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(tidyverse)
```

Preprocessing

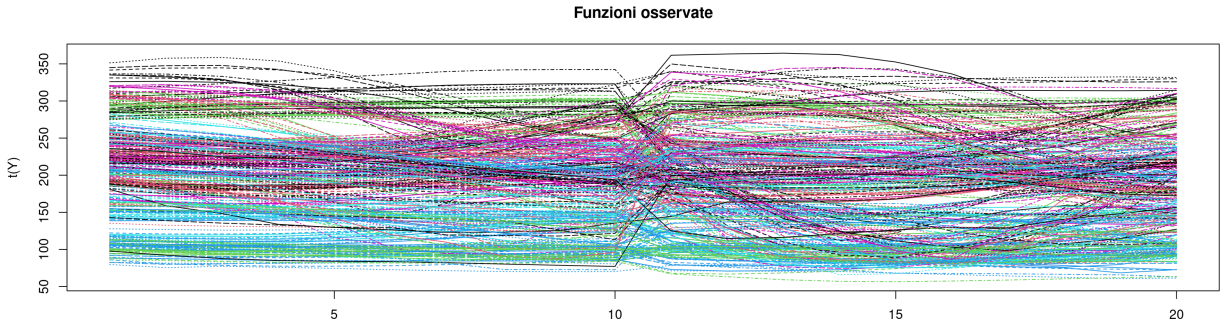
Costruiamo il dataset in formato "largo" per la stima delle fPCA:

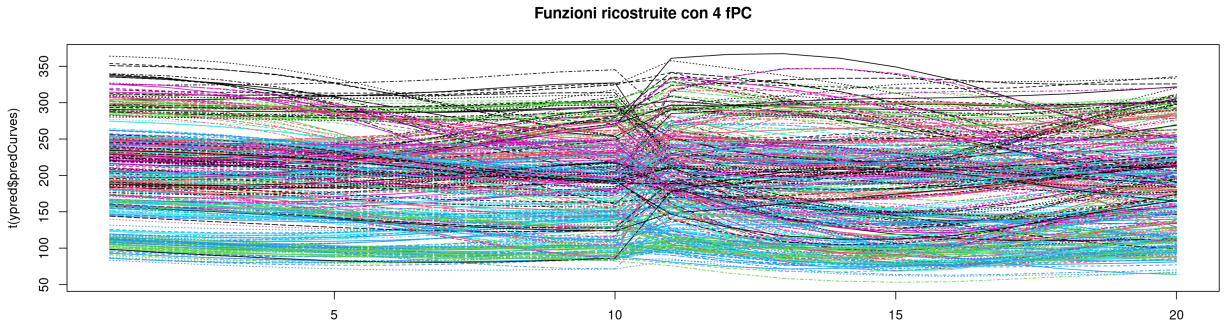
```
mandarino_wide = spread(mandarino[ , -6 ], time, "f0")
Y = as.matrix(mandarino_wide[ , 5:(NCOL(mandarino_wide)) ])     # Funzioni osservate

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

Analisi con componenti principali funzionali (fPCA)

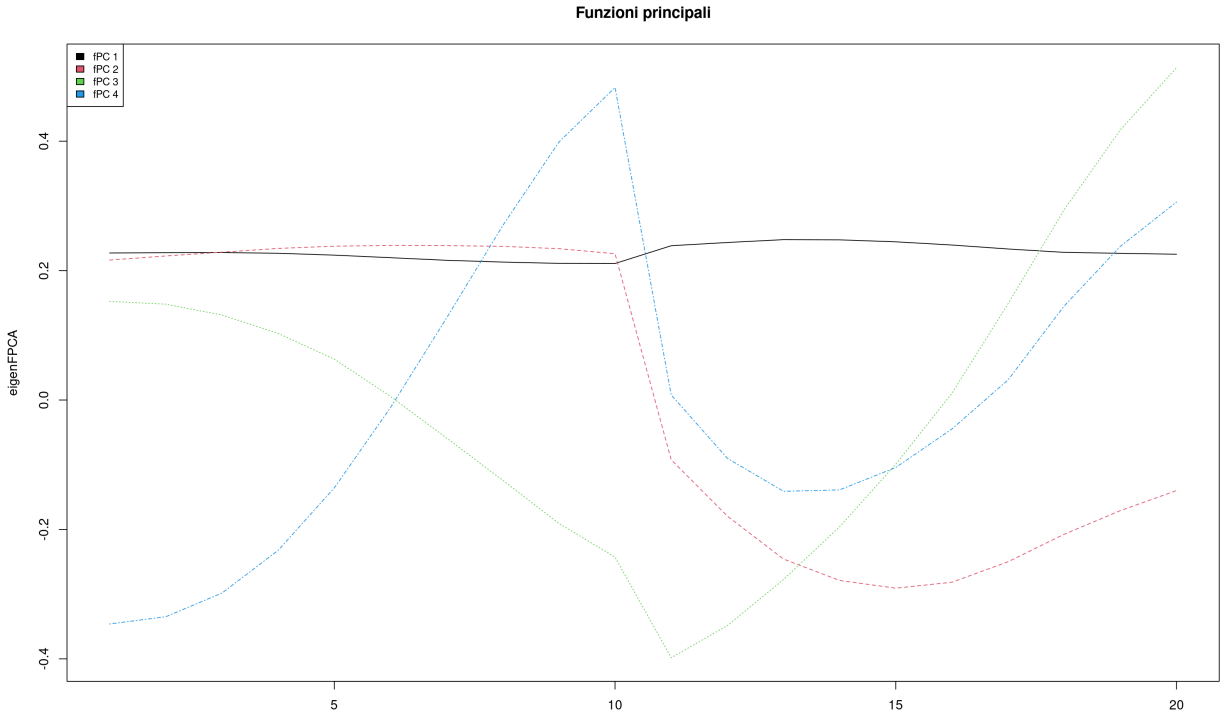
Stima delle funzioni principali che spieghino il 99% della varianza funzionale:

```
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)
```



