional observation is a short time-series "trial' '.

Data Formating

Let's load the data and take a look at it's structure.

```
dat <- read.csv("time_series.csv") # read in data
head(dat[,1:6])
```

```
##   id trial treatment       Y.1        Y.2        Y.3
## 1  1     1         0 -2.779477 -2.2031073 -1.0292572
## 2  1     2         0 -1.581592 -1.5353845 -0.4135614
## 3  1     3         0 -1.378131 -1.2898723 -0.9604425
## 4  1     4         0 -1.896447 -1.1348146 -0.7991782
## 5  1     5         0 -1.301714 -0.5457362 -1.0763990
## 6  1     6         0 -2.111444 -1.6129569 -0.7286003
```

This dataset has $N = \sum_{i=1}^{I} n_i$ rows, where n_i is the number of repeated measures observations of subject/cluster i. The first 3 columns of the dataset include the subject ID, and the covariates `trial` and `treatment`. Each of the remaining columns are observations of our functional outcome named [`Y.1`, `Y.2`, ... `Y.L`] where L is the number of points we observe the function on.

When we specify a model below using the `fui()` function, it will take the columns starting with the string 'Y.' and assume they are ordered from left to right. For that reason, make sure no other variables begin with the same substring.

Alternatively, instead of relying on the column names, we can save the matrix $N \times L$ matrix associated with our functional outcomes as a "column'' of a dataframe.

```
Y_mat <- dat[,-seq(1,3)]
head(Y_mat[,1:5])
```

Now we save the `Y_mat` matrix as a "column'' in the dataframe:

```
dat <- data.frame(Y = Y_mat, dat[,seq(1,3)])
```

The `fui()` function accepts either format style.

Model Fitting and Function Arguments

Let's start with a simple model not too unlike the functional version of a paired-samples t-test: we include the binary `treatment` covariate with a random slope. Observe that the syntax for the model formula is based off of the `lmer()` function in the `lme4` package. The model is then:

$$Y_{ij}(s) = \beta_0(s) + X_{ij}\beta_1(s) + u_i(s) + \epsilon_{ij}(s)$$

We start with implementation of the model with analytic inference.

```
mod <- fui(Y ~ treatment + # main effect of cue
                 (1 | id),  # random intercept
               data = dat)
```

Unless specified otherwise, `fui()` assumes a Gaussian family and provides inference based on an analytic form. Below we fit a similar model but include a random subject-specific random slope for treatment.

- If we instead wished to use bootstrap confidence intervals, we simply set the argument `analytic=FALSE`. The number of bootstrap replicates can be set with the argument `boot`. This defaults to `boot=500`.
- Both analytic and bootstrap inference can be sped up substantially by parallelizing the univariate model-fitting with `parallel=TRUE`. The number of cores used for parallelizing can be specified in `num_cores`. Otherwise, `R` will set this automatically based on the number of cores available on your computer.

```
mod <- fui(Y ~ treatment + # main effect of cue
                 (treatment | id),  # random intercept
                 data = dat,
                 parallel = TRUE,
                 analytic = FALSE) # bootstrap
```

- The `family` argument is the standard GLM family from `R stats::glm()`. More specifically, our package allows for any family in the `glmer()` and `lmer()` functions offered by the `lme4` package.
- At times one might wish to analyze only a subset of the functional domain by, for example, downsampling the number of points of the functional outcome, or using only certain subintervals. This can be set by providing a vector of indices of which points of the functional outcome to use in the analysis using the `argvals` argument. For example, the following analyzes every 3rd point of the functional outcome.

```
Y_mat <- dat[,-seq(1,3)])
L <- ncol(Y_mat) # number of columns of functional outcome

mod <- fui(Y ~ treatment + # main effect of cue
                  (treatment | id),  # random intercept
                  data = dat,
                  argvals = seq(from = 1, to = L, by = 3) # every 3rd data point
                  )
```

