Statistical Analysis Plan (SAP) - Protocol #015078

David Moreau

8/19/2015

The present document accompanies the document "Protocol_#015078", which details the planned implementation of the study "High-Intensity Training to Enhance Cognition: A randomized, controlled, 6-week intervention in children" by Moreau et al.

Oveview

Statistical analyses will be performed in R (R Core Team, 2015). We will report all packages used in our analyses. R code will be shared on GitHub. The repository includes data sets, R scripts, details and script of the HIT workout, the CONSORT flow diagram and the CONSORT checklist. We will report descriptive statistics for both groups, on age, gender, sample size, handedness, BMO, previous clinical diagnosis, previous remediation, experience with videogaming, physical exercise, and self-reported health, sleep quality and happiness. We will report Bayesian model comparisons, to allow quantifying the degree of evidence for a given model compared to other models tested, combined with Bayesian parameter estimations when relevant. We will also report the equivalent frequentist analyses for transparency, and in order to facilitate reading. We will examine normality of distribution for all continuous variables. If distributions is skewed, leptokurtic or platykurtic, we will compare results using non-corrected vs. log-transformed data, and will look for discrepancies. Note that the analyses we will present are extremely robust to outliers, as priors can be adapted to reflect deviations from normality. Regardless, we will check consistency using standard approaches to outlier exclusion, to facilitate direct comparisons with frequentist tests. Outliers will be defined as values more than 3/2 times the upper quartile or less than 3/2 times the lower quartile of a given distribution, and systematically checked consistency of our results with and without inclusion.

Data cleaning

```
### PREPING ENVIRONMENT ###

# Clear cache
rm(list = ls())

# Set working directory
setwd("")

# Colors
oldpalette <- c("#7B8D78",##darkgreen
                "#FF8F00",##orange
                "#843418",##darkred
                "#6959CD",##darkpurple
                "#4F94CD",##lightblue
                "#CDAD00")##gold
palette <- c("#58D9BF", "#46ABE3", "#4485C5", "#3276BE", "#2A5766") #(from lighter to darker)

# Load packages (+ install if required)
packages <- c("car", "BayesFactor", "LaplacesDemon", "ggplot2", "gridExtra", "dplyr",
              "statcheck",
              "BEST", "rjags", "lsr", "psych",
```

```r
               "scales", "sm")
new.packages <- packages[!(packages %in% installed.packages()[,"Package"])]
if(length(new.packages)) install.packages(new.packages)
sapply(packages, suppressPackageStartupMessages(require), warn.conflicts=F, quietly=T, character.only=T)

# Set default ggplot parameters
default <-  theme_bw() +
  theme(axis.line.x = element_line(colour = "black", size=.5, linetype = 1),
        axis.line.y = element_line(colour = "black", size=.5, linetype = 1),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.border = element_blank(),
        panel.background = element_blank())

# Source functions
source("summarySE.R") #for summary stats and plots
source("summarySEwithin.R") #for summary stats and plots

# Load data
raw <- read.csv("raw.csv", header = T, stringsAsFactors=FALSE)
str(raw) #data frame has N columns and N rows
glimpse(raw)
colnames(raw)

# Table form
raw <- tbl_df(raw)


### DATA CLEANING ###

# Some variables are factors (Gender, Time, Condition)
raw$Gender <- as.factor(raw$Gender)
raw$Time <- as.factor(raw$Time)
raw$Condition <- as.factor(raw$Condition)

# Center and scale cognitive measures
cog <- raw %>% select(A1:F) %>%
  #log() %>%
  #scale(center=T, scale=T) %>%
  data.frame() %>%
  mutate(A = ((A2 + A3) / 2 * (1/A1))) %>%
  #scales:::rescale(hit$A, to = c(0, 100))
  #mutate(C = ((C2 + C3) / 2 * (1/C1))) %>%
  mutate(C = C3 * (1/C1)) %>%
  mutate(D = (D2 * (1/D1))) %>%
  mutate(E = (E2 * (1/E1)))

cog$A <- scale(cog$A) #convert to z-scores
cog$B <- scale(cog$B)
cog$C <- scale(cog$C)
cog$D <- scale(cog$D)
cog$E <- scale(cog$E)
cog$F <- scale(cog$F)
```

```r
cog <- cog %>%
  mutate(ATT = (A + B + C) / 3) %>%
  mutate(WM = (D + E + F) / 3)

#cog$ATT <- scale(cog$ATT)
#cog$WM <- scale(cog$WM)

# Identify deviations from normality
#log()

# Separate non-cog measures from rest
info <- raw %>% #non-cog
  select(ID:Time) %>%
  select(-c(First_name, Last_name, School)) %>%
  mutate(Target = HR_rest + .80*((220 - Age) - HR_rest)) #add HR target variable

#info$BDNF <- ifelse(info$BDNF=="GG", "val", "met")
info$BDNFvalmet[info$BDNF=="G"] <- "val"
info$BDNFvalmet[info$BDNF=="GA"] <- "met"
info$BDNFvalmet[info$BDNF=="A"] <- "met"

# Whole data set
hit <- data.frame(info, cog)
write.csv(hit, file="hit.csv", row.names=F)

### Create additional data set in wide format ###
library(reshape2)
library(data.table) #for the setDT function below
hit.wide <- dcast(setDT(hit), ID + Condition ~ Time,
                  value.var = c("Age", "HR_rest", "BDNF", "COMT", "BDNFvalmet",
                                                                     "A", "B", "C", "D", "E", "F",
                                                                     "ATT", "WM"))
#hit.wide$ATT_1 <- scale(hit.wide$ATT_1)
#hit.wide$ATT_2 <- scale(hit.wide$ATT_2)

#hit.wide$WM_1 <- scale(hit.wide$WM_1)
#hit.wide$WM_2 <- scale(hit.wide$WM_2)


hit.wide <- hit.wide %>%
  mutate(ATT_gain = ATT_2 - ATT_1) %>%
  mutate(WM_gain = WM_2 - WM_1) %>%
  mutate(HR_gain = HR_rest_2 - HR_rest_1)

#hit.wide$WM_gain <- scale(hit.wide$WM_gain)
#hit.wide$ATT_gain <- scale(hit.wide$ATT_gain)

write.csv(hit.wide, file="hitwide.csv", row.names=F)

### Create additional data set with hit condition only ###
hit.only <- hit.wide %>%
  filter(Condition==1)
```

```r
write.csv(hit.only, file="hitonly.csv", row.names=F)

### Create additional data set with control condition only ###
hit.only <- hit.wide %>%
  filter(Condition==0)

write.csv(hit.only, file="controlonly.csv", row.names=F)

### Create data set with genotyped individuals only
genes <- hit.wide[complete.cases(hit.wide$BDNFvalmet_1),]
write.csv(genes, file="genes.csv", row.names=F)

### EFA data
efa.data <- hit %>%
  filter(Time == 1) %>%
  select(A, B, C, D, E, F) %>% #select only one measure / task
  na.omit()

write.csv(efa.data, file="efa.csv", row.names=F)

# Without outliers for CC and WMC at times 1 and 2
no.out <- hit.wide %>%
  filter(-c(ID))


### Create additional data set for open access ###
moreau_hit_2017 <- hit %>%
  select(-c(ID, COMT, BDNF, BDNFvalmet, Fitbit)) %>%
  write.csv(file= "moreau_hit_2017.csv")
```