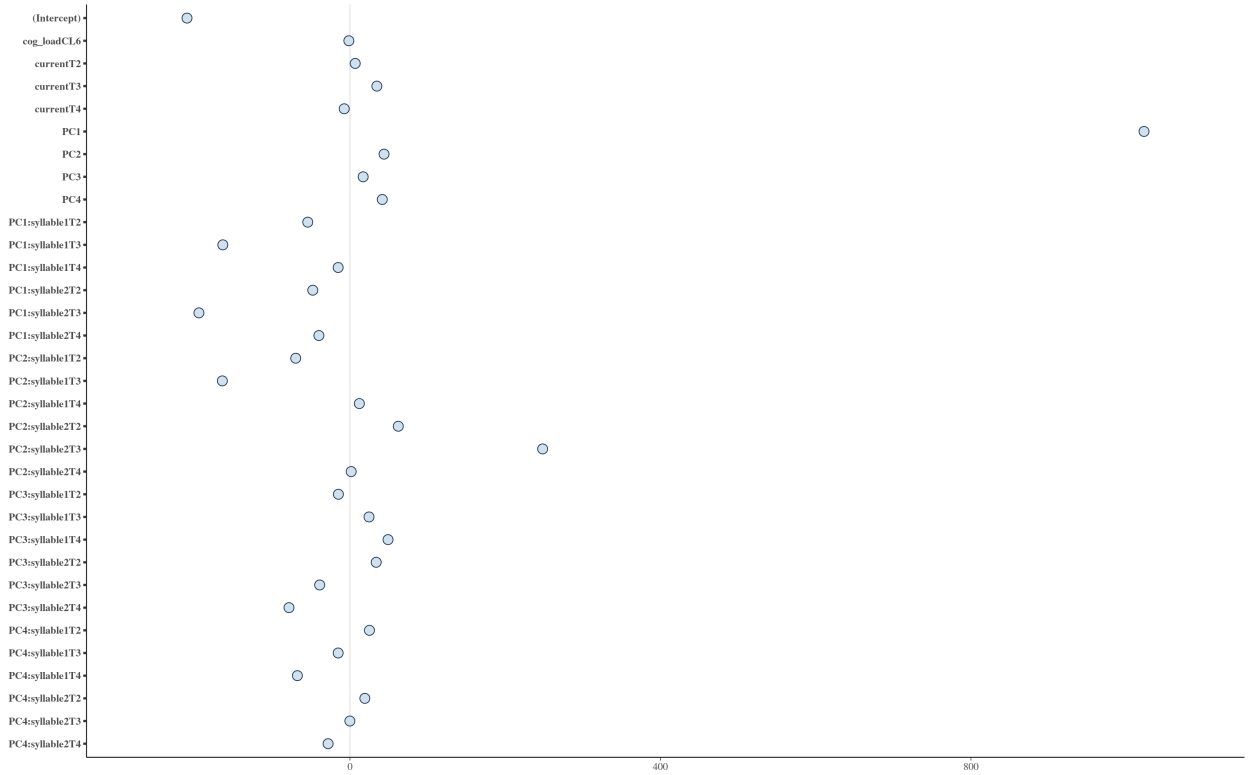

```r
# Aggiungo le autofunzioni come covariate al dataset
eigenColumns = NULL
for(i in 1:NROW(Y)){
    eigenColumns = rbind(eigenColumns, eigenFPCA)
}

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

```r
mandarino_fpca = mandarino_fpca[permute(1:NROW(mandarino_fpca)), ]
```

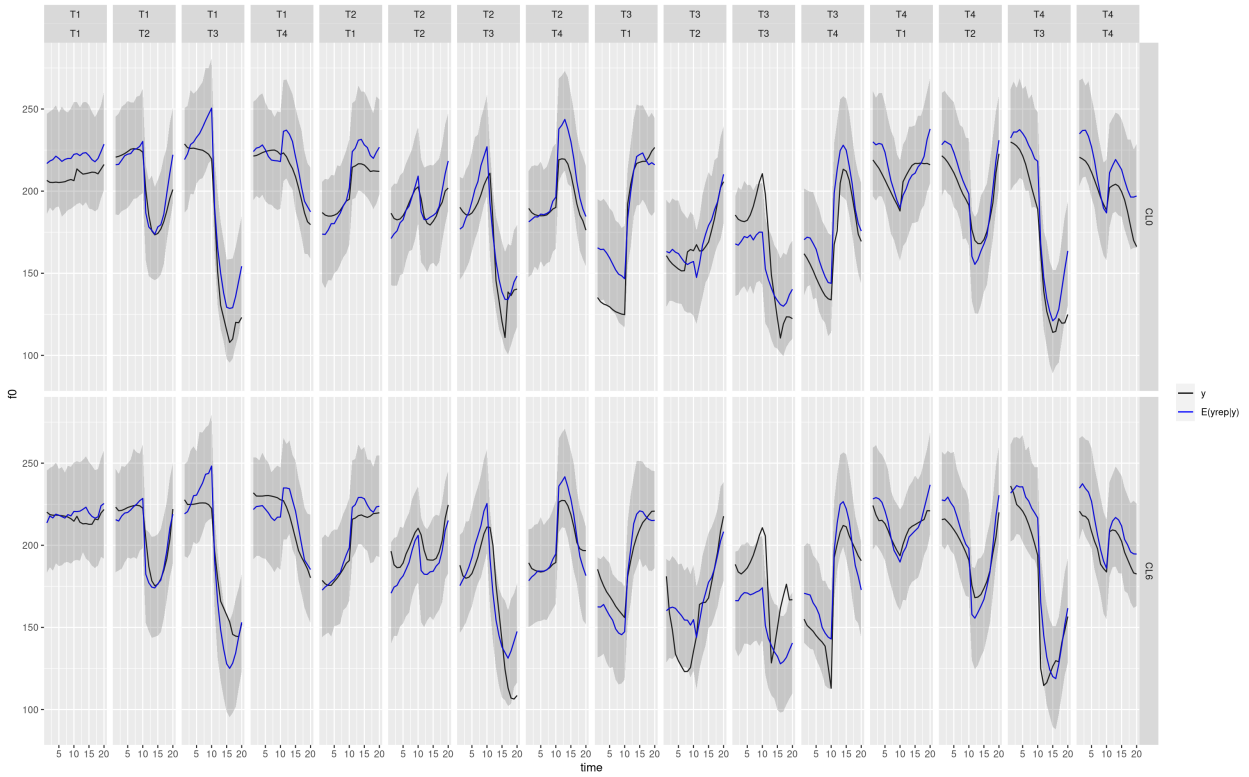
```r
if(file.exists("fitStanLmerFpca.Rdata")){
    load("fitStanLmerFpca.Rdata")
} else{
    fitStanLmerFpca = stan_glmer(f0 ~ cog_load + current +
                                 PC1 + PC2 + PC3 + PC4 +
                                 PC1:syllable1 + PC1:syllable2 +
                                 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",
