Web-based Supplementary Materials for Identifying Longitudinal Trends within EEG Experiments by Hasenstab, K., Sugar, C., Telesca, D., McEvoy, K., Jeste, S. and Şentürk, D.

Web Appendix C: R and SAS code

The following code is provided for implementing the proposed MAP-ERP procedure in R and the proposed weighted mixed effects model in SAS. The code includes the meta-preprocessing step and mixed effects modeling using the parametrization in the data analysis section of the manuscript. The sample dataset used for illustration of the meta-preprocessing step, "Raw_ERP.csv", is available online at http://dsenturk.bol.ucla.edu/supplements.html.

```
##############################################################################
#
# Description:
# Script for implementing MAP-ERP.
#
# Author: KH
# Hasenstab, K., Sugar, C., Telesca, D., McEvoy, K., Jeste, S. and
# Senturk, D. (2014) Identifying Longitudinal Trends within EEG
# Experiments.
#
# R 3.1.0
##############################################################################

##############################
##### Meta-Preprocessing #####
##############################

# The code is for meta-preprocessing of the data from a single subject,
# electrode and condition

##### Data Description
# Load or subset data as "dataset" for the ith subject, jth electrode
# and lth condition
# Dimension: (Size of set {K_i}) x (T + 6) where K_i is the set of
#  non-missing trials for subject i
# column 1: id: subject id
# column 2: group: group (ASD,TD)
# column 3: region: region
# column 4: electrode: electrode
# column 5: condition: condition (expected,unexpected)
# column 6: trial: trial index k prior to meta-preprocessing
```

```
# columns 7 through (T + 6): t: ERP time
dataset <- read.csv("Raw_ERP.csv", header=TRUE)
names(dataset) <- c("id", "group", "region", "electrode", "condition",
                    "trial", seq(-100, 896, 4))

##### Install relevant packages
install.packages("bisoreg") # For loess cross-validation
install.packages("splines") # For spline fits
install.packages("nlme") # For mixed effects modeling

##### Implement meta-preprocessing
T <- 250 # Number of observed ERP time points
# Dataframe containing a single raw ERP in each row
Xt <- dataset[,7:(T+6)]
# Set of observed trials for subject i
K_i <- dataset$trial
# window size of moving average
b <- 30

# Perform meta-preprocessing for a single subject, electrode and condition
# The meta-preprocessing function is given below
Y.tilde <- MetaPreprocessing(raw.erp = Xt, trial.set = K_i,
                             window.size = b,
                             trial.range = c(1, 120),
                             span.range = c(0.2, 1),
                             span.inc = 0.1, k.fold = 10,
                             find.peak = TRUE, find.dip = TRUE,
                             peak.range = c(190, 350),
                             dip.range = c(100, 250),
                             erp.domain = seq(-100, 896, 4),
                             verbose = TRUE)

# Combines the Y.tilde dataframe with the id, group, region, electrode,
#  and condition information
Y.tilde <- suppressWarnings(
  cbind(dataset[1, c("id", "group", "region", "electrode", "condition")],
        Y.tilde))


#########################################################################
##### R functions for meta-preprocessing

#########################################################################
# PeakDetection: MetaPreprocessing calls for this function
# ------------------------------------------------------------------ #
```