Physiological data

Physiological improvements will provide corroborating evidence for the hypothesized changes associated with exercise, and they will allow identifying idiosyncratic parameters often characteristic of training interventions. We will conduct an (Bayesian) ANCOVA on change in resting heart rate, with Condition (HIT vs. Control) as a fixed factor and baseline heart rate as a covariate. We will report the main group effect, and test for homogeneity of variance (Levene's test). This will be complemented by a Student/Welch (Bayesian) two-sample t-test on resting heart rate change, by Condition (HIT vs. Control), after a split on resting heart rate at pretest. We will use (Bayesian) linear regression (after checking assumptions) to model effort and workout load across training sessions.

```r
setwd("")
source('HIT_cleanup.R')

# Get Fitbit data
#setwd("/Users/davidmoreau/Google Drive/MovinCog/Data/Fitbit")
#day.act <- read.csv("dailyActivity.csv")
#day.cal <- read.csv("dailyCalories.csv")
#day.int <- read.csv("dailyIntensities.csv")
#day.ste <- read.csv("dailySteps.csv")
#hr <- read.csv("heartrate_seconds.csv")

# Compare HR_gains between groups (hit, control)
```

```r
hr <- hit %>% #extract info needed from hit file
  select(ID, Condition, Time, HR_rest, Target)

hrgains <- reshape(hr, timevar="Time", idvar = c("ID", "Condition"), direction="wide") #reshape to wide

hrgains <- hrgains %>% #add gain pre to post (HR)
  mutate(Gain = HR_rest.2 - HR_rest.1)

# Test-retest reliability (assessed on controls at pretest and posttest)
controls <- hit.wide %>%
  filter(Condition==0)
cor(controls$HR_rest_1, controls$HR_rest_2, use="complete.obs")


# PLOTS
plotA <- ggplot(hrgains, aes(Condition, Gain, color=Condition)) +
  geom_violin(trim=F, size=1) +
  geom_boxplot(width=0.1) +
  stat_summary(fun.y=mean, geom="point", shape=16, size=2) + #add means
  geom_hline(aes(yintercept = 0), linetype="dashed", color="gray", size=.7) +
  scale_color_manual(values=c("gray30", "#FF8F00")) +
  scale_x_discrete(breaks=c("0", "1"),
                   labels=c("Control", "HIT")) +
  xlab("Group") + ylab("HR rest Change") +
  ylim(-30, 30) +
  ggtitle("A") +
  theme_bw() +
  theme(legend.position="none") +
  theme(plot.title=element_text(family="Arial", face="bold", size=20)) +
  theme(axis.line.x = element_line(colour = "black", size=.5, linetype = 1),
        axis.line.y = element_line(colour = "black", size=.5, linetype = 1),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.border = element_blank(),
        panel.background = element_blank())



# PLOT FIGURE 1 B ##########################################################

#day.ste[day.ste == 0] <- NA #mark missing values as NA

dataB <- read.csv("dataB.csv", header=T)
dataB$Time <- as.factor(dataB$Time) #needed for values to be treated discretely

mean(dataB$hrdev)
sd(dataB$hrdev)

plotB <- ggplot(dataB, aes(ID2, hrratio, color=Time)) +
  geom_point() +
  geom_hline(aes(yintercept = 1), linetype="dashed", color="#4F94CD", size=1) +
  scale_color_manual(values=c("gray32", "gray"),
                     name="Time",
                     breaks=c("1", "2"),
                     labels=c("Pretest", "Posttest")) +
  xlab("Participant") + ylab("Accuracy Ratio") +
  ggtitle("B") +
  theme_bw() +
  theme(plot.title=element_text(family="Arial", face="bold", size=20)) +
  theme(axis.line.x = element_line(colour = "black", size=.5, linetype = 1),
        axis.line.y = element_line(colour = "black", size=.5, linetype = 1),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.border = element_blank(),
        panel.background = element_blank())

#plotB <- ggplot(complete, aes(Target, hrmax)) +
  geom_point(color="gray33") +
  geom_abline(color="#4F94CD", intercept = coef(lm(hrmax ~ Target, data=complete))[1]) +
  geom_smooth(color="darkolivegreen3") +
  xlab("Target HR") + ylab("Measured HR") +
  ggtitle("B") +
  theme_bw() +
  theme(plot.title=element_text(family="Arial", face="bold", size=20)) +
  theme(axis.line.x = element_line(colour = "black", size=.5, linetype = 1),
        axis.line.y = element_line(colour = "black", size=.5, linetype = 1),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.border = element_blank(),
        panel.background = element_blank()) +
  theme(legend.position="none")


# PLOT FIGURE 1 C ##############################################################

dataC <- read.csv("dataC.csv", header=T)
summary(lm(dataC$hrmax ~ dataC$Sessions))

#plotC <- ggplot(dataC, aes(Sessions, hrmax_corrected)) +
  geom_line() +
  geom_smooth(color="darkolivegreen3") +
  xlab("Session") + ylab("HR max (avg)") +
  ggtitle("C") +
  theme_bw() +
  theme(legend.position="none") +
  theme(plot.title=element_text(family="Arial", face="bold", size=20)) +
  theme(axis.line.x = element_line(colour = "black", size=.5, linetype = 1),
        axis.line.y = element_line(colour = "black", size=.5, linetype = 1),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.border = element_blank(),
        panel.background = element_blank())

plotC <- ggplot(dataC, aes(Sessions, hrmax)) +
  geom_line() +
  geom_smooth(color="darkolivegreen3") +
  xlab("Session") + ylab("HR max (avg)") +
  ggtitle("C") +
  theme_bw() +
  theme(legend.position="none") +
  theme(plot.title=element_text(family="Arial", face="bold", size=20)) +
  theme(axis.line.x = element_line(colour = "black", size=.5, linetype = 1),
        axis.line.y = element_line(colour = "black", size=.5, linetype = 1),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.border = element_blank(),
        panel.background = element_blank())

# PLOT FIGURE 1D ##########################################################

dataD <- read.csv("dataD.csv", header=T)
summary(lm(dataD$Steps ~ dataD$Day))

# Effort (steps) as a function of Sessions
effort <- dataD %>%
  group_by(Day) %>%
  summarise_each(funs(sum)) %>%
  mutate(Steps_per_Day = Steps/30)

plotD <- ggplot(effort, aes(Day, Steps_per_Day))+
  geom_line() +
  geom_smooth(color="red3") +
  xlab("Session") + ylab("Steps (avg)") +
  ggtitle("D") +
  theme_bw() +
  theme(legend.position="none") +
  theme(plot.title=element_text(family="Arial", face="bold", size=20)) +
  theme(axis.line.x = element_line(colour = "black", size=.5, linetype = 1),
        axis.line.y = element_line(colour = "black", size=.5, linetype = 1),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.border = element_blank(),
        panel.background = element_blank())



# ARRANGE PLOTS ##########################################################

allplots <- grid.arrange(plotA, plotB, plotC, plotD, ncol=2); allplots
```

EFA

An exploratory factor analysis using principal component extraction and promax rotation will be performed on all six cognitive measures at pretest (Flanker – Go/no-go – Stroop – Backward digit span – Backward Corsi blocks – Visual 2-back). Promax allows factors to correlate; this property is especially appropriate when the factors extracted are assumed to be correlated to some degree. This is to be expected in the proposed design. We will inspect the corresponding scree plot and eigenvalues to determine the number of components, and will report factor loadings for all constructs. We will confirm the structure of the model with the corresponding chi-squared value and Bayesian Information Criterion. The factors extracted from the EFA will be used for subsequent group comparisons at the construct level.

```r
setwd("")
source('HIT_cleanup.R')

### EFA ###
#efa.data <- scales:::rescale(efa.data, to = c(0, 100))
#efa.data <- log(efa.data)
cor(na.omit(efa.data))
fit <- factanal(na.omit(efa.data), 2,
                rotation = "promax", scores="regression") #the model
print(fit, digits=2, cutoff=.1, sort=TRUE)
nrow(na.omit(efa.data)) # N for efa


#The chi-square statistic and p-value in factanal are testing the hypothesis that the model fits the da
#When the p value is low we can reject this hypothesis - so in this case, the 2-factor model does not f
#Uniquenesses are the variance in each item that is not explained by the two factors.
fit2 <- fa.promax(efa.data, factors=2, digits=2, sort=T)
load <- fit$loadings[,1:2]
plot(load,type="l",main="Screeplot") # set up plot
text(load,labels=names(mydata),cex=.7) # add variable names


### PCA ###
# run PCA analysis to confirm the two-factor model
pca.data <- hit %>%
  select(A, B, C, D, E, F) #select only one measure / task
pca <- prcomp(na.omit(pca.data))
print(pca)
plot(pca, type="l", main="Screeplot") #plot variances
sumr <- summary(pca); sumr
plot(sumr[3,], type="l") #plot cummulative proportions
```