                     iter = 20000,
                     QR = TRUE)
    save(fitStanLmerFpca, file="fitStanLmerFpca.Rdata")
}
# summary(fitStanLmerFpca)
```

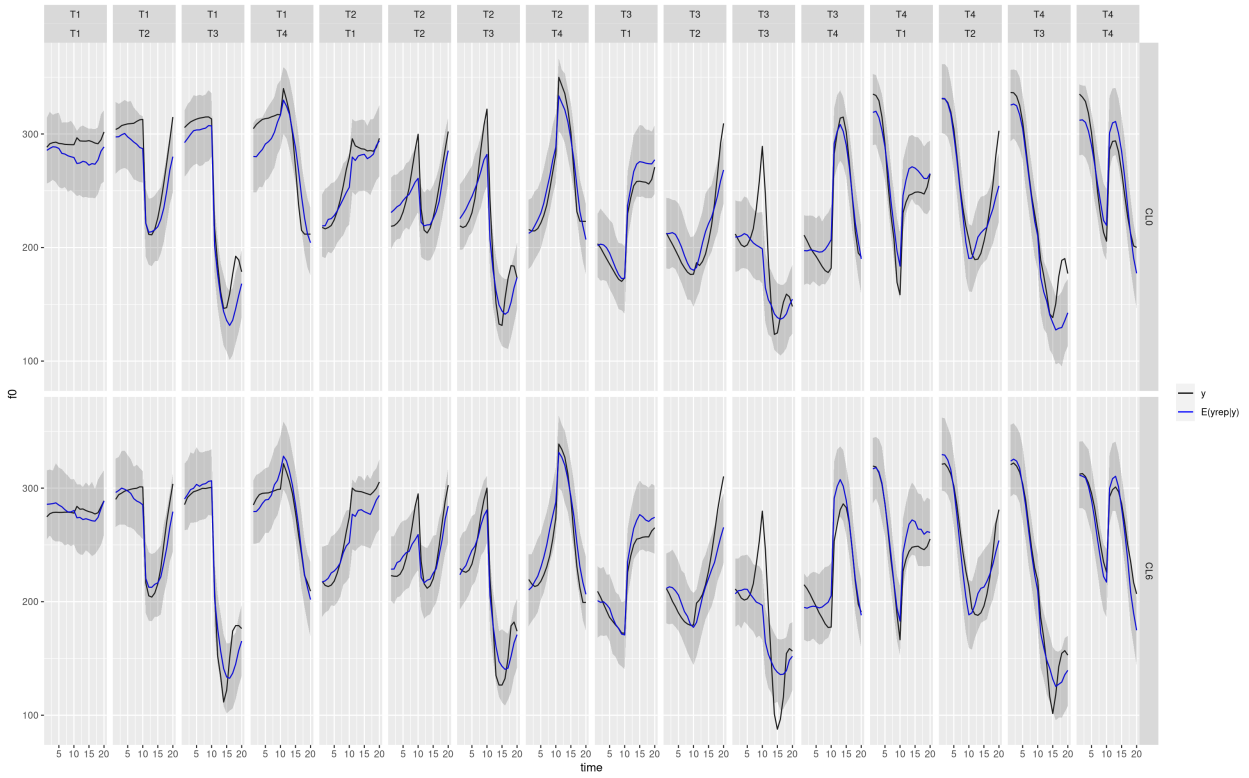
```
plot(fitStanLmerFpca, pars=names(fitStanLmerFpca$coefficients)[1:33])
```



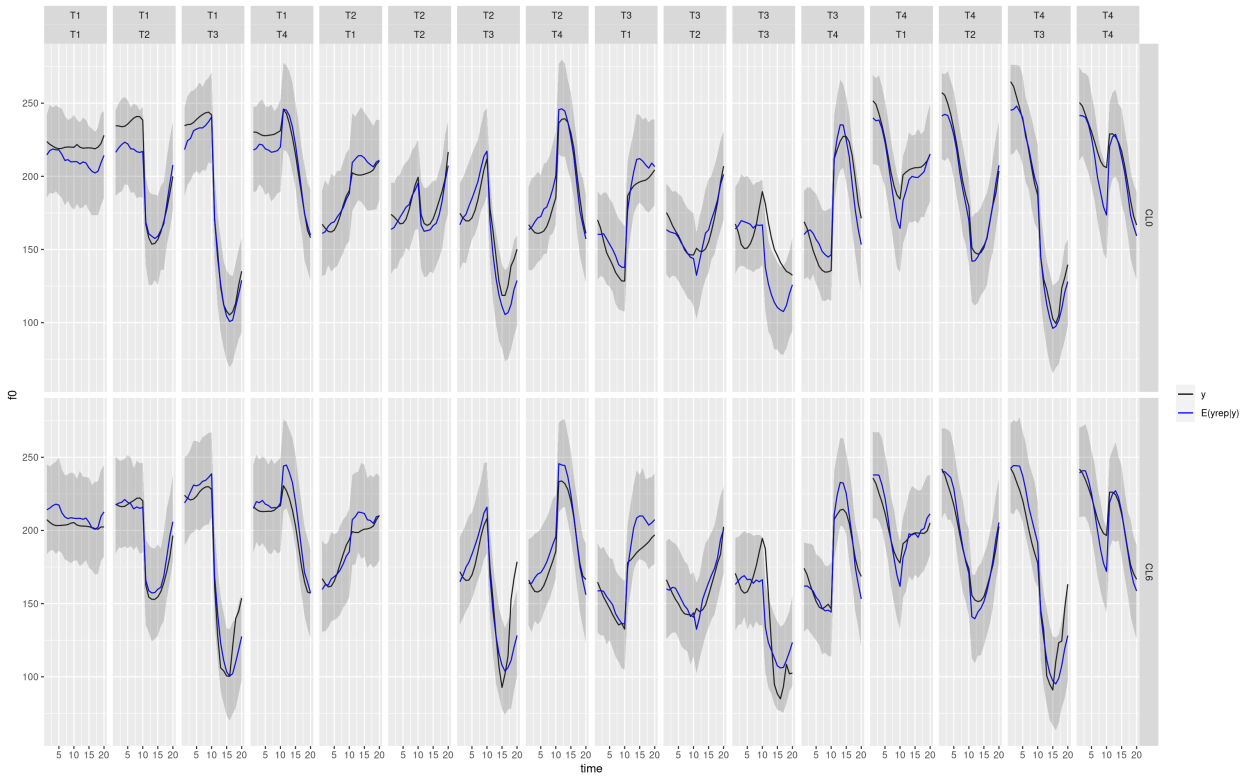

```
ypredStanLmerFpca = posterior_predict(fitStanLmerFpca, draws = 500)
ypred = apply(ypredStanLmerFpca, 2, mean)
stdev = apply(ypredStanLmerFpca, 2, sd)
lmin = apply(ypredStanLmerFpca, 2, function(col) quantile(col, p = 0.05))
lmax = apply(ypredStanLmerFpca, 2, function(col) quantile(col, p = 0.95))

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



