Modellazione multilevel con componenti principali funzionali

Daniele Zago, Simon Mazzarolo, Silvia Brosolo, Emanuele Donà

2021-04-28

```
library(rstanarm)
library(bayesplot)
library(tidybayes)
library(ggplot2)
library(magrittr)
library(lme4)
library(dplyr)
library(gtools)
library(tidyr)
```

Preprocessing

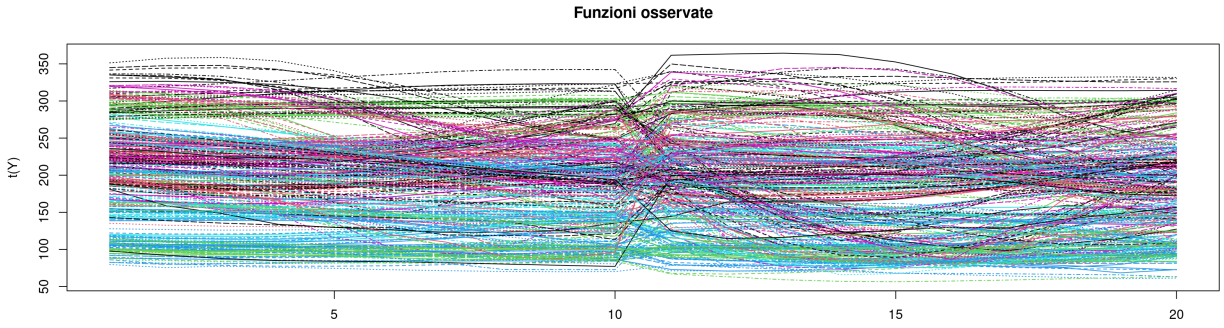
Costruiamo il dataset in formato "largo" per la stima delle fPCA:

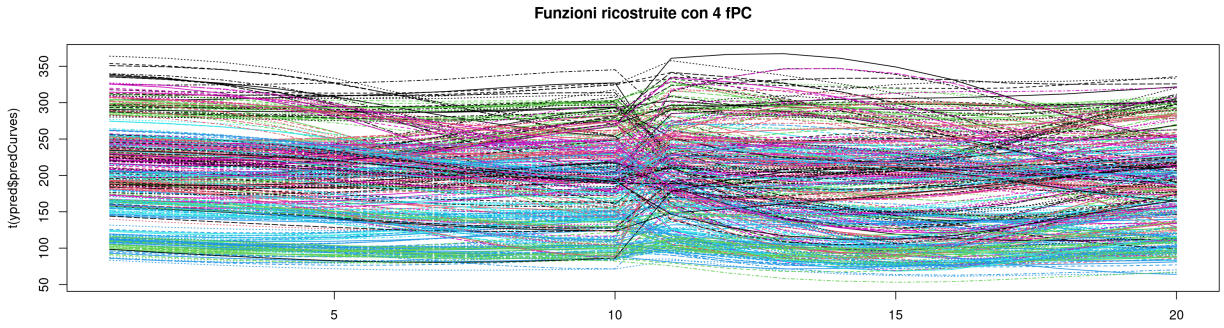
```
mandarino_wide = spread(mandarino[ , -6 ], time, "f0")
Y = as.matrix(mandarino_wide[ , 5:(NCOL(mandarino_wide)) ])     # Funzioni osservate

mandarino$peak = ifelse(mandarino$time >= 7 & mandarino$time <= 13, 1, 0)
```

Analisi con componenti principali funzionali (fPCA)

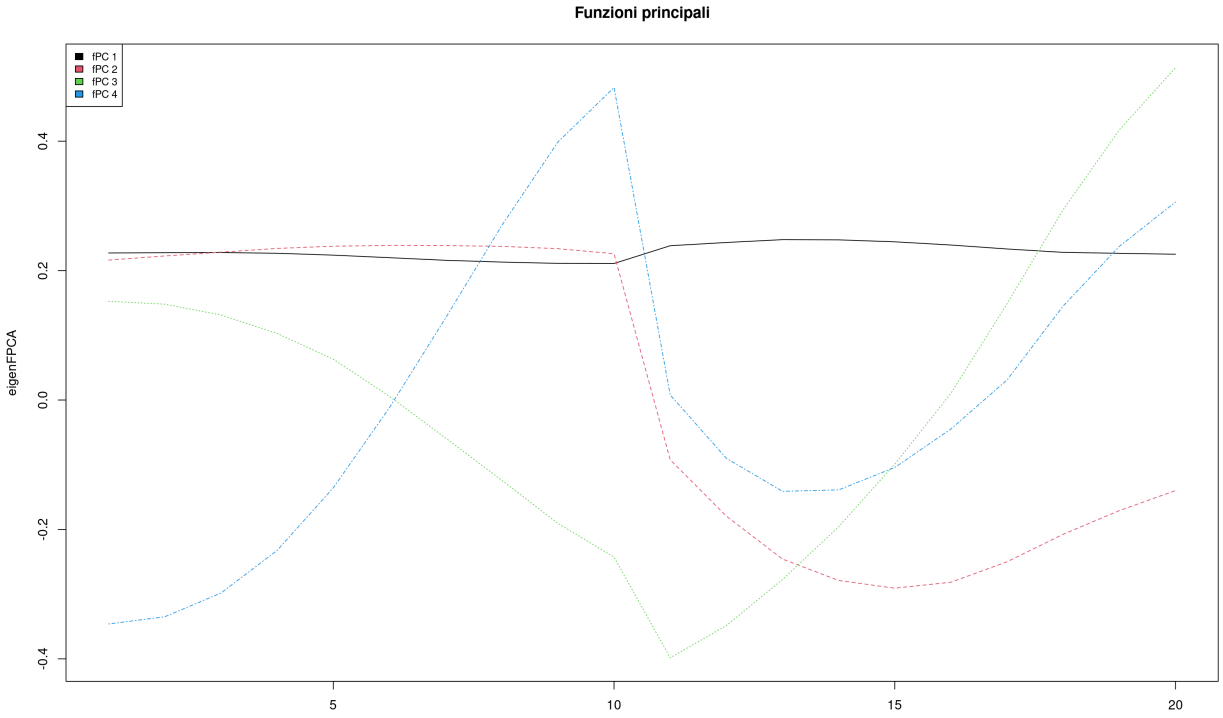
Stima delle funzioni principali che spieghino il 99% della varianza funzionale:

```
library(fdapace)

Y_list = lapply(seq_len(NROW(Y)), function(i) Y[i,])
t_list = lapply(seq_len(NROW(Y)), function(i) 1:20)

fpca = FPCA(Ly = Y_list, Lt = t_list, optns = list(FVEthreshold = fve))
K = fpca$selectK

ypred = predict(fpca, Y_list, t_list)
{
    # Confronto curve osservate con ricostruite tramite fPCA
    par(mfrow = c(2,1))
    matplot(t(Y), type = "l", main = "Funzioni osservate")
    matplot(t(ypred$predCurves), type = "l",
            main = paste0("Funzioni ricostruite con ", K, " fPC"))
    par(mfrow = c(1,1))
}
```





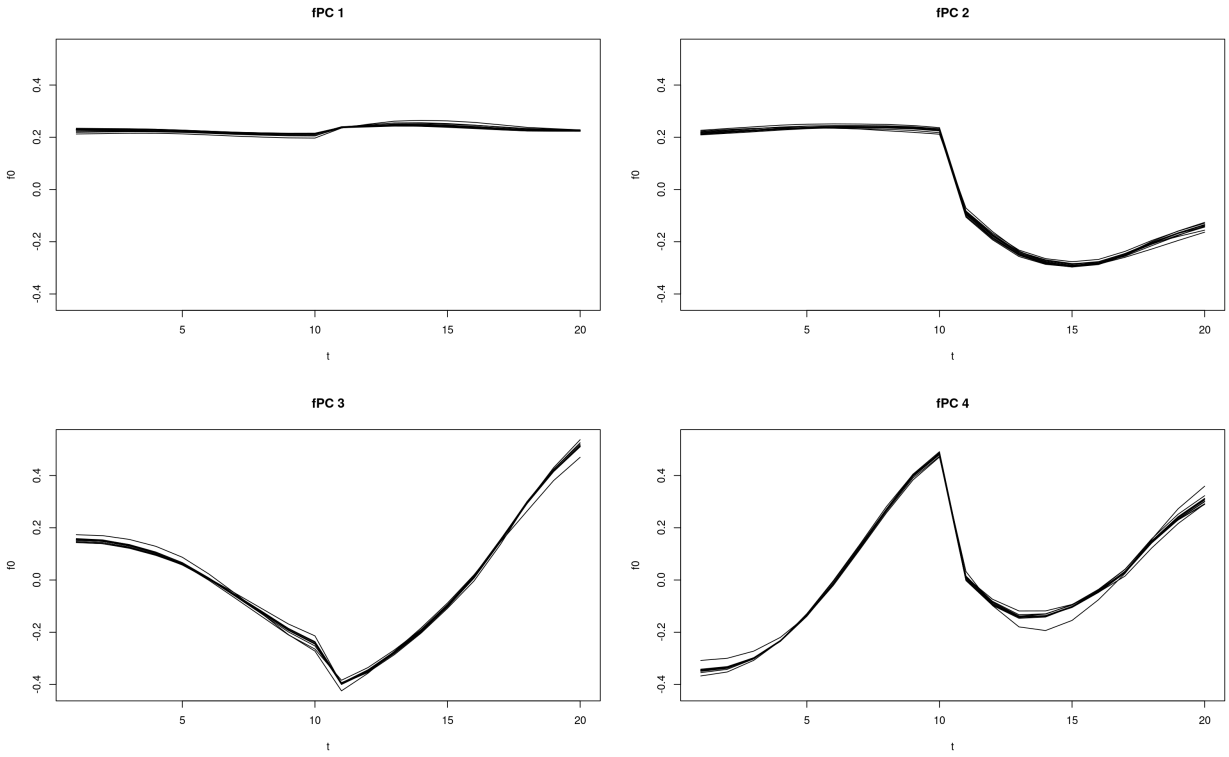

```r
# Autofunzioni stimate
eigenFPCA = fpca$phi
matplot(eigenFPCA, type = "l", main = "Funzioni principali")
legend("topleft", legend=paste0("fPC ", 1:K), col=1:K,cex=0.8, fill=1:K)
```