```
# DESCRIPTION : Identifies peak,dip magnitude and latency by taking a
#               cross-sectional average, implementing a loess smooth
#               and identifying maximums/minimums within feature
#               intervals defined in the literature.
# INPUT
# raw.erp: Dataframe with T columns containing entire single raw ERP
#          X_ijkl in each row for a given subject, electrode and
#          condition. T represents the ERP time dimension across times
#          t. Usually a subset used for meta-preprocessing.
# span.range: 1 x 2 numeric vector representing the lower and upper
#             bounds of the candidate spans for the loess smooth in
#             the peak detection algorithm to perform cross-validation.
#             Default is c(0.2, 1).
# span.inc: Increment of span.range for the candidate spans for the
#           loess smooth in the peak detection algorithm. Default is 0.1.
# k.fold: Number of folds for cross-validation of the loess smooth in
#         the peak detection algorithm. Default is 10.
# find.peak: If TRUE, identifies peaks and latency of those peaks
#            within a given interval of the ERP. Default is TRUE.
# find.dip: If TRUE, identifies dip and latency of those dips within a
#           given interval of the ERP. Default is TRUE.
# peak.range: 1 x 2 numeric vector representing the initial bounds for
#             identifying peaks during meta-preprocessing. Default is
#             c(190, 350) for the P3 peak.
# dip.range: 1 x 2 numeric vector representing the initial bounds for
#             identifying dips during meta-preprocessing. Default is
#             c(100, 250) for the N1 dip.
# erp.domain: 1 x T numeric vector of the observable ERP times points
#             t in function X_ijkl(t). Entries of erp.domain correspond
#             to the columns of raw.erp.
#
# OUTPUT
# A 1 x 2 numeric vector containing the peak OR dip amplitude
#  and latency or a 1 x 4 numeric vector containing the peak AND dip
#  amplitudes and latency
# ------------------------------------------------------------------- #
PeakDetection <- function(sub.erp, span.range = c(0.2, 1),
                          span.inc = 0.1, k.fold = 10,
                          find.peak = TRUE, find.dip = TRUE,
                          peak.range = c(190, 350),
                          dip.range = c(100, 250),
                          erp.domain) {

  # Peak or dip ranges must not lie on the boundary of the ERP time domain
  if ((peak.range[2] >= max(erp.domain)) |
```

```
      (peak.range[1] <= min(erp.domain)) |
      (dip.range[2] >= max(erp.domain)) |
      (dip.range[1] <= min(erp.domain))) {
  stop("Feature ranges must not contain endpoints of ERP time domain")
}

# Take cross-sectional mean of subset of X_ijkl(t)
mean.time  <- apply(sub.erp, 2, mean)

# Implement loess smoothing with cross-validation
loess.smooth <- loess.wrapper(erp.domain, mean.time,
                              span.vals=seq(span.range[1],
                                            span.range[2], span.inc),
                              folds=k.fold)
loess.smooth <- list(x=loess.smooth$x, y=loess.smooth$fitted)

#############################################################################

# Peak feature
if (find.peak) {

  # Indicator for boundary issues.  If peak estimate is located on
  # twice the size of the original feature boundary, the estimate
  # will be considered missing
  boundary <- 0

  # Interval for identifying peak
  peak.interval <- which(loess.smooth$x >= peak.range[1] &
                         loess.smooth$x <= peak.range[2])

  # Check if peak is on the boundary and iterate
  repeat {

    # Identify maximum within interval
    peak.ind <- which.max(loess.smooth$y[peak.interval])

    # Shift interval if maximum on boundary
    if (peak.ind == length(peak.interval)) {
      peak.interval <- peak.interval + 1
    } else if (peak.ind == 1) {
      peak.interval <- peak.interval - 1
    } else {
      break
    }  # end boundary if statement for peak
```

```
    # Quality check to see if estimate is far from original interval
    half.range <- (peak.range[2] - peak.range[1])/2
    if ((max(erp.domain[peak.interval]) > peak.range[2] + half.range) |
          (min(erp.domain[peak.interval]) < peak.range[1] - half.range) |
          (max(erp.domain[peak.interval]) >= max(erp.domain)) |
          (min(erp.domain[peak.interval]) <= min(erp.domain))) {
      boundary <- 1
      break
    }  # end if statement