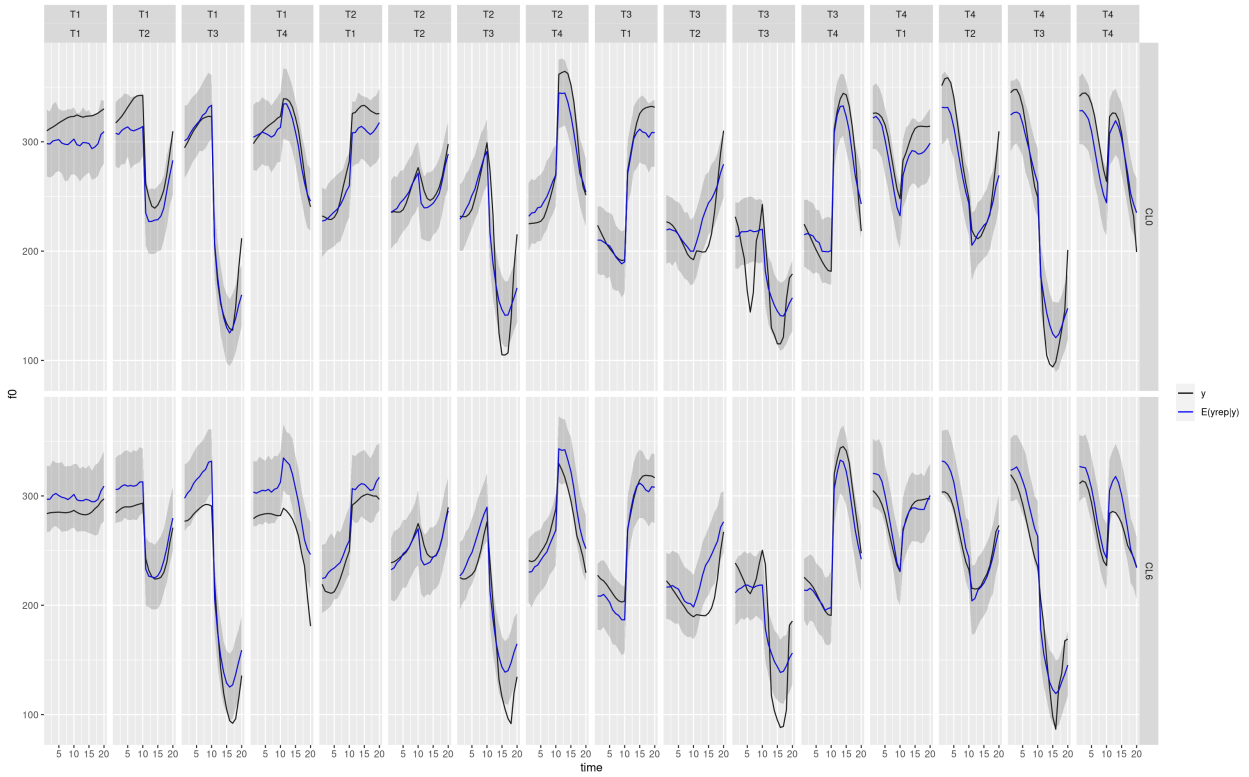

```
ppc_subject("S2", mandarino, ypred, lmin, lmax)
```



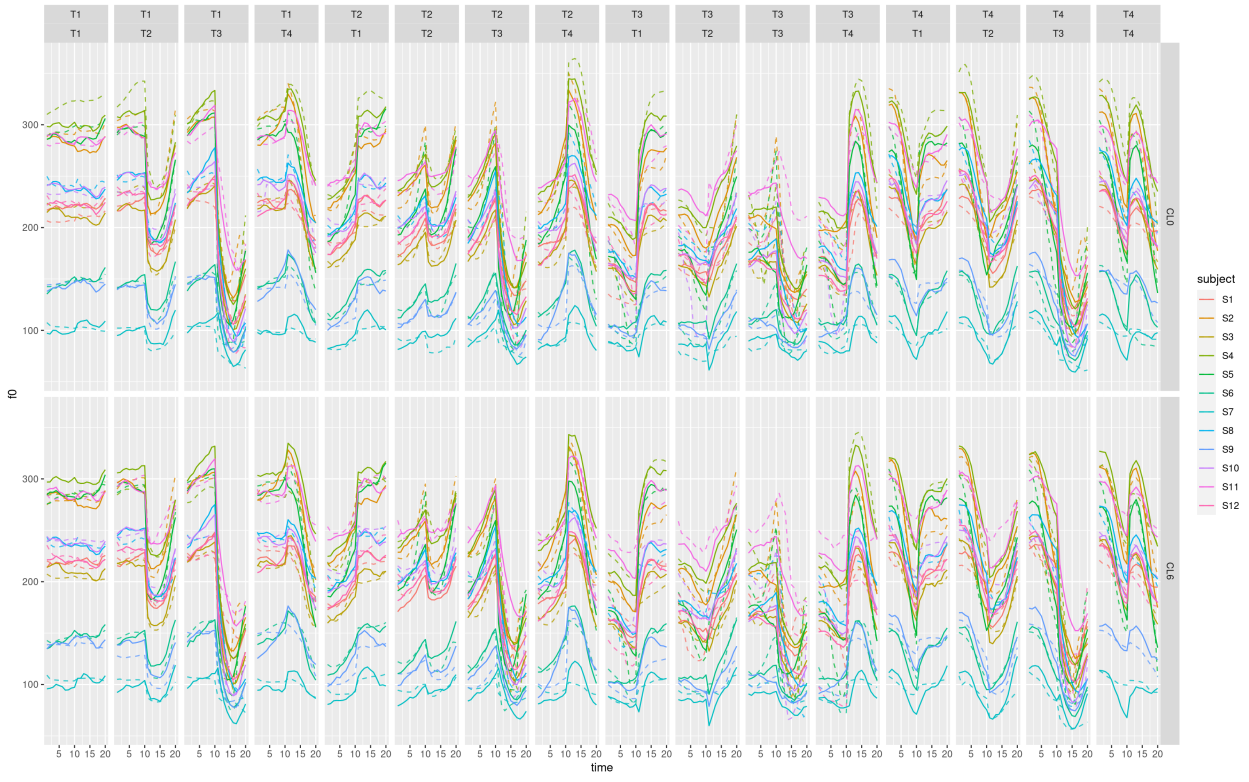
```
ppc_subject("S3", mandarino, ypred, lmin, lmax)
```




```
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)){
    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 = 3,
                        iter = 20000,
                        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 = 15)
        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]])
    }
    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("kf1.Rdata")){
    load("kf1.Rdata")
} else{
    kf1 = cv_stanvb(fitStanLmerFpca1, K = K, cores = kfcores)