Variabilità delle funzioni principali rispetto alla rimozione dei soggetti.

```
mandarino$subject = factor(x = mandarino$subject, levels = mixedsort(levels(mandarino$subject)))
fpca_list = vector(mode = "list", length = length(unique(mandarino$subject)))
i = 1
for(subj in unique(mandarino$subject)){
    idx = mandarino_wide$subject == subj
    Y_sub = as.matrix(mandarino_wide[!idx , 5:(NCOL(mandarino_wide)) ])             # funzioni senza il
    Y_list_sub = lapply(seq_len(NROW(Y_sub)), function(i) Y_sub[i,])
    t_list_sub = lapply(seq_len(NROW(Y_sub)), function(i) 1:20)
    fpca_list[[i]] = FPCA(Ly = Y_list_sub, Lt = t_list_sub, optns = list(FVEthreshold = fve))
    i = i+1
}


plot_phi_list = function(fpca_list, m = 1){
    # Plot delle funzioni principali sovrapposte da una lista di oggetti fpca
    #
    # @param fpca_list: lista di fpca
    # @param m: ordine della funzione principale da plottare
    #
    # @return oggetto plot contenente il grafico sovrapposto delle fpc

    ymax = max(unlist(lapply(fpca_list, function(obj) max(obj$phi))))
    ymin = min(unlist(lapply(fpca_list, function(obj) min(obj$phi))))

    plot(fpca_list[[1]]$obsGrid, ylim = c(ymin, ymax), type = "n", xlab = "t", ylab = "f0", main = past
    for(i in 1:length(fpca_list)){
        lines(fpca_list[[i]]$obsGrid, fpca_list[[i]]$phi[ , m ])
```

```
    }
}

par(mfrow = c(2,2))
plot_phi_list(fpca_list, 1)
plot_phi_list(fpca_list, 2)
plot_phi_list(fpca_list, 3)
plot_phi_list(fpca_list, 4)
```




```
par(mfrow = c(1,1))

# Aggiungo le autofunzioni come covariate al dataset
eigenColumns = matrix(0, nrow = NROW(mandarino), ncol = K)
for(lv in 1:length(levels(mandarino$subject))){
    sj = levels(mandarino$subject)[lv]
    idx = (mandarino$subject == sj)
    print(sum(idx))
    for(k in 1:K){
        eigenColumns[idx, k] = rep(fpca_list[[lv]]$phi[ , k ], sum(idx) / 20)
    }
}
```

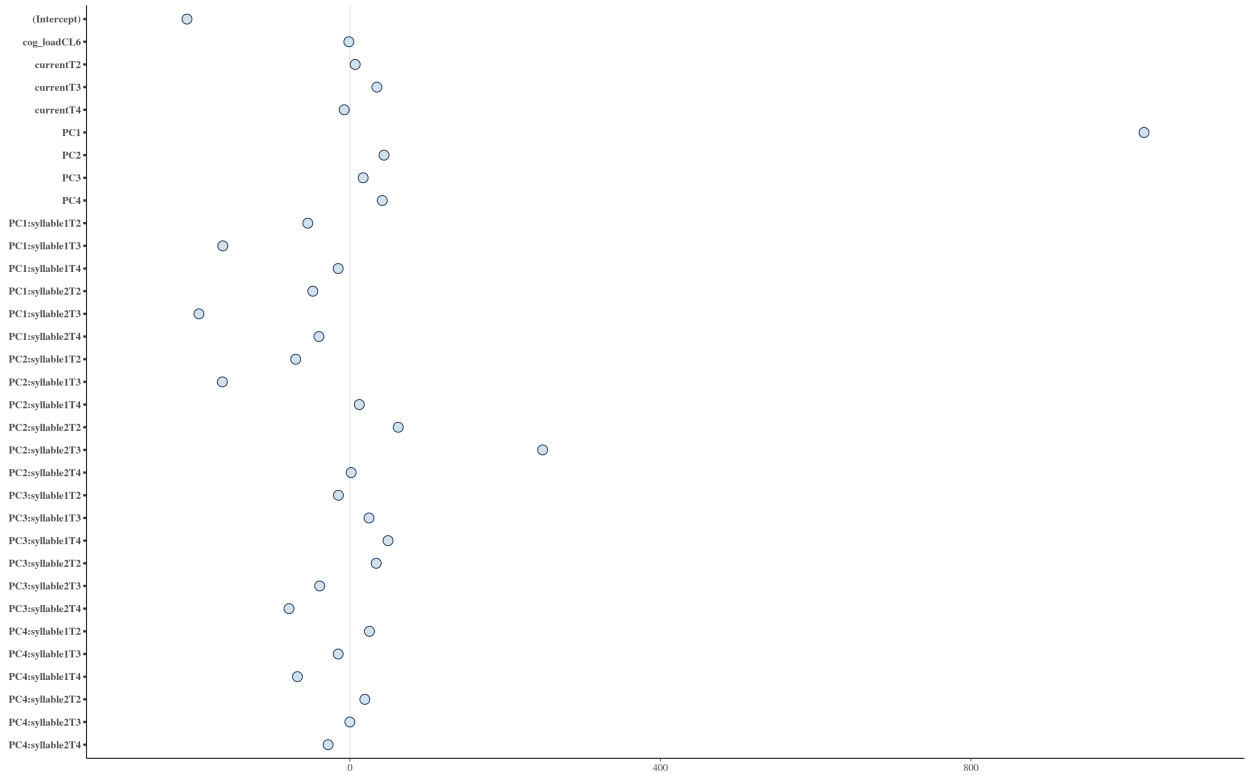
```
## [1] 640
## [1] 640
## [1] 620
## [1] 640
## [1] 640
## [1] 620
## [1] 640
```

```
## [1] 640
## [1] 640
## [1] 640
## [1] 640
## [1] 640
```