MCMC

Where appropriate, we will provide details or relevant references about the MCMC algorithm we use. MCMC will be used to generate posterior samples via the Metropolis-Hastings algorithm (see for details Rubinstein & Kroese, 2011). All analyses will be set at 10,000 iterations, with diagnostic checks for convergence. One chain per analysis will be used for all analyses reported in the paper, with a thinning interval of 1 (i.e., no iteration to be discarded).

```r
## source script


# Load rjags. Assumes JAGS is already installed.
try( library(rjags) )
# Load runjags. Assumes JAGS is already installed.
try( library(runjags) )
try( runjags.options( inits.warning=FALSE , rng.warning=FALSE ) )

# set default number of chains and parallelness for MCMC:
library(parallel) # for detectCores().
```

```
nCores = detectCores()
if ( !is.finite(nCores) ) { nCores = 1 }
if ( nCores > 4 ) {
  nChainsDefault = 4  # because JAGS has only 4 rng's.
  runjagsMethodDefault = "parallel"
}
if ( nCores == 4 ) {
  nChainsDefault = 3  # save 1 core for other processes.
  runjagsMethodDefault = "parallel"
}
if ( nCores < 4 ) {
  nChainsDefault = 3
  runjagsMethodDefault = "rjags" # NOT parallel
}

#------------------------------------------------------------------------------
# Functions for opening and saving graphics that operate the same for
# Windows and Macintosh and Linux operating systems. At least, that's the hope!

openGraph = function( width=7 , height=7 , mag=1.0 , ... ) {
  if ( .Platform$OS.type != "windows" ) { # Mac OS, Linux
    tryInfo = try( X11( width=width*mag , height=height*mag , type="cairo" ,
                        ... ) )
    if ( class(tryInfo)=="try-error" ) {
      lineInput = readline("WARNING: Previous graphics windows will be closed because of too many open
      graphics.off()
      X11( width=width*mag , height=height*mag , type="cairo" , ... )
    }
  } else { # Windows OS
    tryInfo = try( windows( width=width*mag , height=height*mag , ... ) )
    if ( class(tryInfo)=="try-error" ) {
      lineInput = readline("WARNING: Previous graphics windows will be closed because of too many open
      graphics.off()
      windows( width=width*mag , height=height*mag , ... )
    }
  }
}

saveGraph = function( file="saveGraphOutput" , type="pdf" , ... ) {
  if ( .Platform$OS.type != "windows" ) { # Mac OS, Linux
    if ( any( type == c("png","jpeg","jpg","tiff","bmp")) ) {
      sptype = type
      if ( type == "jpg" ) { sptype = "jpeg" }
      savePlot( file=paste0(file,".",type) , type=sptype , ... )
    }
    if ( type == "pdf" ) {
      dev.copy2pdf(file=paste0(file,".",type) , ... )
    }
    if ( type == "eps" ) {
      dev.copy2eps(file=paste0(file,".",type) , ... )
    }
  } else { # Windows OS
    file=paste0(file,".",type)
    savePlot( file=file , type=type , ... )
  }
}

#------------------------------------------------------------------------------
# Functions for computing limits of HDI's:

HDIofMCMC = function( sampleVec , credMass=0.95 ) {
  # Computes highest density interval from a sample of representative values,
  #   estimated as shortest credible interval.
  # Arguments:
  #   sampleVec
  #     is a vector of representative values from a probability distribution.
  #   credMass
  #     is a scalar between 0 and 1, indicating the mass within the credible
  #     interval that is to be estimated.
  # Value:
  #   HDIlim is a vector containing the limits of the HDI
  sortedPts = sort( sampleVec )
  ciIdxInc = ceiling( credMass * length( sortedPts ) )
  nCIs = length( sortedPts ) - ciIdxInc
  ciWidth = rep( 0 , nCIs )
  for ( i in 1:nCIs ) {
    ciWidth[ i ] = sortedPts[ i + ciIdxInc ] - sortedPts[ i ]
  }
  HDImin = sortedPts[ which.min( ciWidth ) ]
  HDImax = sortedPts[ which.min( ciWidth ) + ciIdxInc ]
  HDIlim = c( HDImin , HDImax )
  return( HDIlim )
}

HDIofICDF = function( ICDFname , credMass=0.95 , tol=1e-8 , ... ) {
  # Arguments:
  #   ICDFname is R's name for the inverse cumulative density function
  #     of the distribution.
  #   credMass is the desired mass of the HDI region.
  #   tol is passed to R's optimize function.
  # Return value:
  #   Highest density iterval (HDI) limits in a vector.
  # Example of use: For determining HDI of a beta(30,12) distribution, type
  #   HDIofICDF( qbeta , shape1 = 30 , shape2 = 12 )
  #   Notice that the parameters of the ICDFname must be explicitly named;
  #   e.g., HDIofICDF( qbeta , 30 , 12 ) does not work.
  # Adapted and corrected from Greg Snow's TeachingDemos package.
  incredMass =  1.0 - credMass
  intervalWidth = function( lowTailPr , ICDFname , credMass , ... ) {
    ICDFname( credMass + lowTailPr , ... ) - ICDFname( lowTailPr , ... )
  }
  optInfo = optimize( intervalWidth , c( 0 , incredMass ) , ICDFname=ICDFname ,
                      credMass=credMass , tol=tol , ... )
  HDIlowTailPr = optInfo$minimum
  return( c( ICDFname( HDIlowTailPr , ... ) ,
             ICDFname( credMass + HDIlowTailPr , ... ) ) )
```

```r
}

HDIofGrid = function( probMassVec , credMass=0.95 ) {
  # Arguments:
  #   probMassVec is a vector of probability masses at each grid point.
  #   credMass is the desired mass of the HDI region.
  # Return value:
  #   A list with components:
  #   indices is a vector of indices that are in the HDI
  #   mass is the total mass of the included indices
  #   height is the smallest component probability mass in the HDI
  # Example of use: For determining HDI of a beta(30,12) distribution
  #   approximated on a grid:
  #   > probDensityVec = dbeta( seq(0,1,length=201) , 30 , 12 )
  #   > probMassVec = probDensityVec / sum( probDensityVec )
  #   > HDIinfo = HDIofGrid( probMassVec )
  #   > show( HDIinfo )
  sortedProbMass = sort( probMassVec , decreasing=TRUE )
  HDIheightIdx = min( which( cumsum( sortedProbMass ) >= credMass ) )
  HDIheight = sortedProbMass[ HDIheightIdx ]
  HDImass = sum( probMassVec[ probMassVec >= HDIheight ] )
  return( list( indices = which( probMassVec >= HDIheight ) ,
                mass = HDImass , height = HDIheight ) )
}

#------------------------------------------------------------------------------
# Function(s) for plotting properties of mcmc coda objects.

DbdaAcfPlot = function( codaObject , parName=varnames(codaObject)[1] , plColors=NULL ) {
  if ( all( parName != varnames(codaObject) ) ) {
    stop("parName must be a column name of coda object")
  }
  nChain = length(codaObject)
  if ( is.null(plColors) ) plColors=1:nChain
  xMat = NULL
  yMat = NULL
  for ( cIdx in 1:nChain ) {
    acfInfo = acf(codaObject[,c(parName)][[cIdx]],plot=FALSE)
    xMat = cbind(xMat,acfInfo$lag)
    yMat = cbind(yMat,acfInfo$acf)
  }
  matplot( xMat , yMat , type="o" , pch=20 , col=plColors , ylim=c(0,1) ,
           main="" , xlab="Lag" , ylab="Autocorrelation" )
  abline(h=0,lty="dashed")
  EffChnLngth = effectiveSize(codaObject[,c(parName)])