  }  # end boundary repeat loop for peak


  if (boundary == 1) {

    peak.time <- NA
    peak <- NA

  } else {

    amp.ind <- peak.interval[peak.ind]
    peak.time <- loess.smooth$x[amp.ind] # peak latency
    peak <- mean(mean.time[amp.ind]) # peak amplitude

  } # end peak boundary if statement

}  # end peak feature

#########################################################################

# Dip feature
if (find.dip) {

  boundary <- 0

  # Interval for identifying dip
  dip.interval <- which(loess.smooth$x >= dip.range[1] &
                          loess.smooth$x <= dip.range[2])

  # Check if dip is on the boundary and iterate
  repeat {

    # Identify minimum within interval
    dip.ind <- which.min(loess.smooth$y[dip.interval])
```

```
    # Shift interval if maximum on boundary
    if (dip.ind == length(dip.interval)) {
      dip.interval <- dip.interval + 1
    } else if (dip.ind == 1) {
      dip.interval <- dip.interval - 1
    } else {
      break
    }  # end boundary if statement for dip

    # Quality check to see if estimate is far from original interval
    half.range <- (dip.range[2] - dip.range[1])/2
    if ((max(erp.domain[dip.interval]) > dip.range[2] + half.range) |
          (min(erp.domain[dip.interval]) < dip.range[1] - half.range) |
          (max(erp.domain[dip.interval]) >= max(erp.domain)) |
          (min(erp.domain[dip.interval]) <= min(erp.domain))) {
      boundary <- 1
      break
    }  # end if statement

  }  # end boundary repeat loop for dip

  # Identify time
  if (boundary == 1) {

    dip.time <- NA
    dip <- NA

  } else {

    amp.ind <- dip.interval[dip.ind]
    dip.time <- loess.smooth$x[amp.ind] # dip latency
    dip <- mean(mean.time[amp.ind]) # dip amplitude

  }  # end dip boundary if statement

} # end dip feature

# Output peak,dip magnitude and latency
if (find.peak & find.dip) {
  features <- c(peak, peak.time, dip, dip.time)
  names(features) <-  c("peak", "peak.time", "dip", "dip.time")
} else if (find.peak & !find.dip) {
  features <- c(peak, peak.time)
  names(features) <-  c("peak", "peak.time")
} else if (!find.peak & find.dip) {
```

```
    features <- c(dip, dip.time)
    names(features) <-  c("dip", "dip.time")
  }

  return(features)

}  # end function


##############################################################################
# MetaPreprocessing
# ------------------------------------------------------------------------ #
# DESCRIPTION : Performs meta-preprocessing on a single subject,
#               electrode and condition to identify peaks (or dips)
#               longitudinally among noisy ERP
# INPUT
# raw.erp: Dataframe of dimension (size of set {K_i} x T) containing
#          entire single raw ERP X_ijkl in each row for a given subject,
#          electrode and condition. K_i represents the set of
#          non-missing trials for the ith subject. T represents the ERP
#          time dimension across times t.
# trial.set: 1 x K_i numeric vector of observed trials whose entries
#            correspond to the rows in the raw.erp dataframe.
# window.size: Size of the window, b, in the moving average across trials.
# trial.range: 1 x 2 numeric vector representing the range of
#              observable trials, c(1, K), among all subjects
# span.range: 1 x 2 numeric vector representing the lower and upper
#             bounds of the candidate spans for the loess smooth in the
#             peak detection algorithm to perform cross-validation.
#             Default is c(0.2, 1).
# span.inc: Increment of span.range for the candidate spans for the
#           loess smooth in the peak detection algorithm. Default is 0.1.
# k.fold: Number of folds for cross-validation of the loess smooth in
#         the peak detection algorithm. Default is 10.
# find.peak: If TRUE, identifies peaks and latency of those peaks within
#            a given interval of the ERP. Default is TRUE.
# find.dip: If TRUE, identifies dip and latency of those dips within a
#           given interval of the ERP. Default is TRUE.
# peak.range: 1 x 2 numeric vector representing the initial bounds for
#             identifying peaks during meta-preprocessing. Default is
#             c(190, 350) for the P3 peak.
# dip.range: 1 x 2 numeric vector representing the initial bounds for
#            identifying dips during meta-preprocessing. Default is
#            c(100, 250) for the N1 dip.
# erp.domain: 1 x T numeric vector of the observable ERP times points t
```