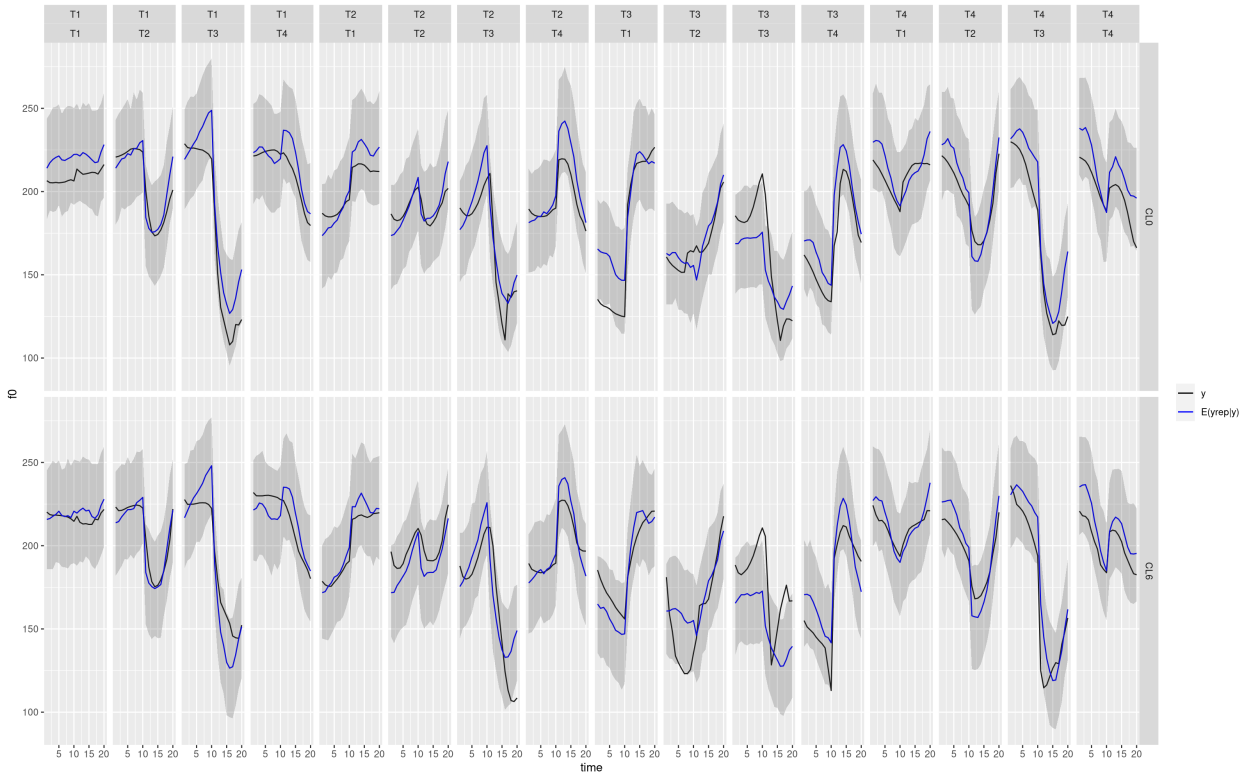
```
mandarino_fpca = cbind(mandarino, eigenColumns)
colnames(mandarino_fpca) = c(colnames(mandarino), paste0("PC", 1:K))
```

```
if(file.exists("fitStanLmerFpca.Rdata")){
    load("fitStanLmerFpca.Rdata")
} else{
    fitStanLmerFpca = stan_glmer(f0 ~ cog_load + current +
                                 PC1 + PC2 + PC3 + PC4 +
                                 PC1:syllable1 + PC1:syllable2 +
                                 PC2:syllable1 + PC2:syllable2 +
                                 PC3:syllable1 + PC3:syllable2 +
                                 PC4:syllable1 + PC4:syllable2 +
                                 (1 | subject) +
                                 (0 + current|subject) +
                                 (0 + PC1 + PC2 + PC3 + PC4|subject) +
                                 (0 + PC1:syllable1 | subject) +
                                 (0 + PC1:syllable2 | subject) +
                                 (0 + PC2:syllable1 | subject) +
                                 (0 + PC2:syllable2 | subject) +
                                 (0 + PC3:syllable1 | subject) +
                                 (0 + PC3:syllable2 | subject) +
                                 (0 + PC4:syllable1 | subject) +
                                 (0 + PC4:syllable2 | subject)
                                 ,
                             data = mandarino_fpca,
                        algorithm = "meanfield",
                        prior_covariance = decov(regularization = 2),
                        iter = 10000,
                        QR = TRUE)
    save(fitStanLmerFpca, file="fitStanLmerFpca.Rdata")
}
# summary(fitStanLmerFpca)
```

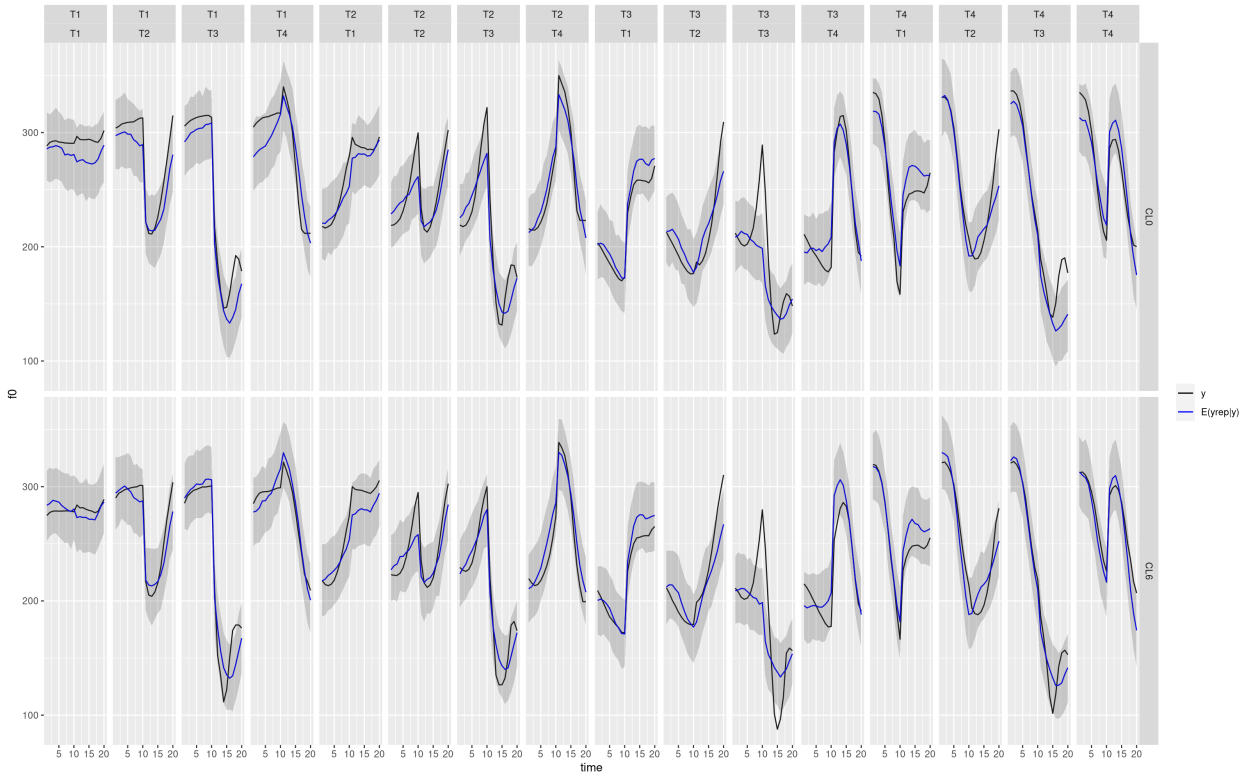
```
plot(fitStanLmerFpca, pars=names(fitStanLmerFpca$coefficients)[1:33])
```



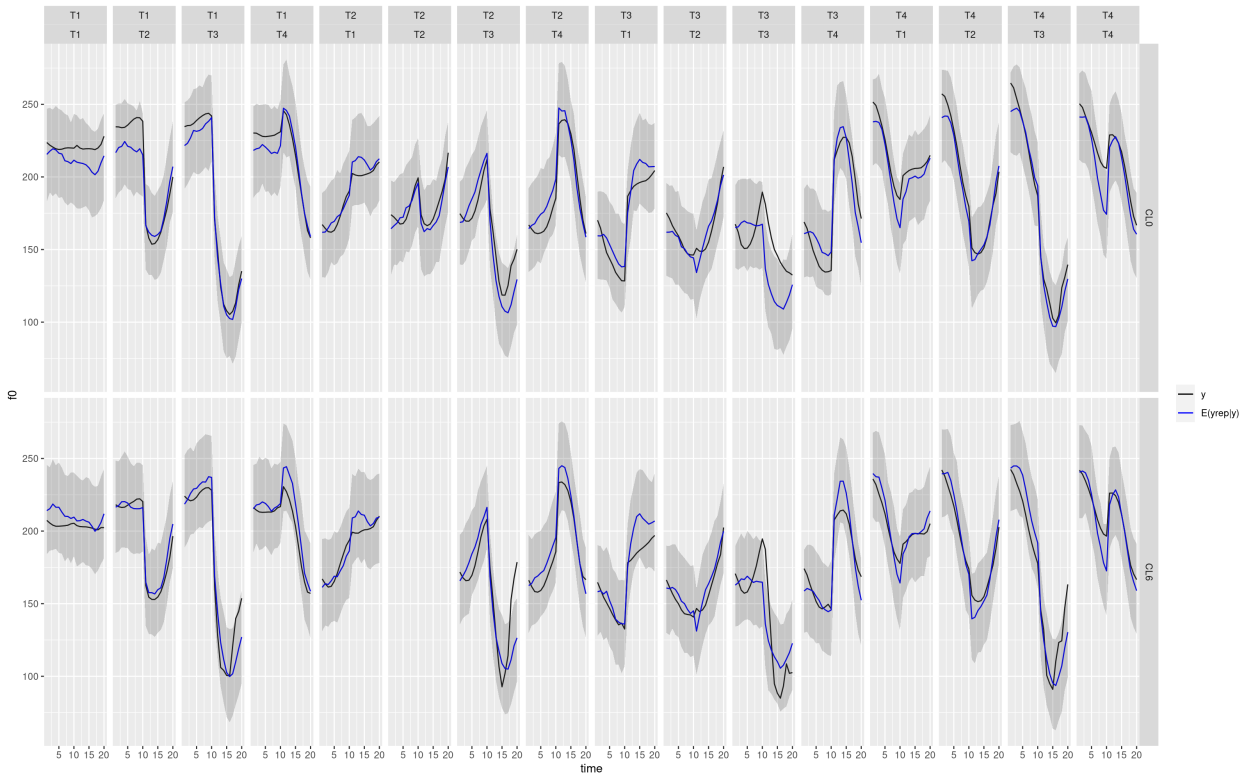

```
ypredStanLmerFpca = posterior_predict(fitStanLmerFpca, draws = 500)
ypred = apply(ypredStanLmerFpca, 2, mean)
stdev = apply(ypredStanLmerFpca, 2, sd)
lmin = apply(ypredStanLmerFpca, 2, function(col) quantile(col, p = 0.05))
lmax = apply(ypredStanLmerFpca, 2, function(col) quantile(col, p = 0.95))

ppc_subject("S1", mandarino, ypred, lmin, lmax)
```