  text( x=max(xMat) , y=max(yMat) , adj=c(1.0,1.0) , cex=1.25 ,
        labels=paste("ESS =",round(EffChnLngth,1)) )
}

DbdaDensPlot = function( codaObject , parName=varnames(codaObject)[1] , plColors=NULL ) {
  if ( all( parName != varnames(codaObject) ) ) {
    stop("parName must be a column name of coda object")
  }
  nChain = length(codaObject) # or nchain(codaObject)
  if ( is.null(plColors) ) plColors=1:nChain
  xMat = NULL
  yMat = NULL
  hdiLims = NULL
  for ( cIdx in 1:nChain ) {
    densInfo = density(codaObject[,c(parName)][[cIdx]])
    xMat = cbind(xMat,densInfo$x)
    yMat = cbind(yMat,densInfo$y)
    hdiLims = cbind(hdiLims,HDIofMCMC(codaObject[,c(parName)][[cIdx]]))
  }
  matplot( xMat , yMat , type="l" , col=plColors ,
           main="" , xlab="Param. Value" , ylab="Density" )
  abline(h=0)
  points( hdiLims[1,] , rep(0,nChain) , col=plColors , pch="|" )
  points( hdiLims[2,] , rep(0,nChain) , col=plColors , pch="|" )
  text( mean(hdiLims) , 0 , "" , adj=c(0.5,-0.2) )
  EffChnLngth = effectiveSize(codaObject[,c(parName)])
  MCSE = sd(as.matrix(codaObject[,c(parName)]))/sqrt(EffChnLngth)
  text( max(xMat) , max(yMat) , adj=c(1.0,1.0) , cex=1.25 ,
        paste("MCSE =\n",signif(MCSE,3)) )
}

diagMCMC = function( codaObject , parName=varnames(codaObject)[1] ,
                     saveName=NULL , saveType="jpg" ) {
  DBDAplColors = c("darkolivegreen3", "black", "#FF8F00", "#4F94CD")
  openGraph(height=5,width=7)
  par( mar=0.5+c(3,4,1,0) , oma=0.1+c(0,0,2,0) , mgp=c(2.25,0.7,0) ,
       cex.lab=1.5 )
  layout(matrix(1:4,nrow=2))
  # traceplot and gelman.plot are from CODA package:
  require(coda)
  coda::traceplot( codaObject[,c(parName)] , main="" , ylab="Param. Value" ,
                   col=DBDAplColors )
  tryVal = try(
    coda::gelman.plot( codaObject[,c(parName)] , main="" , auto.layout=FALSE ,
                       col=DBDAplColors )
  )
  # if it runs, gelman.plot returns a list with finite shrink values:
  if ( class(tryVal)=="try-error" ) {
    plot.new()
    print(paste0("Warning: coda::gelman.plot fails for ",parName))
  } else {
    if ( class(tryVal)=="list" & !is.finite(tryVal$shrink[1]) ) {
      plot.new()
      print(paste0("Warning: coda::gelman.plot fails for ",parName))
    }
  }
  DbdaAcfPlot(codaObject,parName,plColors=DBDAplColors)
  DbdaDensPlot(codaObject,parName,plColors=DBDAplColors)
  mtext( text=parName , outer=TRUE , adj=c(0.5,0.5) , cex=2.0 )
  if ( !is.null(saveName) ) {
    saveGraph( file=paste0(saveName,"Diag",parName), type=saveType)
  }
}

diagStanFit = function( stanFit , parName ,
                        saveName=NULL , saveType="jpg" ) {
  codaFit = mcmc.list( lapply( 1:ncol(stanFit) ,
                               function(x) { mcmc(as.array(stanFit)[,x,]) } ) )
  DBDAplColors = c("darkolivegreen3", "black", "#FF8F00", "#4F94CD")
  openGraph(height=5,width=7)
  par( mar=0.5+c(3,4,1,0) , oma=0.1+c(0,0,2,0) , mgp=c(2.25,0.7,0) , cex.lab=1.5 )
  layout(matrix(1:4,nrow=2))
  # traceplot is from rstan package
  require(rstan)
  traceplot(stanFit,pars=parName,nrow=1,ncol=1)#,main="",ylab="Param. Value",col=DBDAplColors)
  # gelman.plot are from CODA package:
  require(coda)
  tryVal = try(
    coda::gelman.plot( codaObject[,c(parName)] , main="" , auto.layout=FALSE ,
                       col=DBDAplColors )
  )
  # if it runs, gelman.plot returns a list with finite shrink values:
  if ( class(tryVal)=="try-error" ) {
    plot.new()
    print(paste0("Warning: coda::gelman.plot fails for ",parName))
  } else {
    if ( class(tryVal)=="list" & !is.finite(tryVal$shrink[1]) ) {
      plot.new()
      print(paste0("Warning: coda::gelman.plot fails for ",parName))
    }
  }
  DbdaAcfPlot(codaFit,parName,plColors=DBDAplColors)
  DbdaDensPlot(codaFit,parName,plColors=DBDAplColors)
  mtext( text=parName , outer=TRUE , adj=c(0.5,0.5) , cex=2.0 )
  if ( !is.null(saveName) ) {
    saveGraph( file=paste0(saveName,"Diag",parName), type=saveType)
  }
}

#------------------------------------------------------------------------------
# Functions for summarizing and plotting distribution of a large sample;
# typically applied to MCMC posterior.

normalize = function( v ){ return( v / sum(v) ) }

require(coda) # loaded by rjags, but redundancy doesn't hurt

summarizePost = function( paramSampleVec ,
                          compVal=NULL , ROPE=NULL , credMass=0.95 ) {
  meanParam = mean( paramSampleVec )
  medianParam = median( paramSampleVec )
  dres = density( paramSampleVec )
  modeParam = dres$x[which.max(dres$y)]
  mcmcEffSz = round( effectiveSize( paramSampleVec ) , 1 )
  names(mcmcEffSz) = NULL
  hdiLim = HDIofMCMC( paramSampleVec , credMass=credMass )
  if ( !is.null(compVal) ) {
    pcgtCompVal = ( 100 * sum( paramSampleVec > compVal )
                    / length( paramSampleVec ) )
  } else {
    compVal=NA
    pcgtCompVal=NA
  }
  if ( !is.null(ROPE) ) {
    pcltRope = ( 100 * sum( paramSampleVec < ROPE[1] )
                 / length( paramSampleVec ) )
    pcgtRope = ( 100 * sum( paramSampleVec > ROPE[2] )
                 / length( paramSampleVec ) )
    pcinRope = 100-(pcltRope+pcgtRope)
  } else {
    ROPE = c(NA,NA)
    pcltRope=NA
    pcgtRope=NA
    pcinRope=NA
  }
  return( c( Mean=meanParam , Median=medianParam , Mode=modeParam ,
             ESS=mcmcEffSz ,
             HDImass=credMass , HDIlow=hdiLim[1] , HDIhigh=hdiLim[2] ,
             CompVal=compVal , PcntGtCompVal=pcgtCompVal ,
             ROPElow=ROPE[1] , ROPEhigh=ROPE[2] ,
             PcntLtROPE=pcltRope , PcntInROPE=pcinRope , PcntGtROPE=pcgtRope ) )
}

plotPost = function( paramSampleVec , cenTend=c("mode","median","mean")[1] ,
                     compVal=NULL, ROPE=NULL, credMass=0.95, HDItextPlace=0.7,
                     xlab=NULL , xlim=NULL , yaxt=NULL , ylab=NULL ,
                     main=NULL , cex=NULL , cex.lab=NULL ,
                     col=NULL , border=NULL , showCurve=FALSE , breaks=NULL ,
                     ... ) {
  # Override defaults of hist function, if not specified by user:
  # (additional arguments "..." are passed to the hist function)
  if ( is.null(xlab) ) xlab="Param. Val."
  if ( is.null(cex.lab) ) cex.lab=1.5
  if ( is.null(cex) ) cex=1.4
  if ( is.null(xlim) ) xlim=range( c( compVal , ROPE , paramSampleVec ) )
  if ( is.null(main) ) main=""
  if ( is.null(yaxt) ) yaxt="n"
  if ( is.null(ylab) ) ylab=""
  if ( is.null(col) ) col="darkolivegreen3"
  if ( is.null(border) ) border="white"

  # convert coda object to matrix:
  if ( class(paramSampleVec) == "mcmc.list" ) {
    paramSampleVec = as.matrix(paramSampleVec)
  }

  summaryColNames = c("ESS","mean","median","mode",
                      "hdiMass","hdiLow","hdiHigh",
                      "compVal","pGtCompVal",
                      "ROPElow","ROPEhigh","pLtROPE","pInROPE","pGtROPE")
  postSummary = matrix( NA , nrow=1 , ncol=length(summaryColNames) ,