- We may wish to compare model fit between multiple models. To speed this process up, we can turn off the full inference by setting `var = FALSE` and then use the `AIC` and `BIC` naturally outputted in the model objects.

```
Y_mat <- dat[,-seq(1,3)])
L <- ncol(Y_mat) # number of columns of functional outcome

# model 1: random slope model
mod1 <- fui(Y ~ treatment + # main effect of cue
                  (treatment | id),  # random intercept
                  data = dat,
                  var = FALSE)

# model 2: random intercept model
mod2 <- fui(Y ~ treatment + # main effect of cue
                  (1 | id),  # random intercept
                  data = dat,
                  var = FALSE)

# compare model fits
colMeans(mod1$aic)
colMeans(mod2$aic)
```

Since model 1 has a lower average `AIC`, one might wish to proceed with inference using that model.

- For any complex random effect structures (e.g., nested) or anytime more than one variable is specified on the righthand side of the `|` symbol in the random effects formula , the column name in the `data` data.frame corresponding to the subject-ID should be specified in the argument `subj_ID` to avoid any confusion. See the example below:

```
mod <- fui(Y ~ treatment + # main effect of cue
                  (treatment | id/trial),  # random intercept
                  data = dat,
                  subj_ID = "id")
```

Model Fit Plots

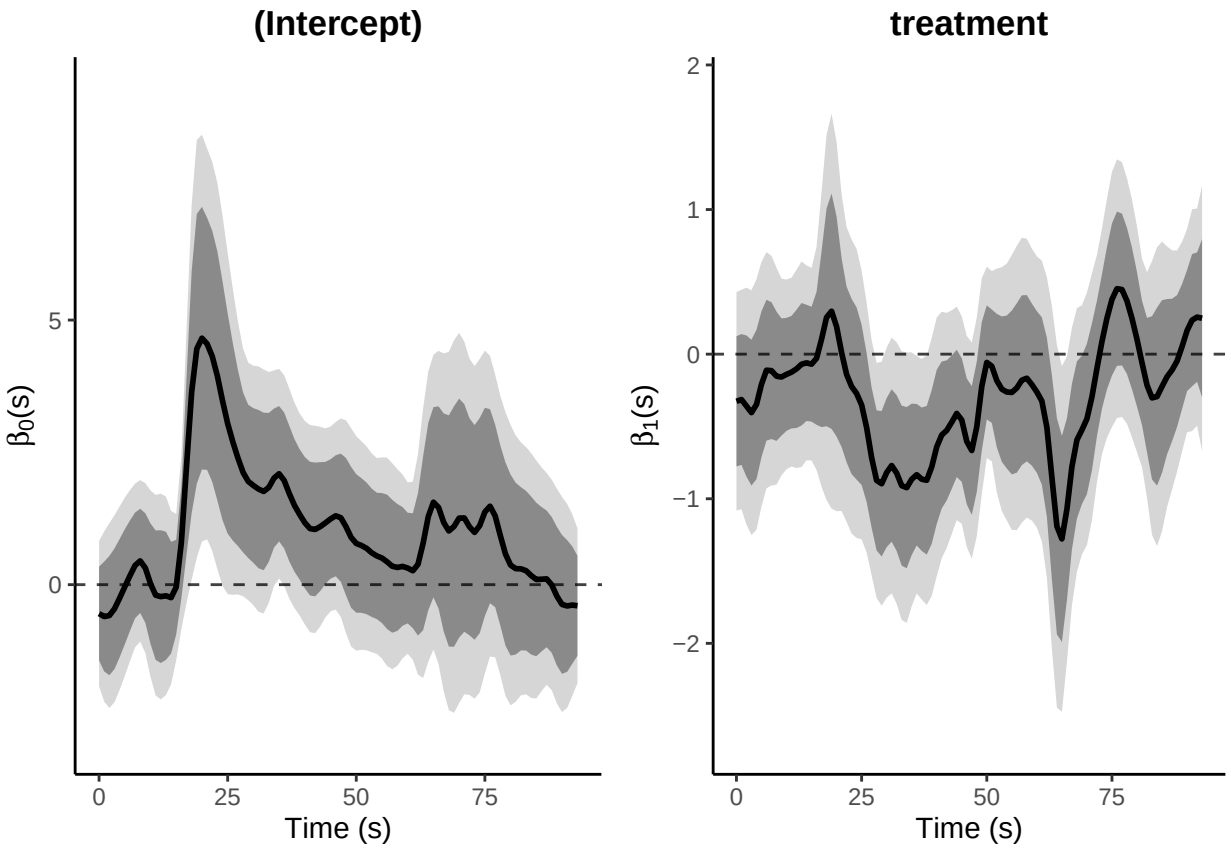
After fitting a model, one can quickly visualize estimates and 95% confidence intervals with out plot function.

```
mod <- fui(Y ~ treatment + # main effect of cue
                  (treatment | id),  # random intercept
                  data = dat)
```

```
## [1] "Step 1: Fit Massively Univariate Mixed Models"
## [1] "Step 2: Smoothing"
## [1] "Step 3: Inference (Analytic)"
## [1] "Step 3.1: Preparation"
## [1] "Step 3.1.1: Preparation B"
## [1] "Step 3.1.1: Method of Moments Covariance Estimator"
## [1] "Step 3.1.2: Smooth G"
## [1] "Step 3.2: First step"
## [1] "Step 3.3: Second step"

## Complete!
##  -Use plot_fui() function to plot estimates.
##  -For more information, run the command:  ?plot_fui
```