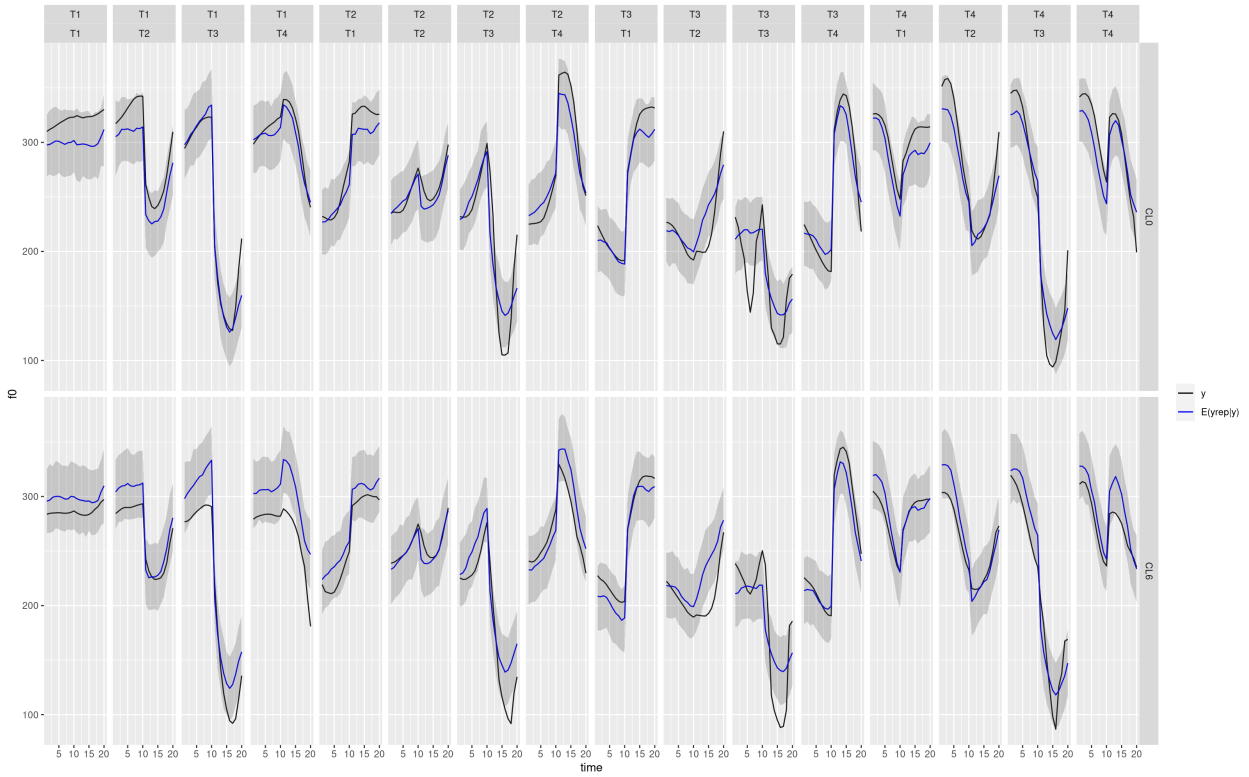

```
ppc_subject("S2", mandarino, ypred, lmin, lmax)
```



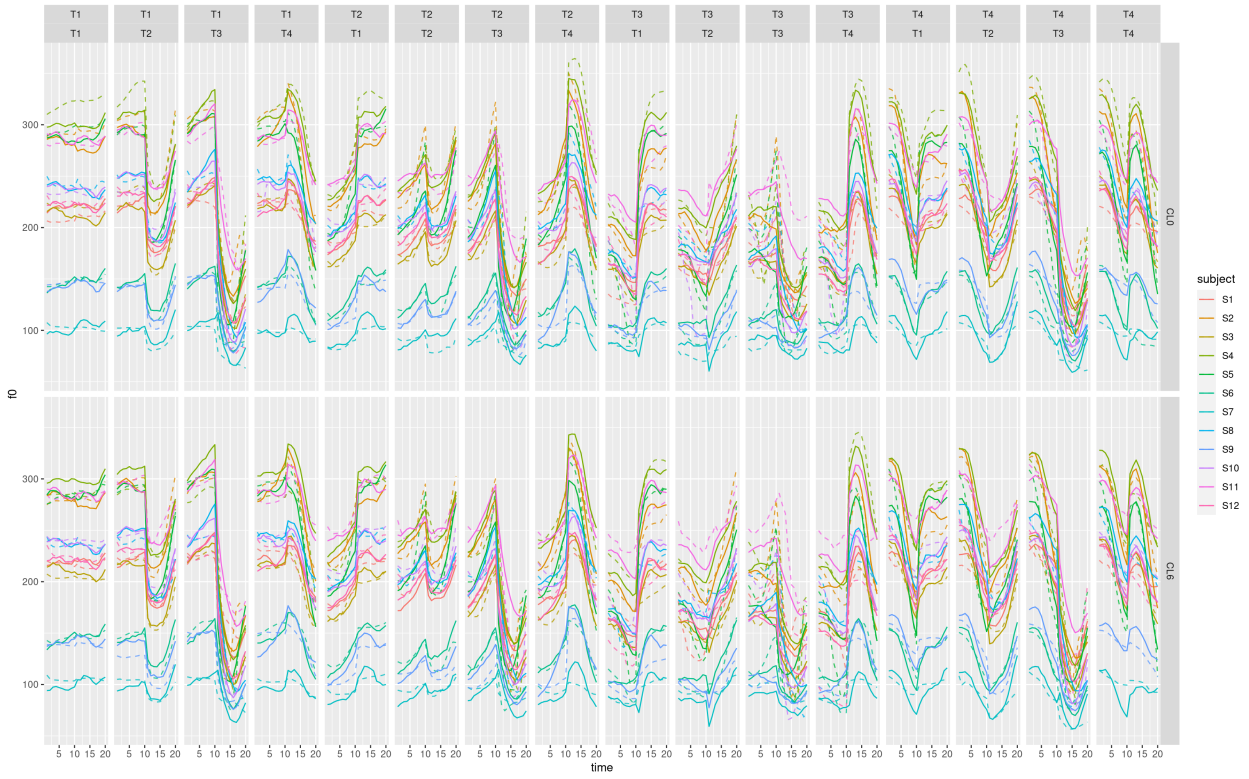
```
ppc_subject("S3", mandarino, ypred, lmin, lmax)
```




```
ppc_subject("S4", mandarino, ypred, lmin, lmax)
```



```
# ppc_subject("S5", mandarino, ypred, lmin, lmax)
# ppc_subject("S6", mandarino, ypred, lmin, lmax)
# ppc_subject("S7", mandarino, ypred, lmin, lmax)
# ppc_subject("S8", mandarino, ypred, lmin, lmax)
# ppc_subject("S9", mandarino, ypred, lmin, lmax)
# ppc_subject("S10", mandarino, ypred, lmin, lmax)
# ppc_subject("S11", mandarino, ypred, lmin, lmax)
# ppc_subject("S12", mandarino, ypred, lmin, lmax)

ppc_full(mandarino, ypred)
```