                        dimnames=list( c( xlab ) , summaryColNames ) )

  # require(coda) # for effectiveSize function
  postSummary[,"ESS"] = effectiveSize(paramSampleVec)

  postSummary[,"mean"] = mean(paramSampleVec)
  postSummary[,"median"] = median(paramSampleVec)
  mcmcDensity = density(paramSampleVec)
  postSummary[,"mode"] = mcmcDensity$x[which.max(mcmcDensity$y)]

  HDI = HDIofMCMC( paramSampleVec , credMass )
  postSummary[,"hdiMass"]=credMass
  postSummary[,"hdiLow"]=HDI[1]
  postSummary[,"hdiHigh"]=HDI[2]

  # Plot histogram.
  cvCol = "#FF8F00"
  ropeCol = "#4F94CD"
  if ( is.null(breaks) ) {
    if ( max(paramSampleVec) > min(paramSampleVec) ) {
      breaks = c( seq( from=min(paramSampleVec) , to=max(paramSampleVec) ,
                       by=(HDI[2]-HDI[1])/18 ) , max(paramSampleVec) )
    } else {
      breaks=c(min(paramSampleVec)-1.0E-6,max(paramSampleVec)+1.0E-6)
      border="darkolivegreen3"
    }
  }
  if ( !showCurve ) {
    par(xpd=NA)
    histinfo = hist( paramSampleVec , xlab=xlab , yaxt=yaxt , ylab=ylab ,
                     freq=F , border=border , col=col ,
                     xlim=xlim , main=main , cex=cex , cex.lab=cex.lab ,
                     breaks=breaks , ... )
  }
  if ( showCurve ) {
    par(xpd=NA)
    histinfo = hist( paramSampleVec , plot=F )
    densCurve = density( paramSampleVec , adjust=2 )
    plot( densCurve$x , densCurve$y , type="l" , lwd=5 , col=col , bty="n" ,
          xlim=xlim , xlab=xlab , yaxt=yaxt , ylab=ylab ,
          main=main , cex=cex , cex.lab=cex.lab , ... )
  }
  cenTendHt = 0.9*max(histinfo$density)
  cvHt = 0.7*max(histinfo$density)
  ROPEtextHt = 0.55*max(histinfo$density)
  # Display central tendency:
  mn = mean(paramSampleVec)
  med = median(paramSampleVec)
  mcmcDensity = density(paramSampleVec)
```

```
mo = mcmcDensity$x[which.max(mcmcDensity$y)]
if ( cenTend=="mode" ){
  text( mo , cenTendHt ,
        bquote(mode==.(signif(mo,3))) , adj=c(.5,0) , cex=cex )
}
if ( cenTend=="median" ){
  text( med , cenTendHt ,
        bquote(median==.(signif(med,3))) , adj=c(.5,0) , cex=cex , col=cvCol )
}
if ( cenTend=="mean" ){
  text( mn , cenTendHt ,
        bquote(mean==.(signif(mn,3))) , adj=c(.5,0) , cex=cex )
}
# Display the comparison value.
if ( !is.null( compVal ) ) {
  pGtCompVal = sum( paramSampleVec > compVal ) / length( paramSampleVec )
  pLtCompVal = 1 - pGtCompVal
  lines( c(compVal,compVal) , c(0.96*cvHt,0) ,
         lty="dotted" , lwd=2 , col=cvCol )
  text( compVal , cvHt ,
        #bquote( .(round(100*pLtCompVal,1)) * "% < " *
                 #.(signif(compVal,3)) * " < " *
                 #.(round(100*pGtCompVal,1)) * "%" ) ,
        adj=c(pLtCompVal,0) , cex=0.8*cex , col=cvCol )
  postSummary[,"compVal"] = compVal
  postSummary[,"pGtCompVal"] = pGtCompVal
}
# Display the ROPE.
if ( !is.null( ROPE ) ) {
  pInROPE = ( sum( paramSampleVec > ROPE[1] & paramSampleVec < ROPE[2] )
              / length( paramSampleVec ) )
  pGtROPE = ( sum( paramSampleVec >= ROPE[2] ) / length( paramSampleVec ) )
  pLtROPE = ( sum( paramSampleVec <= ROPE[1] ) / length( paramSampleVec ) )
  lines( c(ROPE[1],ROPE[1]) , c(0.96*ROPEtextHt,0) , lty="dotted" , lwd=2 ,
         col=ropeCol )
  lines( c(ROPE[2],ROPE[2]) , c(0.96*ROPEtextHt,0) , lty="dotted" , lwd=2 ,
         col=ropeCol)
  text( mean(ROPE) , ROPEtextHt ,
        #bquote( .(round(100*pLtROPE,1)) * "% < " * .(ROPE[1]) * " < " *
                # .(round(100*pInROPE,1)) * "% < " * .(ROPE[2]) * " < " *
                # .(round(100*pGtROPE,1)) * "%" ) ,
        adj=c(pLtROPE+.5*pInROPE,0) , cex=1 , col=ropeCol )

  postSummary[,"ROPElow"]=ROPE[1]
  postSummary[,"ROPEhigh"]=ROPE[2]
  postSummary[,"pLtROPE"]=pLtROPE
  postSummary[,"pInROPE"]=pInROPE
  postSummary[,"pGtROPE"]=pGtROPE
}
# Display the HDI.
lines( HDI , c(0,0) , lwd=4 , lend=1 )
text( mean(HDI) , 0 , #bquote(.(100*credMass) * "% HDI" ) ,
      adj=c(.5,-1.7) , cex=cex )
```

```r
  text( HDI[1] , 0 , #bquote(.(signif(HDI[1],3))) ,
        adj=c(HDItextPlace,-0.5) , cex=cex )
  text( HDI[2] , 0 , #bquote(.(signif(HDI[2],3))) ,
        adj=c(1.0-HDItextPlace,-0.5) , cex=cex )
  par(xpd=F)
  #
  return( postSummary )
}

#------------------------------------------------------------------------------

# Shape parameters from central tendency and scale:

betaABfromMeanKappa = function( mean , kappa ) {
  if ( mean <=0 | mean >= 1) stop("must have 0 < mean < 1")
  if ( kappa <=0 ) stop("kappa must be > 0")
  a = mean * kappa
  b = ( 1.0 - mean ) * kappa
  return( list( a=a , b=b ) )
}

betaABfromModeKappa = function( mode , kappa ) {
  if ( mode <=0 | mode >= 1) stop("must have 0 < mode < 1")
  if ( kappa <=2 ) stop("kappa must be > 2 for mode parameterization")
  a = mode * ( kappa - 2 ) + 1
  b = ( 1.0 - mode ) * ( kappa - 2 ) + 1
  return( list( a=a , b=b ) )
}

betaABfromMeanSD = function( mean , sd ) {
  if ( mean <=0 | mean >= 1) stop("must have 0 < mean < 1")
  if ( sd <= 0 ) stop("sd must be > 0")
  kappa = mean*(1-mean)/sd^2 - 1
  if ( kappa <= 0 ) stop("invalid combination of mean and sd")
  a = mean * kappa
  b = ( 1.0 - mean ) * kappa
  return( list( a=a , b=b ) )
}

gammaShRaFromMeanSD = function( mean , sd ) {
  if ( mean <=0 ) stop("mean must be > 0")
  if ( sd <=0 ) stop("sd must be > 0")
  shape = mean^2/sd^2
  rate = mean/sd^2
  return( list( shape=shape , rate=rate ) )
}

gammaShRaFromModeSD = function( mode , sd ) {
  if ( mode <=0 ) stop("mode must be > 0")
  if ( sd <=0 ) stop("sd must be > 0")
  rate = ( mode + sqrt( mode^2 + 4 * sd^2 ) ) / ( 2 * sd^2 )
  shape = 1 + mode * rate
  return( list( shape=shape , rate=rate ) )
```

```
}

#------------------------------------------------------------------------------

### LOW-LEVEL script


#===============================================================================
genMCMC = function( datFrm , yName="y" , xNomName="xNom" , xMetName="xMet" ,
                    numSavedSteps=50000 , thinSteps=1 , saveName=NULL ,
                    runjagsMethod=runjagsMethodDefault ,
                    nChains=nChainsDefault ) {
  #-----------------------------------------------------------------------------
  # THE DATA.
  # Convert data file columns to generic xNom,y variable names for model:
  y = as.numeric(datFrm[,yName])
  xNom = as.numeric(as.factor(datFrm[,xNomName]))
  xNomlevels = levels(as.factor(datFrm[,xNomName]))
  xMet = as.numeric(datFrm[,xMetName])
  Ntotal = length(y)
  NxNomLvl = length(unique(xNom))
  # Compute scale properties of data, for passing into prior to make the prior
  # vague on the scale of the data.
  lmInfo = lm( datFrm[,yName] ~ datFrm[,xMetName] + datFrm[,xNomName] )
  residSD = sqrt(mean(lmInfo$residuals^2)) # residual root mean squared deviation
  # For hyper-prior on deflections:
  agammaShRa = unlist( gammaShRaFromModeSD( mode=sd(y)/2 , sd=2*sd(y) ) )
  # Specify the data in a list for sending to JAGS:
  dataList = list(
    y = y ,