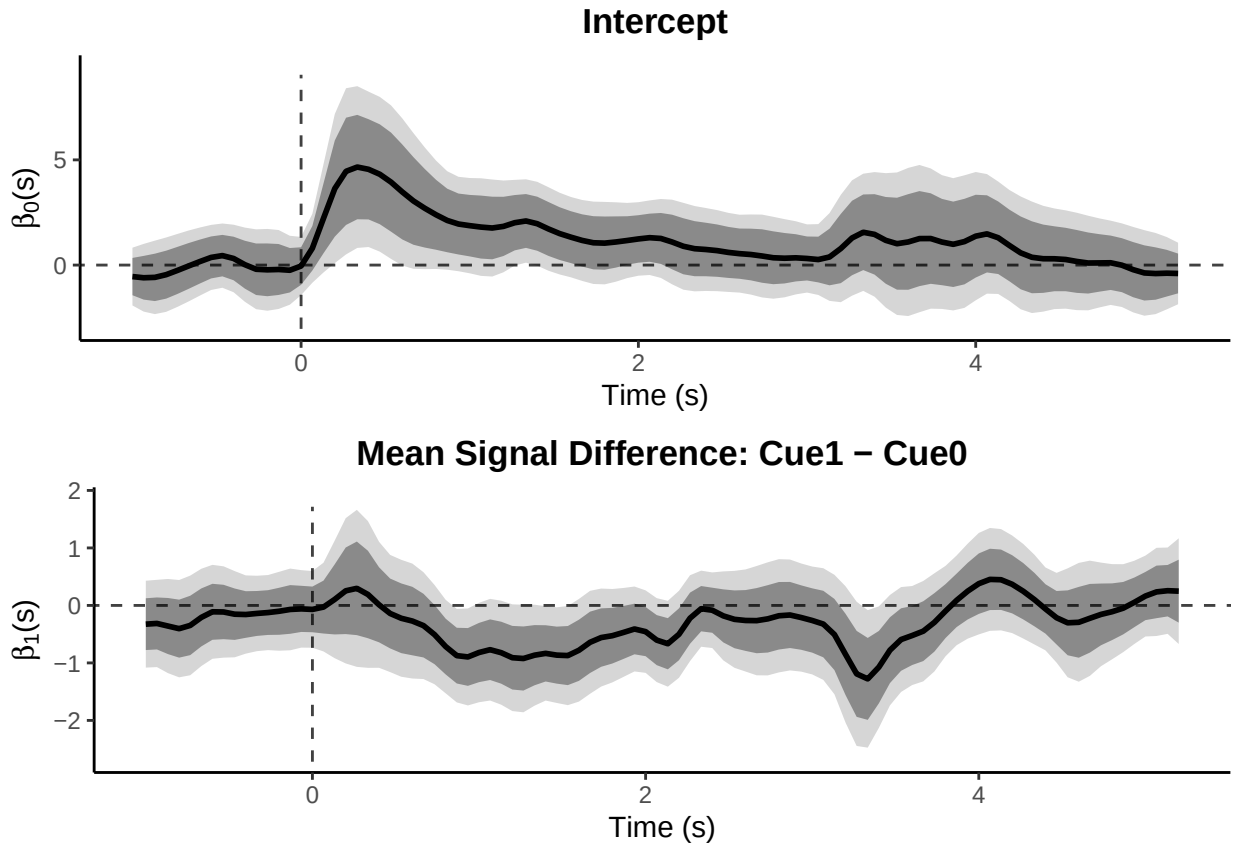
```
fui_plot <- plot_fui(mod)
```



In some cases, especially if the functional domain is time, one may wish to rescale the x-axis (using the `x_rescale` argument) according to the sampling rate and align the plot to a certain time-point (using the `align_x` argument). We also allow the user to adjust how many rows the plots are plotted on (through the `nrow` argument). Each coefficient's plot can be given a plot title through the vector argument `title_names` and the x-axis can be named through the `xlab` argument.

```
align_time <- 1 # cue onset is at 2 seconds
sampling_Hz <- 15 # sampling rate
# plot titles: interpretation of beta coefficients
```

```
plot_names <- c("Intercept", "Mean Signal Difference: Cue1 - Cue0")
fui_plot <- plot_fui(mod, # model fit object
                     x_rescale = sampling_Hz, # rescale x-axis to sampling rate
                     align_x = align_time, # align to cue onset
                     title_names = plot_names,
                     xlab = "Time (s)",
                     num_row = 2)
```



One can access the regression coefficient estimates and associated pointwise and joint 95% confidence intervals by looking at the object outputted from the `plot_fui()` function. This is stored in the object which is a list that has p elements (one for each fixed effect). Each element in that list (named according to the corresponding fixed effects covariate) is a data-frame that has L rows (L is number of points observed along the functional domain) and columns for coefficient estimates and 95% CIs.

```
head(fui_plot$treatment)
```

```
##   s   beta.hat CI.lower.pointwise CI.upper.pointwise CI.lower.joint
## 1 1 -0.3259615         -0.7762194          0.1242964      -1.081435
## 2 2 -0.3135889         -0.7659282          0.1387504      -1.072555
## 3 3 -0.3588443         -0.8469862          0.1292976      -1.177883
## 4 4 -0.4043604         -0.9087335          0.1000127      -1.250633
## 5 5 -0.3479826         -0.8644588          0.1684937      -1.214562
## 6 6 -0.2084777         -0.7129112          0.2959558      -1.054851
##   CI.upper.joint
## 1      0.4295125
```

```
## 2      0.4453774
## 3      0.4601941
## 4      0.4419118
## 5      0.5185971
## 6      0.6378958
```