Selezione effetti fissi e casuali con k-fold

```
cv_stanvb = function(stanfit, K = 10, cores = min(K, 10), draws = 5){
    require("foreach")
    require("parallel")
    require("doParallel")

    cv_stanvb_fit = function(i, f, data, draws = 5){
        require("rstanarm")
        Y_train = data$y_train
        Y_test = data$y_test
        cat("Fit modello ", i)
        fit = stan_glmer(as.formula(f),
                         data = Y_train,
                         algorithm = "meanfield",
                         tol_rel_obj = 0.0001,
                         elbo_samples = 100,
```

```
                        grad_samples = 1,
                        iter = 10000,
                        adapt_delta = 0.8,
                        QR = TRUE)
        cat("Fit modello ", i, " completato\n\n")

        cat("Calcolo errore", i, "...\n")
        oss = Y_test[ , 7]
        pred = posterior_predict(fit, newdata=Y_test, draws)
        err = apply(pred, 1, function(row) mean((row - oss)^2))
        cat("Calcolo errore", i," completato\n\n")
        return(err)
    }

    cl = parallel::makeCluster(cores, outfile="")
    registerDoParallel(cl)
    out = vector(mode = "list", length = K)
    dati = stanfit$data
    rownames(dati) = 1:NROW(dati)
    f = stanfit$formula

    folds = cut(seq(1,nrow(dati)),breaks=10,labels=FALSE)
    folds = permute(folds)

    data_list = vector(mode = "list", length = K)
    for(i in 1:K){
        idx = folds != i
        data_list[[i]]$y_train = dati[idx, ]
        data_list[[i]]$y_test = dati[!idx, ]
    }

    err_list = foreach(i = 1:K, .inorder = FALSE) %dopar% {
        cv_stanvb_fit(i, f, data_list[[i]], draws)
    }
    parallel::stopCluster(cl)
    return(err_list)
}


#funzione per cv togliendo un soggetto alla volta
cv_stanvb_subj = function(stanfit, K = 12, cores = min(K, 12)){
  require("foreach")
  require("parallel")
  require("doParallel")

  cv_stanvb_fit = function(i, f, data, ...){
    require("rstanarm")
    Y_train = data$y_train
    Y_test = data$y_test
    cat("Fit modello ", i)
    fit = stan_glmer(as.formula(f),
                     data = Y_train,
                     algorithm = "meanfield",
```

```
                    tol_rel_obj = 0.0001,
                    elbo_samples = 100,
                    grad_samples = 1,
                    iter = 10000,
                    QR = TRUE)
    cat("Fit modello ", i, " completato\n\n")

    cat("Calcolo errore", i, "...\n")
    oss = Y_test[ , 7]
    pred = posterior_predict(fit, newdata=Y_test, draws = 10)
    err = apply(pred, 1, function(row) mean((row - oss)^2))
    cat("Calcolo errore", i," completato\n\n")
    return(err)
  }

  cl = parallel::makeCluster(cores, outfile="")
  registerDoParallel(cl)
  out = vector(mode = "list", length = K)
  dati = stanfit$data
  rownames(dati) = 1:NROW(dati)
  f = stanfit$formula

  data_list = vector(mode = "list", length = K)
  subj = mixedsort(levels(dati$subject))
  for(i in 1:K){
    data_list[[i]]$y_train = dati[dati$subject != subj[i], ]
    data_list[[i]]$y_test = dati[dati$subject == subj[i], ]
  }

  err_list = foreach(i = 1:K, .inorder = FALSE) %dopar% {
    cv_stanvb_fit(i, f, data_list[[i]])
  }
  parallel::stopCluster(cl)
  return(err_list)
}
```

```
cores  = min(4, parallel::detectCores())
K = 10                                                  # K-fold cross-validation
kfcores = ifelse(parallel::detectCores() >= K, K, cores)    # cores per cv

if(file.exists("fitStanLmerFpca1.Rdata")){
    load("fitStanLmerFpca1.Rdata")
} else{
    fitStanLmerFpca1 = stan_glmer(f0 ~ cog_load + current + syllable1 + syllable2 + syllable1:syllable2
                                  PC1 + PC2 + PC3 + PC4 +
                                  PC1:syllable1 + PC1:syllable2 +
                                  PC2:syllable1 + PC2:syllable2 +
                                  PC3:syllable1 + PC3:syllable2 +
                                  PC4:syllable1 + PC4:syllable2 +
                                  (1 | subject) +
                                  (0 + current|subject) +
                                  (0 + PC1 + PC2 + PC3 + PC4|subject)
                                ,
                                data = mandarino_fpca,
```

```
                                algorithm = "meanfield", tol_rel_obj = 0.0001, elbo_samples = 100, grad_sa
                                iter = 10000,
                                #chains = cores, cores = cores,
                                QR = TRUE)

    save(fitStanLmerFpca1, file="fitStanLmerFpca1.Rdata")
}
```

```
if(file.exists("fitStanLmerFpca4.Rdata")){
    load("fitStanLmerFpca4.Rdata")
} else{
    fitStanLmerFpca4 = stan_glmer(f0 ~ current + cog_load +
                                    syllable1 + syllable2 + syllable1:syllable2 +
                                    PC1 + PC2 + PC3 + PC4 +
                                    PC1:syllable1 + PC1:syllable2 +
                                    PC2:syllable1 + PC2:syllable2 +
                                    PC3:syllable1 + PC3:syllable2 +
                                    PC4:syllable1 + PC4:syllable2 +
                                    (1 | subject) +
                                    (0 + current|subject) +
                                    (0 + PC1 + PC2 + PC3 + PC4|subject) +
                                    (0 + PC1:syllable1 | subject) +
                                    (0 + PC1:syllable2 | subject) +
                                    (0 + PC2:syllable1 | subject) +
                                    (0 + PC2:syllable2 | subject) +
                                    (0 + PC3:syllable1 | subject) +
                                    (0 + PC3:syllable2 | subject) +
                                    (0 + PC4:syllable1 | subject) +
                                    (0 + PC4:syllable2 | subject)
                                ,
                                data = mandarino_fpca,
                                algorithm = "meanfield", tol_rel_obj = 0.0001, elbo_samples = 100, grad_sa
                                iter = 10000,
                                QR = TRUE)
    save(fitStanLmerFpca4, file="fitStanLmerFpca4.Rdata")
}
```

```
if(file.exists("fitStanLmerFpca5.Rdata")){
    load("fitStanLmerFpca5.Rdata")
} else{
    fitStanLmerFpca5 = stan_glmer(f0 ~ current + cog_load +
                                    syllable1 + syllable2 + syllable1:syllable2 +
                                    PC1 + PC2 + PC3 + PC4 +
                                    PC1:syllable1 + PC1:syllable2 +
                                    PC2:syllable1 + PC2:syllable2 +
                                    PC3:syllable1 + PC3:syllable2 +
                                    PC4:syllable1 + PC4:syllable2 +
                                    (1 | subject) +
                                    (0 + current|subject) +
                                    (0 + syllable1 + syllable2 + syllable1:syllable2 | subject) +
                                    (0 + PC1 + PC2 + PC3 + PC4|subject) +
                                    (0 + PC1:syllable1 | subject) +
                                    (0 + PC1:syllable2 | subject) +
                                    (0 + PC2:syllable1 | subject) +
```

```
                                    (0 + PC2:syllable2 | subject) +
                                    (0 + PC3:syllable1 | subject) +
                                    (0 + PC3:syllable2 | subject) +
                                    (0 + PC4:syllable1 | subject) +
                                    (0 + PC4:syllable2 | subject)
                                ,
                                data = mandarino_fpca,
                                algorithm = "meanfield", tol_rel_obj = 0.0001, elbo_samples = 100, grad_sa
                                prior_covariance = decov(regularization = 2),
                                iter = 10000,
                                QR = TRUE)
    save(fitStanLmerFpca5, file="fitStanLmerFpca5.Rdata")
}
```

```
if(file.exists("fitStanLmerFpca6.Rdata")){
    load("fitStanLmerFpca6.Rdata")
} else{
    fitStanLmerFpca6 = stan_glmer(f0 ~ current + cog_load +
                                    syllable1 + syllable2 + syllable1:syllable2 +
                                    PC1 + PC2 + PC3 + PC4 +
                                    PC1:syllable1 + PC1:syllable2 +
                                    PC2:syllable1 + PC2:syllable2 +
                                    PC3:syllable1 + PC3:syllable2 +
                                    PC4:syllable1 + PC4:syllable2 +
                                    PC1:syllable1:syllable2 +
                                    PC2:syllable1:syllable2 +
                                    PC3:syllable1:syllable2 +
                                    PC4:syllable1:syllable2 +
                                    (1 | subject) +
                                    (0 + current|subject) +
                                    (0 + PC1 + PC2 + PC3 + PC4|subject) +
                                    (0 + PC1:syllable1 | subject) +
                                    (0 + PC1:syllable2 | subject) +
                                    (0 + PC2:syllable1 | subject) +
                                    (0 + PC2:syllable2 | subject) +
                                    (0 + PC3:syllable1 | subject) +
                                    (0 + PC3:syllable2 | subject) +
                                    (0 + PC4:syllable1 | subject) +
                                    (0 + PC4:syllable2 | subject)
                                ,
                                data = mandarino_fpca,
                                algorithm = "meanfield", tol_rel_obj = 0.0001, elbo_samples = 100, grad_sa
                                iter = 10000,
                                QR = TRUE)
    save(fitStanLmerFpca6, file="fitStanLmerFpca6.Rdata")
}
```

```
if(file.exists("fitStanLmerFpca2.Rdata")){
    load("fitStanLmerFpca2.Rdata")
} else{
    fitStanLmerFpca2 = stan_glmer(f0 ~ current +
                                PC1 + PC2 + PC3 + PC4 +
                                PC1:syllable1 + PC1:syllable2 +
                                PC2:syllable1 + PC2:syllable2 +
```

```
                                PC3:syllable1 + PC3:syllable2 +
                                PC4:syllable1 + PC4:syllable2 +
                                (1 | subject) +
                                (0 + current|subject) +
                                (0 + PC1 + PC2 + PC3 + PC4|subject) +
                                (0 + PC1:syllable1 | subject) +
                                (0 + PC1:syllable2 | subject) +
                                (0 + PC2:syllable1 | subject) +
                                (0 + PC2:syllable2 | subject) +
                                (0 + PC3:syllable1 | subject) +
                                (0 + PC3:syllable2 | subject) +
                                (0 + PC4:syllable1 | subject) +
                                (0 + PC4:syllable2 | subject)
                                ,
                            data = mandarino_fpca,
                       algorithm = "meanfield",
                       iter = 10000,
                       QR = TRUE)
    save(fitStanLmerFpca2, file="fitStanLmerFpca2.Rdata")
}
```

```
if(file.exists("fitStanLmerFpca3.Rdata")){
    load("fitStanLmerFpca3.Rdata")
} else{
    fitStanLmerFpca3 = stan_glmer(f0 ~ PC1 + PC2 + PC3 + PC4 +
                                    PC1:current + PC2:current + PC3:current + PC4:current+
                                PC1:syllable1 + PC1:syllable2 +
                                PC2:syllable1 + PC2:syllable2 +
                                PC3:syllable1 + PC3:syllable2 +
                                PC4:syllable1 + PC4:syllable2 +
                                (1 | subject) +
                                (0 + PC1 + PC2 + PC3 + PC4|subject) +
                                (0 + PC1:syllable1 | subject) +
                                (0 + PC1:syllable2 | subject) +
                                (0 + PC2:syllable1 | subject) +
                                (0 + PC2:syllable2 | subject) +
                                (0 + PC3:syllable1 | subject) +
                                (0 + PC3:syllable2 | subject) +
                                (0 + PC4:syllable1 | subject) +
                                (0 + PC4:syllable2 | subject)
                                ,
                            data = mandarino_fpca,
                       algorithm = "meanfield",
                       prior_covariance = decov(regularization = 2),
                       iter = 10000,
                       QR = TRUE)
    save(fitStanLmerFpca3, file="fitStanLmerFpca3.Rdata")
}
```

```
{
    print("--------- kf ---------")
    if(file.exists("kf.Rdata")){
        load("kf.Rdata")
    } else{
```

```
    kf = cv_stanvb(fitStanLmerFpca, K = K, cores = kfcores)
    kf = lapply(kf, mean)
    save(kf, file = "kf.Rdata")
}
# kf = kf[-9]             # algoritmo 9 non è andato a convergenza
print(cbind("mean" = mean(unlist(kf)), "sd" = sd(unlist(kf))))

print("--------- kf1 ---------")
if(file.exists("kf1.Rdata")){
    load("kf1.Rdata")
} else{
    kf1 = cv_stanvb(fitStanLmerFpca1, K = K, cores = kfcores)
    kf1 = lapply(kf1, mean)
    save(kf1, file = "kf1.Rdata")
}
print(cbind("mean" = mean(unlist(kf1)), "sd" = sd(unlist(kf1))))

print("--------- kf2 ---------")
if(file.exists("kf2.Rdata")){
    load("kf2.Rdata")
} else{
    kf2 = cv_stanvb(fitStanLmerFpca2, K = K, cores = kfcores)
    kf2 = lapply(kf2, mean)
    save(kf2, file = "kf2.Rdata")
}

print(cbind("mean" = mean(unlist(kf2)), "sd" = sd(unlist(kf2))))

print("--------- kf3 ---------")
if(file.exists("kf3.Rdata")){
    load("kf3.Rdata")
} else{
    kf3 = cv_stanvb(fitStanLmerFpca3, K = K, cores = kfcores)
    kf3 = lapply(kf3, mean)
    save(kf3, file = "kf3.Rdata")
}
print(cbind("mean" = mean(unlist(kf3)), "sd" = sd(unlist(kf3))))

print("--------- kf4 ---------")
if(file.exists("kf4.Rdata")){
    load("kf4.Rdata")
} else{
    kf4 = cv_stanvb(fitStanLmerFpca4, K = K, cores = kfcores)
    kf4 = lapply(kf4, mean)
    save(kf4, file = "kf4.Rdata")
}
print(cbind("mean" = mean(unlist(kf4)), "sd" = sd(unlist(kf4))))

print("--------- kf5 ---------")
if(file.exists("kf5.Rdata")){
    load("kf5.Rdata")
} else{
    kf5 = cv_stanvb(fitStanLmerFpca5, K = K, cores = kfcores)
```

```
        kf5 = lapply(kf5, mean)
        save(kf5, file = "kf5.Rdata")
    }
    print(cbind("mean" = mean(unlist(kf5)), "sd" = sd(unlist(kf5))))

    # print("--------- kf6 ---------")
    # if(file.exists("kf6.Rdata")){
    #     load("kf6.Rdata")
    # } else{
    #     kf6 = cv_stanvb(fitStanLmerFpca6, K = K, cores = kfcores)
    #     save(kf6, file = "kf6.Rdata")
    # }
    # print(cbind("mean" = mean(unlist(kf6)), "sd" = sd(unlist(kf6))))
}
```

```
## [1] "--------- kf ---------"
##          mean       sd
## [1,] 410.8675 23.89217
## [1] "--------- kf1 ---------"
##          mean       sd
## [1,] 636.7633 51.36843
## [1] "--------- kf2 ---------"
##          mean       sd
## [1,] 426.7653 35.54432
## [1] "--------- kf3 ---------"
##          mean       sd
## [1,] 428.1031 37.72624
## [1] "--------- kf4 ---------"
##          mean       sd
## [1,] 388.5147 32.41829
## [1] "--------- kf5 ---------"
##          mean      sd
## [1,] 367.7848 32.8677
```

```
rbind(cbind(mean(unlist(kf5)), sd(unlist(kf5))/sqrt(10)),
      cbind(mean(unlist(kf4)), sd(unlist(kf4))/sqrt(10)),
      cbind(mean(unlist(kf)), sd(unlist(kf))/sqrt(10)),
      cbind(mean(unlist(kf2)), sd(unlist(kf2))/sqrt(10)),
      cbind(mean(unlist(kf3)), sd(unlist(kf3))/sqrt(10)),
      cbind(mean(unlist(kf1)), sd(unlist(kf1))/sqrt(10))
      ) %>%
    set_rownames(c("Eff. misti syllable e PCA:syllable, fissi syllable PCA:syllable",
                   "Eff. misti PCA:syllable, fissi syllable PCA:syllable",
                   "Eff. misti PCA:syllable, fissi no syllable",
                   "Eff. misti PCA:syllable, fissi no syllable no cog_load",
                   "Eff. misti PCA:syllable, fissi no syllable no cog_load no current",
                   "No eff. misti PCA:syllable, fissi syllable PCA:syllable"
                   )) %>%
    set_colnames(c("mse", "sd")) -> fpcaKf

fpcaKf
```