    xNom = xNom ,
    xMet = xMet ,
    xMetMean = mean(xMet) ,
    Ntotal = Ntotal ,
    NxNomLvl = NxNomLvl ,
    # data properties for scaling the prior:
    xMetSD = sd(xMet) ,
    yMean = mean(y) ,
    ySD = sd(y) ,
    residSD = residSD ,
    agammaShRa = agammaShRa
  )
  #-----------------------------------------------------------------------------
  # THE MODEL.
  modelstring = "
  model {
  for ( i in 1:Ntotal ) {
  y[i] ~ dnorm( mu[i] , 1/ySigma^2 )
  mu[i] <- a0 + a[xNom[i]] + aMet*( xMet[i] - xMetMean )
  }
  ySigma ~ dunif( residSD/100 , ySD*10 )
  a0 ~ dnorm( yMean , 1/(ySD*5)^2 )
  for ( j in 1:NxNomLvl ) { a[j] ~ dnorm( 0.0 , 1/aSigma^2 ) }
  aSigma ~ dgamma( agammaShRa[1] , agammaShRa[2] )
  aMet ~ dnorm( 0 , 1/(2*ySD/xMetSD)^2 )
  # Convert a0,a[] to sum-to-zero b0,b[] :
  b0 <- a0 + mean( a[1:NxNomLvl] ) + aMet*(-xMetMean)
  for ( j in 1:NxNomLvl ) { b[j] <- a[j] - mean( a[1:NxNomLvl] ) }
  }
  " # close quote for modelstring
  writeLines(modelstring,con="TEMPmodel.txt")
  #------------------------------------------------------------------------------
  # INTIALIZE THE CHAINS.
  initsList = list(
    a0 = mean(y) - lmInfo$coefficients["datFrm[, xMetName]"] * mean(datFrm[,xMetName]) ,
    a = unname( c(
      lmInfo$coef["(Intercept)"]
      + mean(datFrm[,xMetName]) * lmInfo$coef["datFrm[, xMetName]"] ,
      lmInfo$coef["(Intercept)"]
      + mean(datFrm[,xMetName]) * lmInfo$coef["datFrm[, xMetName]"]
      + lmInfo$coef[grep( "xNomName" , names(lmInfo$coef) )] ) ) - mean(y) ,
    aSigma = sd( c(
      lmInfo$coef["(Intercept)"]
      + mean(datFrm[,xMetName]) * lmInfo$coef["datFrm[, xMetName]"] ,
      lmInfo$coef["(Intercept)"]
      + mean(datFrm[,xMetName]) * lmInfo$coef["datFrm[, xMetName]"]
      + lmInfo$coef[grep( "xNomName" , names(lmInfo$coef) )] ) ) ,
    ySigma = residSD ,
    aMet = unname(lmInfo$coefficients["datFrm[, xMetName]"])
    # Let JAGS do other parameters automatically...
  )
  #show( initsList )
  #------------------------------------------------------------------------------
  # RUN THE CHAINS

  parameters = c( "b0" ,  "b" , "aMet" , "aSigma" , "ySigma" )
  adaptSteps = 500
  burnInSteps = 1000
  runJagsOut <- run.jags( method=runjagsMethod ,
                          model="TEMPmodel.txt" ,
                          monitor=parameters ,
                          data=dataList ,
                          inits=initsList ,
                          n.chains=nChains ,
                          adapt=adaptSteps ,
                          burnin=burnInSteps ,
                          sample=ceiling(numSavedSteps/nChains) ,
                          thin=thinSteps ,
                          summarise=FALSE ,
                          plots=FALSE )
  codaSamples = as.mcmc.list( runJagsOut )

  #   nIter = ceiling( ( numSavedSteps * thinSteps ) / nChains )
  #   # Create, initialize, and adapt the model:
  #   jagsModel = jags.model( "TEMPmodel.txt" , data=dataList , inits=initsList ,
  #                           n.chains=nChains , n.adapt=adaptSteps )
```

```
  #   # Burn-in:
  #   cat( "Burning in the MCMC chain...\n" )
  #   update( jagsModel , n.iter=burnInSteps )
  #   # The saved MCMC chain:
  #   cat( "Sampling final MCMC chain...\n" )
  #   codaSamples = coda.samples( jagsModel , variable.names=parameters ,
  #                               n.iter=nIter , thin=thinSteps )

  # resulting codaSamples object has these indices:
  #   codaSamples[[ chainIdx ]][ stepIdx , paramIdx ]
  if ( !is.null(saveName) ) {
    save( codaSamples , file=paste(saveName,"Mcmc.Rdata",sep="") )
  }
  return( codaSamples )
}

#===============================================================================

smryMCMC = function(  codaSamples , datFrm=NULL , xNomName=NULL , xMetName=NULL ,
                      contrasts=NULL , saveName=NULL ) {
  # All single parameters:
  parameterNames = varnames(codaSamples)
  if ( !is.null(datFrm) & !is.null(xNomName) ) {
    xNomlevels = levels(as.factor(datFrm[,xNomName]))
  }
  summaryInfo = NULL
  mcmcMat = as.matrix(codaSamples,chains=TRUE)
  for ( parName in parameterNames ) {
    summaryInfo = rbind( summaryInfo , summarizePost( mcmcMat[,parName] ) )
    thisRowName = parName
    if ( !is.null(datFrm) & !is.null(xNomName) ) {
      # For row name, extract numeric digits from parameter name. E.g., if
      # parameter name is "beta[12,34]" then pull out 12 and 34:
      levelVal = as.numeric(
        grep( "^[1-9]" , # grep only substrings that begin with digits.
              # Return sll substrings split by "[" or "," or "]":
              unlist( strsplit( parName , "\\[|,|\\]"  ) ) ,
              value=TRUE ) )
      if ( length(levelVal) > 0 ) {
        # Assumes there is only a single factor, i.e., levelVal has only entry:
        thisRowName = paste(thisRowName,xNomlevels[levelVal])
      }
    }
    rownames(summaryInfo)[NROW(summaryInfo)] = thisRowName
  }
  # All contrasts:
  if ( !is.null(contrasts) ) {
    if ( is.null(datFrm) | is.null(xNomName) ) {
      show(" *** YOU MUST SPECIFY THE DATA FILE AND FACTOR NAMES TO DO CONTRASTS. ***\n")
    } else {
      # contrasts:
      if ( !is.null(contrasts) ) {
        for ( cIdx in 1:length(contrasts) ) {
```

```r
          thisContrast = contrasts[[cIdx]]
          left = right = rep(FALSE,length(xNomlevels))
          for ( nIdx in 1:length( thisContrast[[1]] ) ) {
            left = left | xNomlevels==thisContrast[[1]][nIdx]
          }
          left = normalize(left)
          for ( nIdx in 1:length( thisContrast[[2]] ) ) {
            right = right | xNomlevels==thisContrast[[2]][nIdx]
          }
          right = normalize(right)
          contrastCoef = matrix( left-right , ncol=1 )
          postContrast = ( mcmcMat[,paste("b[",1:length(xNomlevels),"]",sep="")]
                           %*% contrastCoef )
          summaryInfo = rbind( summaryInfo ,
                               summarizePost( postContrast ,
                                              compVal=thisContrast$compVal ,
                                              ROPE=thisContrast$ROPE ) )
          rownames(summaryInfo)[NROW(summaryInfo)] = (
            paste( paste(thisContrast[[1]],collapse=""), ".v.",
                   paste(thisContrast[[2]],collapse=""),sep="") )
        }
      }
    }
  }
  # Save results:
  if ( !is.null(saveName) ) {
    write.csv( summaryInfo , file=paste(saveName,"SummaryInfo.csv",sep="") )
  }
  return( summaryInfo )
}

#===============================================================================

plotMCMC = function( codaSamples , datFrm , yName , xNomName , xMetName ,
                     contrasts=NULL , saveName=NULL , saveType="jpg" ) {
  mcmcMat = as.matrix(codaSamples,chains=TRUE)
  chainLength = NROW( mcmcMat )
  y = as.numeric(datFrm[,yName])
  xNom = as.numeric(as.factor(datFrm[,xNomName]))
  xNomlevels = levels(as.factor(datFrm[,xNomName]))