```
#             in function X_ijkl(t). Entries of erp.domain correspond to
#             the columns of raw.erp.
# verbose: If TRUE, prints the trial level iterations of
#          meta-preprocessing. Default is FALSE.
#
# OUTPUT
#   A dataframe containing (size of set {M_i}) rows and the following
#    variables as columns
#   trial: Trial after meta-preprocessing
#   n.avg: Number of curves in the cross-sectional average (c_ijkl)
#   peak: peak amplitude (displayed if find.peak=TRUE)
#   peak.time: peak Latency (displayed if find.peak=TRUE)
#   dip: dip amplitude (displayed if find.dip=TRUE)
#   dip.time: dip Latency (displayed if find.dip=TRUE)
# ------------------------------------------------------------------- #
MetaPreprocessing <- function(raw.erp, trial.set, window.size,
                              trial.range,
                              span.range = c(0.2, 1),
                              span.inc = 0.1,
                              k.fold = 10, find.peak = TRUE,
                              find.dip = TRUE,
                              peak.range = c(190, 350),
                              dip.range = c(100, 250),
                              erp.domain, verbose = FALSE) {

  # Call package for loess cross validation
  require(bisoreg)

  # Store moving average window
  trial.window <- c(1, window.size)
  trial.window.initial <- trial.window

  # Shift trial.set and trial.range so that trial.range starts at 1
  trial.shift <- trial.range[1] - 1
  trial.set <- trial.set - trial.shift
  trial.range <- trial.range - trial.shift

  # Output dataframe
  if (find.peak & find.dip) {
    features <- data.frame(trial=numeric(0), n.avg=numeric(0),
                           peak=numeric(0), peak.time=numeric(0),
                           dip=numeric(0), dip.time=numeric(0))
  } else if (find.peak & !find.dip) {
    features <- data.frame(trial=numeric(0), n.avg=numeric(0),
                           peak=numeric(0), peak.time=numeric(0))
```

```
} else if (!find.peak & find.dip) {
  features <- data.frame(trial=numeric(0), n.avg=numeric(0),
                         dip=numeric(0), dip.time=numeric(0))
}

# Loop over trials
for (window.ind in 1:trial.range[2]) {

  # Subset trials within trial window based on B_k partitions
  if (window.ind < (trial.window.initial[2]/2)) {

    # First set of B_k
    endpoint1 <- window.ind - (window.ind - 1)
    endpoint2 <- window.ind + (window.ind - 1)
    sub.erp   <- raw.erp[trial.set %in% (endpoint1:endpoint2), ]

  } else if (window.ind >= (trial.window.initial[2]/2) &
              window.ind <= (trial.range[2] -
                               trial.window.initial[2]/2)) {

    # Second set of B_k
    sub.erp <- raw.erp[trial.set %in%
                         (trial.window[1]:trial.window[2]), ]
    trial.window <- trial.window + 1

  } else if (window.ind > (trial.range[2] -
                             trial.window.initial[2]/2)) {

    # Third set of B_k
    endpoint1 <- window.ind - (trial.range[2] - window.ind)
    endpoint2 <- window.ind + (trial.range[2] - window.ind)
    sub.erp <- raw.erp[trial.set %in% (endpoint1:endpoint2), ]

  }  # end if statement

  # Identify amplitude and latency via peak detection function
  if (nrow(sub.erp) > 0) {
    feature <- c(window.ind, nrow(sub.erp),
                 PeakDetection(sub.erp, span.range, span.inc, k.fold,
                               find.peak, find.dip, peak.range,
                               dip.range, erp.domain))
    names(feature)[1:2] <- c("trial", "n.avg")
    features[nrow(features) + 1, ] <- feature
  } # end peak detection if statement
```

```
    if (verbose) {
      print(paste(window.ind, " out of ",
                  trial.range[2], " trials", sep=""))
    } # end verbose

  }  # end window.ind loop

  # Shift trials back to original domain
  features$trial <- features$trial + trial.shift

  # Output peak, dip magnitude and latency
  return(features)

}  # end function
##############################################################################


##############################################################################
##### Fitting of the weighted mixed effects model of peaks in R #####
##############################################################################