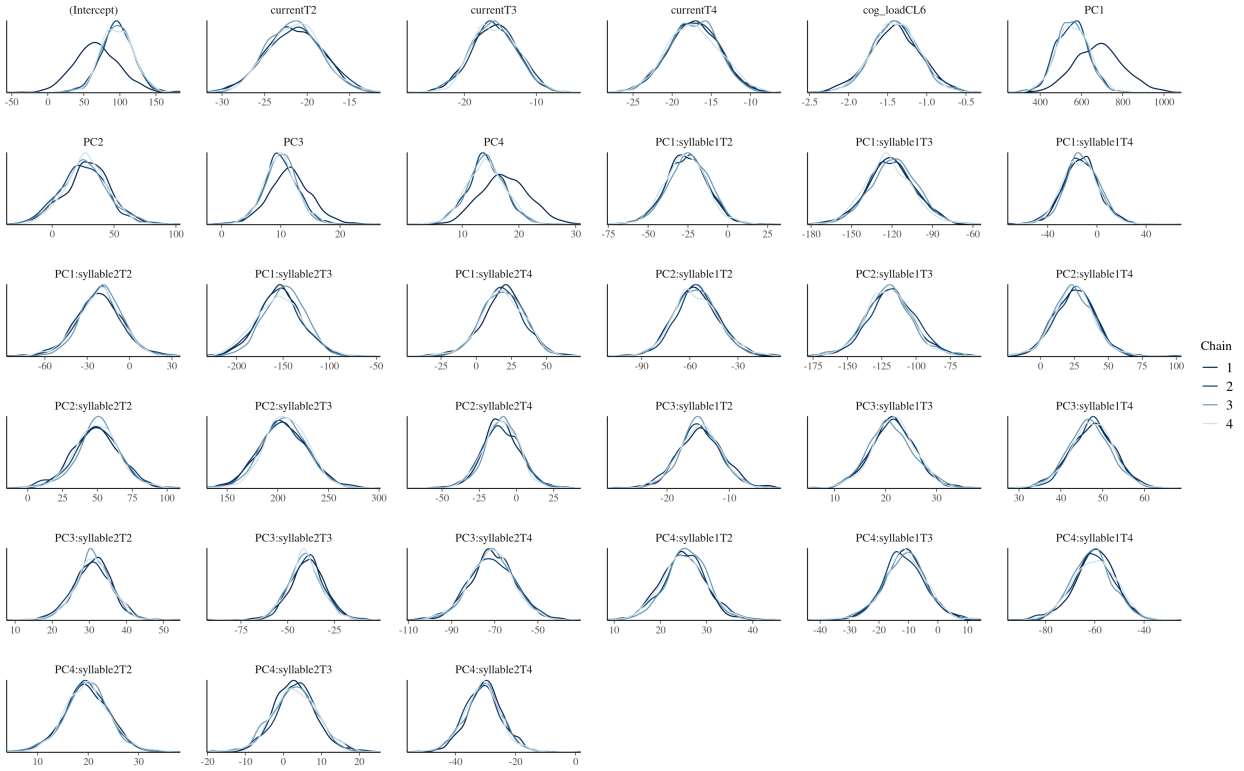
```
##                                                                      mse        sd
## Eff. misti syllable e PCA:syllable, fissi syllable PCA:syllable   367.7848 10.393680
```

```
## Eff. misti PCA:syllable, fissi syllable PCA:syllable              388.5147 10.251564
## Eff. misti PCA:syllable, fissi no syllable                        410.8675  7.555366
## Eff. misti PCA:syllable, fissi no syllable no cog_load            426.7653 11.240101
## Eff. misti PCA:syllable, fissi no syllable no cog_load no current 428.1031 11.930085
## No eff. misti PCA:syllable, fissi syllable PCA:syllable           636.7633 16.244124
```