  xMet = as.numeric(datFrm[,xMetName])
  Ntotal = length(y)
  NxNomLvl = length(unique(xNom))

  # Display data with posterior predictive distributions:
  for ( xNomLvlIdx in 1:NxNomLvl ) {
    # Open blank graph with appropriate limits:
    xLim = c( min(xMet)-0.2*(max(xMet)-min(xMet)) ,
              max(xMet)+0.2*(max(xMet)-min(xMet)) )
    yLim = c( min(y)-0.2*(max(y)-min(y)) ,
              max(y)+0.2*(max(y)-min(y)) )
    openGraph(width=4,height=5)
    par(mar=c(3,3,3,0.5)) # number of margin lines: bottom,left,top,right
    par(mgp=c(1.75,0.5,0)) # which margin lines to use for labels
    plot(2*max(xLim),2*max(yLim), # point out of range not seen
         xlab=xMetName , xlim=xLim , ylab=yName , ylim=yLim ,
         main=paste(xNomlevels[xNomLvlIdx],"Data\nwith Post. Pred. Distrib.") )
    # plot credible regression lines and noise profiles:
    nSlice = 3
    curveXpos = seq(min(xMet),max(xMet),length=nSlice)
    curveWidth = (max(xMet)-min(xMet))/(nSlice+2)
    nPredCurves=30
    for ( i in floor(seq(from=1,to=nrow(mcmcMat),length=nPredCurves)) ) {
      intercept = mcmcMat[i,"b0"]+mcmcMat[i,paste0("b[",xNomLvlIdx,"]")]
      slope = mcmcMat[i,"aMet"]
      noise = mcmcMat[i,"ySigma"]
      abline( a=intercept , b=slope , col="darkolivegreen3" )
      for ( j in 1:nSlice ) {
        hdiLo = intercept+slope*curveXpos[j] - 1.96*noise
        hdiHi = intercept+slope*curveXpos[j] + 1.96*noise
        yComb = seq( hdiLo , hdiHi , length=75 )
        xVals = dnorm( yComb , mean=intercept+slope*curveXpos[j] , sd=noise )
        xVals = curveWidth * xVals / max(xVals)
        lines( curveXpos[j] - xVals , yComb , col="black" )
        lines( curveXpos[j] - 0*xVals , yComb , col="black" , lwd=2 )
      }
    }
    # plot data points:
    includeVec = ( xNom == xNomLvlIdx )
    xVals = xMet[includeVec]
    yVals = y[includeVec]
    points( xVals , yVals , pch=1 , cex=1.5 , col="#FF8F00" )


    if ( !is.null(saveName) ) {
      saveGraph( file=paste0(saveName,"PostPred-",xNomlevels[xNomLvlIdx]),
                 type=saveType)
    }
  }

  # Display contrast posterior distributions:
  if ( !is.null(contrasts) ) {
    if ( is.null(datFrm) | is.null(xNomName) ) {
      show(" *** YOU MUST SPECIFY THE DATA FILE AND FACTOR NAMES TO DO CONTRASTS. ***\n")
    } else {
      for ( cIdx in 1:length(contrasts) ) {
        thisContrast = contrasts[[cIdx]]
        left = right = rep(FALSE,length(xNomlevels))
        for ( nIdx in 1:length( thisContrast[[1]] ) ) {
          left = left | xNomlevels==thisContrast[[1]][nIdx]
        }
        left = normalize(left)
        for ( nIdx in 1:length( thisContrast[[2]] ) ) {
          right = right | xNomlevels==thisContrast[[2]][nIdx]
        }
        right = normalize(right)
```

```
        contrastCoef = matrix( left-right , ncol=1 )
        postContrast = ( mcmcMat[,paste("b[",1:length(xNomlevels),"]",sep="")]
                         %*% contrastCoef )
        openGraph(height=8,width=4)
        layout(matrix(1:2,ncol=1))
        plotPost( postContrast , xlab="Difference" ,
                  main=paste0(
                    paste(thisContrast[[1]],collapse="."),
                    "\nvs\n",
                    paste(thisContrast[[2]],collapse=".") ) ,
                  compVal=thisContrast$compVal , ROPE=thisContrast$ROPE )
        plotPost( postContrast/mcmcMat[,"ySigma"] ,
                  xlab="Effect Size" ,
                  main=paste0(
                    paste(thisContrast[[1]],collapse="."),
                    "\nvs\n",
                    paste(thisContrast[[2]],collapse=".") ) ,
                  compVal=0.0 ,
                  ROPE=c(-0.1,0.1) )

        if ( !is.null(saveName) ) {
          saveGraph( file=paste0(saveName, paste0(
            paste(thisContrast[[1]],collapse=""),
            ".v.",
            paste(thisContrast[[2]],collapse="") ) ),
            type=saveType )
        }
      }
    }
  } # end if ( !is.null(contrasts) )
}



##### HIGH-LEVEL script


### HIT intervention ### David Moreau, June 14, 2016
### Main analysis (JAGS) ####
### High-level script ###

graphics.off() # This closes all of R's graphics windows.
rm(list=ls())  # Careful! This clears all of R's memory!

#-------------------------------------------------------------------------------
# Set wd and source files
setwd("")
source("HIT_cleanup.R") # tidy data

# Load The data file
library(dplyr)
myDataFrame = hit.wide %>% na.omit # generic from cleanup output file
myDataFrame <- data.frame(myDataFrame)
```

```r
# Specify the column names in the data file relevant to the analysis:
yName="ATT_gain"
xNomName="Condition"
xMetName="ATT_1"              # the covariate

levels(hit.wide$Condition)[levels(hit.wide$Condition)=="0"] <- "control"
levels(hit.wide$Condition)[levels(hit.wide$Condition)=="1"] <- "hit"

# Specify desired contrasts.
# Each main-effect contrast is a list of 2 vectors of level names,
# a comparison value (typically 0.0), and a ROPE (which could be NULL):
contrasts = list("control", "hit", compVal=0.0 , ROPE=c(-1.5,1.5))


# Specify filename root and graphical format for saving output.
# Otherwise specify as NULL or leave saveName and saveType arguments
# out of function calls.
fileNameRoot = "HIT_ATT_BaselineCovariate_"
graphFileType = "eps"

#-------------------------------------------------------------------------------
# Load the relevant model into R's working memory:
source("HIT_Main_Bayesian_Low.R") # the JAGS model
#-------------------------------------------------------------------------------

#-------------------------------------------------------------------------------
# Generate the MCMC chain:
mcmcCoda = genMCMC( datFrm=myDataFrame ,
                    yName=yName , xNomName=xNomName , xMetName=xMetName ,
                    numSavedSteps=11000 , thinSteps=10 , saveName=fileNameRoot )
#-------------------------------------------------------------------------------
# Display diagnostics of chain, for specified parameters:
parameterNames = varnames(mcmcCoda)
show( parameterNames ) # show all parameter names, for reference
for ( parName in parameterNames ) {
  diagMCMC( codaObject=mcmcCoda , parName=parName ,
            saveName=fileNameRoot , saveType=graphFileType )
}
#-------------------------------------------------------------------------------
# Get summary statistics of chain:
summaryInfo = smryMCMC( mcmcCoda , datFrm=myDataFrame , xNomName=xNomName ,
                        xMetName=xMetName , contrasts=contrasts ,
                        saveName=fileNameRoot )
show(summaryInfo)
# Display posterior information:
plotMCMC( mcmcCoda , datFrm=myDataFrame , yName=yName , xNomName=xNomName ,
          xMetName=xMetName , contrasts=contrasts ,
          saveName=fileNameRoot , saveType=graphFileType )
#-------------------------------------------------------------------------------
```