##### The mixed model function will read the meta-preprocessed data
#####  from all subjects
# Implement meta-preprocessing step on each subject, electrode
#  and condition and store as "dataset.model"
# You may need to loop over the above code
# The dataframe "dataset.model" has 11 columns
# column 1: id: subject id
# column 2: group: group (ASD,TD)
# column 3: region: region
# column 4: electrode: electrode
# column 5: condition: condition (expected,unexpected)
# column 6: trial: trial index after meta-preprocessing
# column 7: n.avg: c_ijkl values
# column 8: peak: estimated peak value Y.tilde (e.g., P3)
# column 9: peak.time: peak latency
# column 10: dip: estimated dip value Y.tilde (e.g., N1)
# column 11: dip.time: dip latency
dataset.model <- read.csv("MetaPreprocessed_Data.csv", header=TRUE)

# Load relevant packages
library(splines) # splines package
library(nlme) # nlme package for mixed effects modeling

# Change levels of factor variable (for reference group)
```

```
dataset.model$group <- factor(dataset.model$group,
                              levels(dataset.model$group)[c(2, 1)])
dataset.model$condition <- factor(dataset.model$condition,
                                  levels(dataset.model$condition)[c(2, 1)])

# Exclude missing values from "dataset.model"
dataset.model <- subset(dataset.model, !is.na(peak))

# Calculate spline bases and merge to meta-preprocessed data
spline <- as.data.frame(cbind(1:120, ns(1:120, df=5)))
names(spline) <- c("trial", paste("spline", 1:5, sep=""))
dataset.model <- merge(dataset.model, spline, by="trial")

# Partition c_ijkl values to correct for heteroskedasticity
dataset.model$n.factor <- cut(dataset.model$n.avg,
                              breaks=c(1, 6, 11, 16, 21, 26, 31),
                              right=FALSE)

# Fixed Effects
f <- peak ~ spline1 + spline2 + spline3 + spline4 + spline5 +
  spline1*group + spline2*group + spline3*group +
  spline4*group + spline5*group +
  spline1*condition + spline2*condition + spline3*condition +
  spline4*condition + spline5*condition +
  group * condition +
  spline1*group*condition + spline2*group*condition +
  spline3*group*condition +
  spline4*group*condition + spline5*group*condition

# Random effects
r <- list(id=pdDiag(~spline1 + spline2 + spline3 + spline4 + spline5),
          region=pdDiag(~spline1 + spline2 + spline3 + spline4 + spline5))

# Fit mixed effects model on the peaks
fit <- lme(f, data=dataset.model, r, weights=varIdent(form=~1|n.factor),
           control=list(maxIter=1000, msMaxIter=1000, msVerbose=TRUE),
           keep.data=FALSE)
```

```
/*************************************************************************/
/***** Fitting of the weighted mixed effects model of peaks in SAS *****/
/*************************************************************************/

/*************************************************************************/
/*
* Description:
* Script for fitting weighted mixed effects model.
*
* Author: KH
* Hasenstab, K., Sugar, C., Telesca, D., McEvoy, K., Jeste, S. and
* Senturk, D. (2014) Identifying Longitudinal Trends within EEG
* Experiments.
*
* SAS v9.3
*/
/*************************************************************************/

***** Data Description;
* Load output from meta-preprocessing step performed in R as
*  "dataset_model";
* Note: SAS missingness is denoted by ".", not "NA"
* column 1: id: subject id;
* column 2: group: group (ASD,TD);
* column 3: region: region;
* column 4: electrode: electrode;
* column 5: condition: condition (expected,unexpected);
* column 6: trial: trial index after meta-preprocessing;
* column 7: n_avg: c_ijkl values;
* column 8: n_factor: s partition of c_ijkl values;
* column 9: peak: estimated peak value Y.tilde (e.g., P3);
* column 10: peak_time: peak latency;
* column 11: dip: estimated dip value Y.tilde (e.g., N1);
* column 12: dip_time: dip latency;
* columns 13 through 17: spline1-spline5: splines;
* splines can be transported from R;

***** Weighted mixed effects modeling;
proc mixed method=reml lognote cl covtest data=dataset_model;
     class id group region electrode condition trial n_factor;
     * Fixed effects;
     model peak = spline1-spline5 group condition
                  spline1*group spline2*group spline3*group
                  spline4*group spline5*group
                  spline1*condition spline2*condition spline3*condition
```

```
                spline4*condition spline5*condition
                group*condition
                spline1*group*condition spline2*group*condition
                spline3*group*condition spline4*group*condition
                spline5*group*condition
                / s cl residual;
    * Random effects;
     random intercept spline1-spline5/ type=simple subject=id;
     random intercept spline1-spline5/ type=simple subject=region(id);
     * Weighting;
     repeated trial / type=SIMPLE group=n_factor
                subject=electrode(region condition id);
run;
```