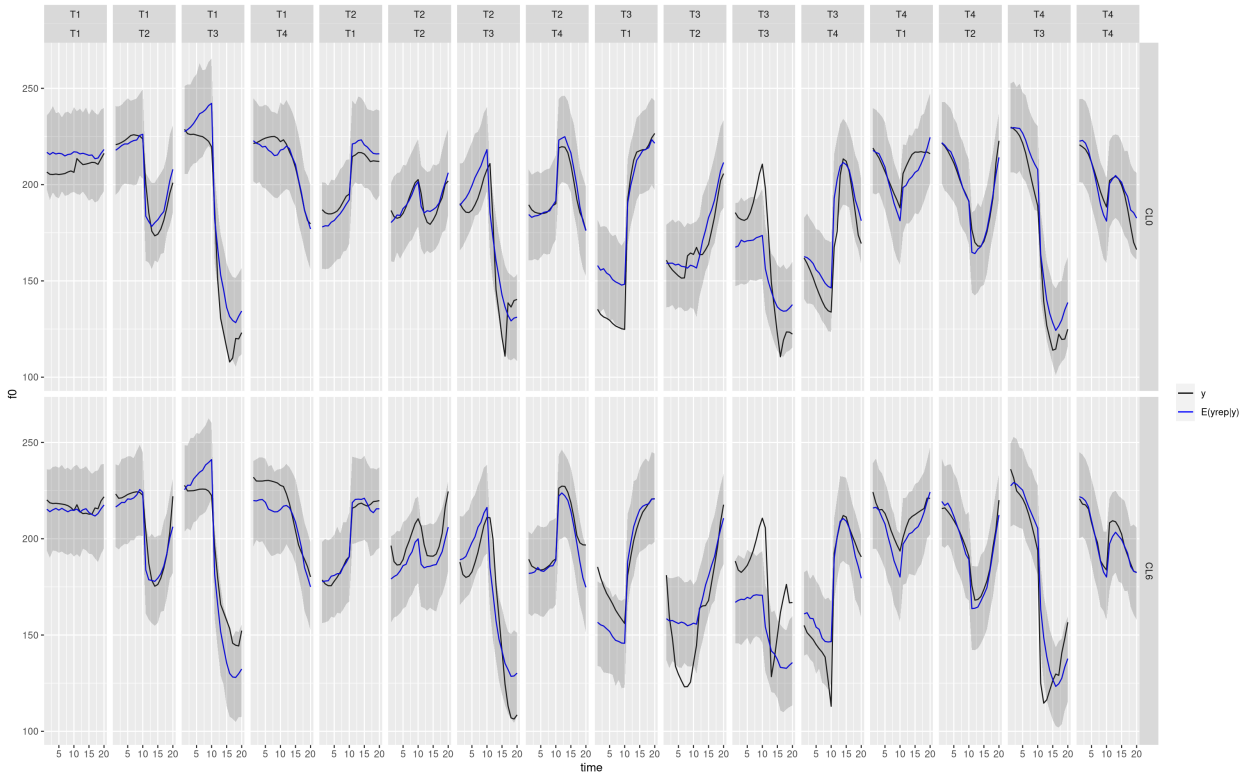
```r
#funzione per cv togliendo un soggetto alla volta
cv_stanvb_subj = function(stanfit, K = 12, cores = min(K, 12)){
  require("foreach")
  require("parallel")
  require("doParallel")

  cv_stanvb_fit = function(i, f, data, ...){
    require("rstanarm")
    Y_train = data$y_train
    Y_test = data$y_test
    cat("Fit modello ", i)
    fit = stan_glmer(as.formula(f),
                     data = Y_train,
                     algorithm = "meanfield",
                     tol_rel_obj = 0.0001,
                     elbo_samples = 100,
                     grad_samples = 1,
                     iter = 10000,
                     QR = TRUE)
    cat("Fit modello ", i, " completato\n\n")

    cat("Calcolo errore", i, "...\n")
    oss = Y_test[ , 7]
    pred = posterior_predict(fit, newdata=Y_test, draws = 5)
    err = apply(pred, 1, function(row) mean((row - oss)^2))
    cat("Calcolo errore", i," completato\n\n")
    return(err)
  }

  cl = parallel::makeCluster(cores, outfile="")
  registerDoParallel(cl)
  out = vector(mode = "list", length = K)
  dati = stanfit$data
  rownames(dati) = 1:NROW(dati)
  f = stanfit$formula

  data_list = vector(mode = "list", length = K)
  subj = mixedsort(levels(dati$subject))
  for(i in 1:K){
    data_list[[i]]$y_train = dati[dati$subject != subj[i], ]
    data_list[[i]]$y_test = dati[dati$subject == subj[i], ]
  }

  err_list = foreach(i = 1:K, .inorder = FALSE) %dopar% {
    cv_stanvb_fit(i, f, data_list[[i]])
  }
  parallel::stopCluster(cl)
  return(err_list)
```

```
}
```

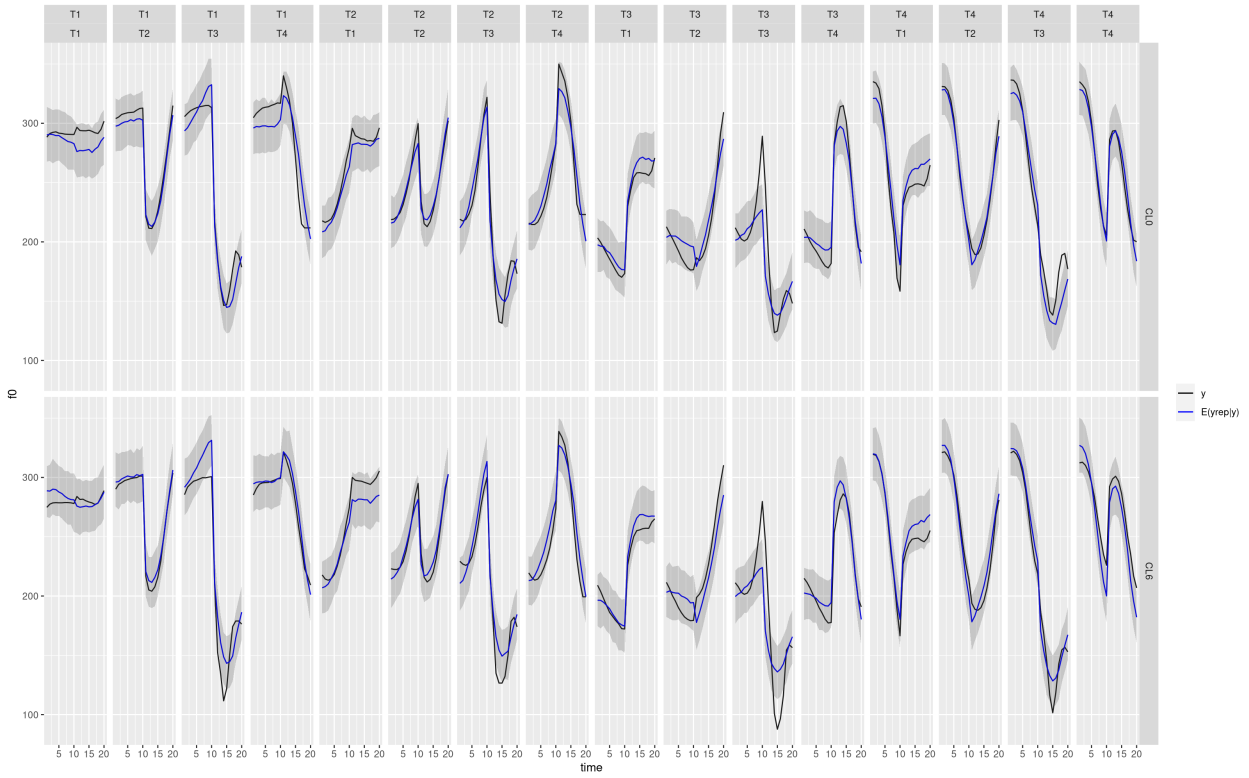
```
if(file.exists("fitStanFpcaFinal.Rdata")){
    load("fitStanFpcaFinal.Rdata")
} else{
    fitStanFpcaFinal = stan_glmer(f0 ~ current + cog_load +
                                  PC1 + PC2 + PC3 + PC4 +
                                  PC1:syllable1 + PC1:syllable2 +
                                  PC2:syllable1 + PC2:syllable2 +
                                  PC3:syllable1 + PC3:syllable2 +
                                  PC4:syllable1 + PC4:syllable2 +
                                  (1 | subject) +
                                  (0 + PC1 + PC2 + PC3 + PC4|subject) +
                                  (0 + PC1:syllable1 | subject) +
                                  (0 + PC1:syllable2 | subject) +
                                  (0 + PC2:syllable1 | subject) +
                                  (0 + PC2:syllable2 | subject) +
                                  (0 + PC3:syllable1 | subject) +
                                  (0 + PC3:syllable2 | subject) +
                                  (0 + PC4:syllable1 | subject) +
                                  (0 + PC4:syllable2 | subject)
                                  ,
                                  data = mandarino_fpca,
                                  prior_covariance = decov(regularization = 2),
                                  cores = 4,
                                  chains = 4,
                                  control = list(max_treedepth = 10),
                                  adapt_delta = 0.825,
                                  QR = TRUE)
    save(fitStanFpcaFinal, file="fitStanFpcaFinal.Rdata")
}
plot(fitStanFpcaFinal, plotfun = "dens_overlay", pars=names(fitStanFpcaFinal$coefficients)[1:33])
```