Cognitive and genetic

We will perform the following analyses on cognitive measures: - Repeated measures (Bayesian) ANOVA on each factor score with Session (pretest vs. posttest) as a within factor and Condition (HIT vs. Control) as a between factor. Test of the interaction. Test for homogeneity of variance (Levene's test). - (Bayesian) regression analysis on factor score gains with change in resting heart rate as a predictor, for each group. - (Bayesian) regression analysis on factor score gains with baseline resting heart rate as a predictor (HIT group only). - Repeated measures (Bayesian) ANOVA on each factor score with Session (pretest vs. posttest) as a within factor and BDNF polymorphism (val vs. met) as a between factor Test of the interaction. Test for homogeneity of variance (Levene's test). - Student/Welch two-sample (Bayesian) t-test on each factor score at pretest, by BDNF polymorphism (val66 vs. met66). - Repeated measures (Bayesian) ANOVA on cognitive scores (Flanker – Go/no-go – Stroop – Backward digit span – Backward Corsi blocks – Visual 2-back) with Session (pretest vs. posttest) as a within factor and Condition (HIT vs. Control) as a between factor. Test of the interaction. Test for homogeneity of variance (Levene's test).

We will use default prior scales across analyses (Morey & Rouder, 2015). For Bayesian repeated measures ANOVA and ANCOVA, the prior scale on fixed effects will be set to 0.5, the prior scale on random effects to 1, and the prior scale on the covariate to 0.354. The latter will also be used in Bayesian Linear Regression. The Bayesian t-test uses a Cauchy prior with a width of ~ 0.707, i.e. half of parameter values lies within the interquartile range [-0.707; 0.707]. For transparency, we will plot the prior and posterior distribution for the comparison between Conditions (HIT vs. Control) for all constructs, to be published in the paper or as online supplemental material.

```r
setwd("")
source('HIT_cleanup.R')
library(BayesFactor)

# Frequentist ttest
t.test(genes$ATT_gain ~ genes$BDNFvalmet_1)
t.test(genes$WM_gain ~ genes$BDNFvalmet_1)

# Check for differences at baseline
t.test(genes$ATT_1 ~ genes$BDNFvalmet_1)
t.test(genes$WM_1 ~ genes$BDNFvalmet_1)

# Levene
library(car)
met <- genes %>%
  filter(genes$BDNFvalmet_1=="met")
val <- genes %>%
  filter(genes$BDNFvalmet_1=="val")

leveneTest(met$ATT_1, met$ATT_2)
leveneTest(met$WM_1, met$WM_2)
leveneTest(val$ATT_1, val$ATT_2)
leveneTest(val$WM_1, val$WM_2)

leveneTest(genes$ATT_gain ~ genes$BDNFvalmet_1)
leveneTest(genes$WM_gain ~ genes$BDNFvalmet_1)

# Bayesian ttest
# ATT
bf.ATT <- ttestBF(formula = ATT_gain ~ BDNFvalmet_1, data = genes)
N <- 10000
chains.ATT <- posterior(bf.ATT, iterations = N)
```

```r
summary(chains.ATT)
plot(chains.ATT[,1:4])
chain.ATT <- data.frame(iteration = 1:N, chains.ATT)

# WM
bf.WM <- ttestBF(formula = WM_gain ~ BDNFvalmet_1, data = genes)
N <- 10000
chains.WM <- posterior(bf.WM, iterations = N)
summary(chains.WM)
plot(chains.WM[,1:4])
chain.WM <- data.frame(iteration = 1:N, chains.WM)

# Plots

# ATT
# Chain for mu
p1 <- ggplot(chain.ATT, aes(iteration, -mu)) +
  geom_line() +
  geom_smooth(color="#FF8F00") +
  xlab("Iteration") + ylab(expression(mu)) +
  ggtitle("A") +
  theme_bw() +
  theme(plot.title=element_text(family="Arial", face="bold", size=20)) +
  theme(axis.line.x = element_line(colour = "black", size=.5, linetype = 1),
        axis.line.y = element_line(colour = "black", size=.5, linetype = 1),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.border = element_blank(),
        panel.background = element_blank()) +
  theme(legend.position="none")

# Density for beta
p2 <- ggplot(chain.ATT, aes(-beta..met...val.)) +
  geom_density(color="darkolivegreen3", size=.8) +
  xlab(expression(beta)) + ylab("Density") +
  ggtitle("B") +
  theme_bw() +
  theme(plot.title=element_text(family="Arial", face="bold", size=20)) +
  theme(axis.line.x = element_line(colour = "black", size=.5, linetype = 1),
        axis.line.y = element_line(colour = "black", size=.5, linetype = 1),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.border = element_blank(),
        panel.background = element_blank()) +
  theme(legend.position="none")

# Chain for sigma
p2b <- ggplot(chain.ATT, aes(sig2)) +
  geom_density(color="darkolivegreen3", size=.8) +
  xlab(expression(sigma^2)) + ylab("Density") +
  ggtitle("B") +
  theme_bw() +
  theme(plot.title=element_text(family="Arial", face="bold", size=20)) +
  theme(axis.line.x = element_line(colour = "black", size=.5, linetype = 1),
        axis.line.y = element_line(colour = "black", size=.5, linetype = 1),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.border = element_blank(),
        panel.background = element_blank()) +
  theme(legend.position="none")

# WM
# Chain for mu
p3 <- ggplot(chain.WM, aes(iteration, -mu)) +
  geom_line() +
  geom_smooth(color="#FF8F00") +
  xlab("Iteration") + ylab(expression(mu)) +
  ggtitle("C") +
  theme_bw() +
  theme(plot.title=element_text(family="Arial", face="bold", size=20)) +
  theme(axis.line.x = element_line(colour = "black", size=.5, linetype = 1),
        axis.line.y = element_line(colour = "black", size=.5, linetype = 1),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.border = element_blank(),
        panel.background = element_blank()) +
  theme(legend.position="none")

# Density for beta
p4 <- ggplot(chain.WM, aes(-beta..met...val.)) +
  geom_density(color="darkolivegreen3", size=.8) +
  xlab(expression(beta)) + ylab("Density") +
  ggtitle("D") +
  theme_bw() +
  theme(plot.title=element_text(family="Arial", face="bold", size=20)) +
  theme(axis.line.x = element_line(colour = "black", size=.5, linetype = 1),
        axis.line.y = element_line(colour = "black", size=.5, linetype = 1),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.border = element_blank(),
        panel.background = element_blank()) +
  theme(legend.position="none")

# Chain for sigma
p4b <- ggplot(chain.WM, aes(sig2)) +
  geom_density(color="darkolivegreen3", size=.8) +
  xlab(expression(sigma^2)) + ylab("Density") +
  ggtitle("D") +
  theme_bw() +
  theme(plot.title=element_text(family="Arial", face="bold", size=20)) +
  theme(axis.line.x = element_line(colour = "black", size=.5, linetype = 1),
        axis.line.y = element_line(colour = "black", size=.5, linetype = 1),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.border = element_blank(),
        panel.background = element_blank()) +
  theme(legend.position="none")

# ARRANGE PLOTS ##########################################################

allplots <- grid.arrange(p1, p2b, p3, p4b, ncol=2); allplots
```

Additional

We will also report analyses for which our a priori hypotheses were null effects. These variables will be collected either to control for potential confounds, or for exploratory purposes. Testing for differences will be conducted using (Bayesian) t-tests. We will ensure there is no difference between groups regarding self-reported enjoyment or motivation, to control for expectation effects (Boot, Simons, Stothart, & Stutts, 2013; Stothart, Simons, Boot, & Kramer, 2014). We will also make sure there is no group difference regarding self-reported belief about cognitive malleability (i.e., mindset), ethnic background, age, gender, handedness, height, weight, diagnosis of learning disorder, brain trauma or epileptic seizures, current or past enrolment in a remediation or a cognitive training program, English as first language, videogaming experience, physical exercise, self-reported happiness, sleep quality, or general health.

```
# Split HR_rest on hit.wide
# This is a median split (quantile[3])
# To reproduce analyses, rerun and save files in HIT_JASP.R (write.csv)
hit.wide$HR_split_quant <- ifelse(hit.wide$HR_rest_1 >= quantile(hit.wide$HR_rest_1, na.rm=T)[3], "high
high.HR <- hit.wide %>%
  filter(HR_split_quant == "high") %>%
  #filter(Condition == 1)
t.test(high.HR$HR_gain ~ high.HR$Condition)
table(high.HR$Condition)
exp(ttest.tstat(t=2.5663, n1=75, n2=66, rscale = 0.707)[['bf']])
describeBy(high.HR$HR_gain, group = high.HR$Condition)

# Run comparisons between groups for each variable aforementioned

# Check if HR_gain predicts cognitive gains in high.HR sample
model_ATT <- lm(-high.HR$ATT_gain ~ high.HR$HR_gain)
summary(model_ATT)
plot(-high.HR$ATT_gain ~ high.HR$HR_gain)
abline(model_ATT, col = "blue")

model_WM <- lm(-high.HR$WM_gain ~ high.HR$HR_gain)
summary(model_WM)
plot(-high.HR$WM_gain ~ high.HR$HR_gain)
abline(model_WM, col = "blue")
```