    save(kf1, file = "kf1.Rdata")
}
print(cbind("mean" = mean(unlist(kf1)), "sd" = sd(unlist(kf1))))
```

```
##          mean       sd
## [1,] 626.0949 37.02347
```

```
if(file.exists("kf.Rdata")){
    load("kf.Rdata")
} else{
    kf = cv_stanvb(fitStanLmerFpca, K = K, cores = kfcores)
    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))))
```

```
##          mean       sd
## [1,] 405.8395 31.35269
```

```
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 = 20000,
                    QR = TRUE)
    save(fitStanLmerFpca2, file="fitStanLmerFpca2.Rdata")
}
```

```
if(file.exists("kf2.Rdata")){
    load("kf2.Rdata")
} else{
    kf2 = cv_stanvb(fitStanLmerFpca2, K = K, cores = kfcores)
    save(kf2, file = "kf2.Rdata")
}
print(cbind("mean" = mean(unlist(kf2)), "sd" = sd(unlist(kf2))))
```

```
##          mean       sd
## [1,] 397.1413 26.59181
```

```
if(file.exists("fitStanLmerFpca3.Rdata")){
    load("fitStanLmerFpca3.Rdata")
} else{
    fitStanLmerFpca3 = stan_glmer(f0 ~ 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,
                         algorithm = "meanfield",
                         iter = 20000,
                         QR = TRUE)
    save(fitStanLmerFpca3, file="fitStanLmerFpca3.Rdata")
}
```

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

```
##          mean       sd
## [1,] 390.1695 19.16239
```

```
rbind(cbind(mean(unlist(kf1)),  sd(unlist(kf1))),
      cbind(mean(unlist(kf)),  sd(unlist(kf))),
      cbind(mean(unlist(kf2)),  sd(unlist(kf2))),
```

```
    cbind(mean(unlist(kf3)),  sd(unlist(kf3)))
    ) %>%
  set_rownames(c("No eff. casuali di interazione", "Eff. casuali interazione", "Eff. casuali interazi
  set_colnames(c("mean", "sd"))
```

```
##                                                    mean       sd
## No eff. casuali di interazione                 626.0949 37.02347
## Eff. casuali interazione                       405.8395 31.35269
## Eff. casuali interazione no cog_load           397.1413 26.59181
## Eff. casuali interazione no cog_load no current 390.1695 19.16239
```