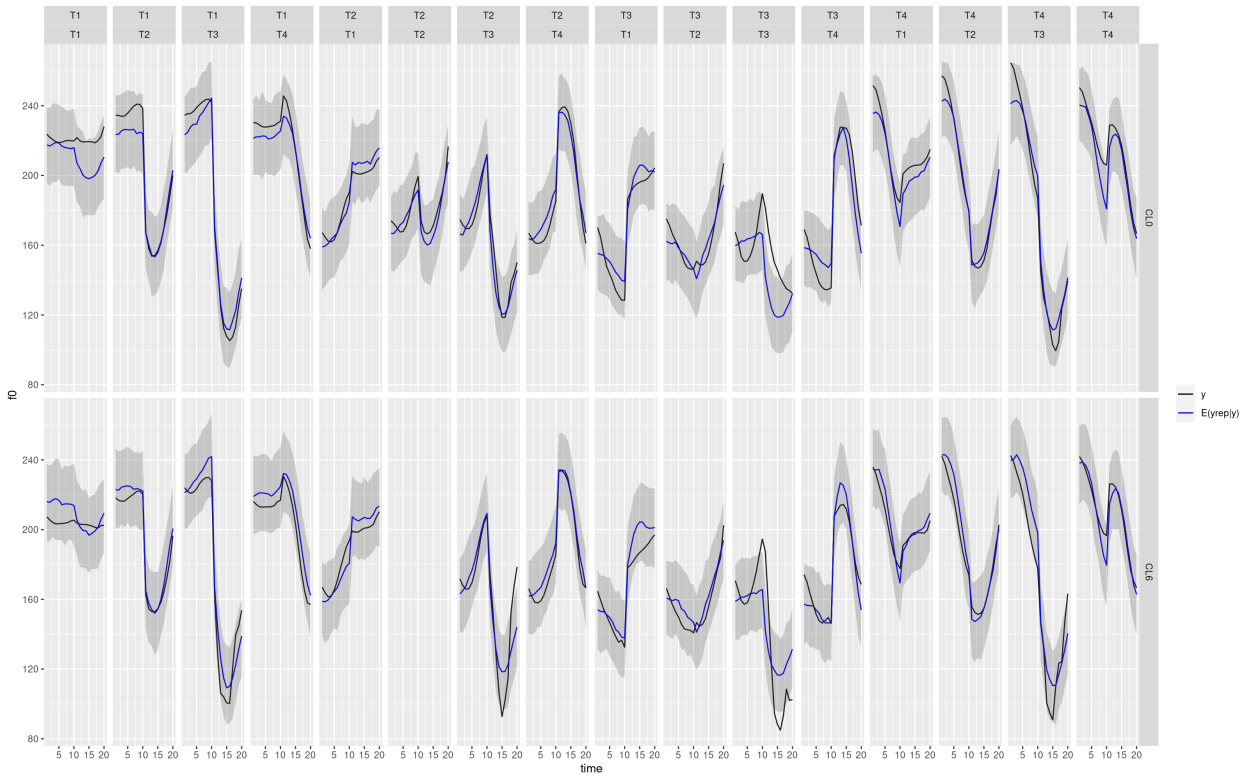

```
ypredStanFpca = posterior_predict(fitStanFpcaFinal, draws = 500)
ypred = apply(ypredStanFpca, 2, mean)
lmin = apply(ypredStanFpca, 2, function(col) quantile(col, p = 0.05))
lmax = apply(ypredStanFpca, 2, function(col) quantile(col, p = 0.95))

ppc_subject("S1", mandarino, ypred, lmin, lmax)
```



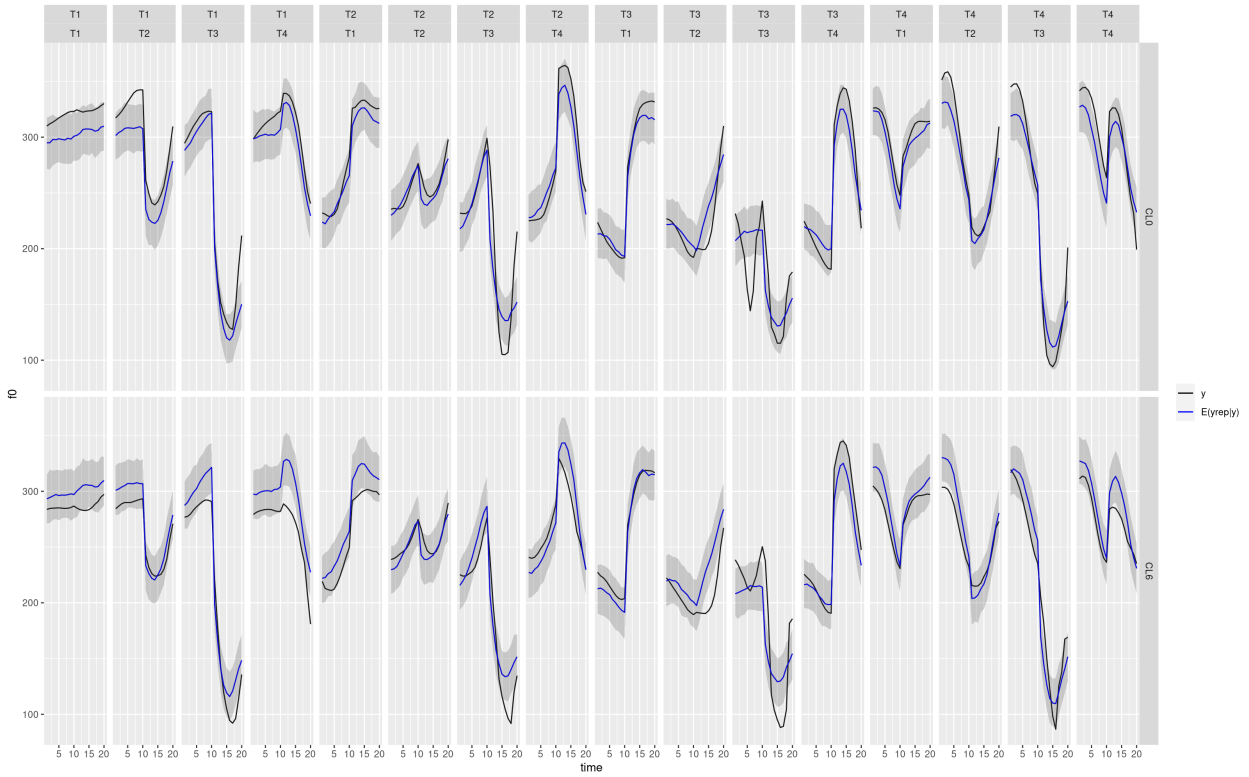

```
ppc_subject("S2", mandarino, ypred, lmin, lmax)
```



```
ppc_subject("S3", mandarino, ypred, lmin, lmax)
```




```
ppc_subject("S4", mandarino, ypred, lmin, lmax)
```



```
# ppc_subject("S5", mandarino, ypred, lmin, lmax)
# ppc_subject("S6", mandarino, ypred, lmin, lmax)
# ppc_subject("S7", mandarino, ypred, lmin, lmax)
# ppc_subject("S8", mandarino, ypred, lmin, lmax)
# ppc_subject("S9", mandarino, ypred, lmin, lmax)
# ppc_subject("S10", mandarino, ypred, lmin, lmax)
# ppc_subject("S11", mandarino, ypred, lmin, lmax)
# ppc_subject("S12", mandarino, ypred, lmin, lmax)

ppc_full(mandarino, ypred)
```