Advanced Options

Additional `fui()` arguments

- Note that the package automatically outputs the status of the code at each step, but this can be turned off by setting `silent = FALSE`.
- The number of knots for smoothing can be adjusted by `nknots_min` argument which defaults to $L/2$.
- The manner for tuning the smoothing hyperparameter (see `mgcv` package for more information) can be adjusted through the argument `smooth_method`. Options are `GCV` and `GCV.Cp` and `REML`. Defaults to `GCV.Cp`.
- The type of spline is adjusted in the argument `splines`. Options include `tp`, `ts`, `cr`, `cs` `te`, `ds`, `bs`, `gp` and more (see `smooth.terms` page from the `mgcv` package for more information). Defaults to thin plate: `tp`.
- `design_mat` specifies whether the design matrix is returned. Defaults to `FALSE`.
- `residuals` specifies whether the residuals of the univariate GLMM fits are returned. Defaults to `FALSE`.
- `G_return` specifies whether the covariance matrices $G()$ are returned. See Cui et al. (2022) for more information. Defaults to `FALSE`.
- `caic` specifies whether conditional AIC model fit criteria are returned alongside `AIC` and `BIC`. Defaults to `FALSE`.
- `REs` specifies whether the BLUP random effect estimates from the univariate GLMM fits are returned. These estimates are directly taken from the output of `lmer()` and `glmer()`. Defaults to `FALSE`.
- `non_neg` specifies whether any non-negativity constraints are placed on variance terms in the Method of Moments estimator for $G()$. We recommend keeping this at its default `non_neg=0`.
- `MoM` specifies the type of Method of Moments estimator for $G()$. It is possible that for very large datasets and certain random effects structures that `MoM=1` may reduce the computational burden but the statistical performance of this setting is unknown. We therefore highly recommend keeping this at its default `MoM=2`.

Additional `plot_fui()` arguments

- `num_row` specifies the number of rows the plots will be displayed on. Defaults to $p/2$
- `align_x` aligns the plot to a certain point on the functional domain and sets this as "0.'' This is particularly useful if time is the functional domain. Defaults to 0.
- `x_rescale` rescales the x-axis of plots which is especially useful if time is the functional domain and one wishes to, for example, account for the sampling rate. Defaults to 1.
- `title_names` Allows one to change the titles of the plots. Takes in a vector of strings. Defaults to NULL which uses the variable names in the dataframe for titles.
- `y_val_lim` is a positive scalar that extends the y-axis by a factor for visual purposes. Defaults to 1.10. Typically does not require adjustment.

- `ylim` takes in a 2×1 vector that specifies limits of the y-axis in plots.
- `y_scal_orig` a positive scalar that determines how much to reduce the length of the y-axis on the bottom. Typically does not require adjustment. Defaults to 0.05.
- `xlab` takes in a string for the x-axis title (i.e., the functional domain). Defaults to "Time (s)''

Troubleshooting

- You are welcome to contact us about `fastFMM` at gloewinger@gmail.com with any questions or error reports.
- This GitHub account gloewing/fastFMM hosts the development version of fastFMM.
- The following warning messages can occur
  - "Warning messages: 1: In sqrt(Eigen$values) : NaNs produced'' – this can occur when using bootstrap confidence intervals because of the used functional PCA functions. It can be safely ignored.
  - 'Warning in checkConv(attr(opt, "derivs"), opt, ctrl = control checkConv: Model failed to converge with max|grad'' -- this is alme4' warning message: for data sets with many observations or for models with many parameters (e.g., random effects, fixed effects), convergence warnings are not uncommon. This warning message comes from point-wise mixed model fits. Therefore if these warnings are issued for many points along the functional domain, you may want to consider your fixed or random effect specification. If the warning is only issued for a few points on the functional domain, you may not need to alter your model specification.
  - 'Warning in checkConv(attr(opt, "derivs"), opt.par, ctrl = control.checkConv, : Model is nearly unidentifiable: very large eigenvalue: - Rescale variables? Model is nearly unidentifiable: large eigenvalue ratio'' -- This is alme4warning message --This warning can frequently be resolved by scaling continuous covariates, for example, with the commandscale()--For more information on thelme4' package, warning messages, model syntax and general mixed modeling information, see: Doug Bates. lme4: Mixed-effects modeling with R (2022).

References

Doug Bates. lme4: Mixed-effects modeling with R (2022).

Erjia Cui, Andrew Leroux, Ekaterina Smirnova, and Ciprian Crainiceanu. Fast Univariate Inference for Longitudinal Functional Models. Journal of Computational and Graphical Statistics (2022).