```
# Pruned script for analyses of gene expression data - DVM Bishop 20 Aug 2016

#based on code for simulating Frederickson et al by Jeffrey A. Walker December 6, 2016
# created git site on Feb 16 and copied script file. Original script as of Feb 16 is archived as Fredrickson_multivariate_lm.init.R in the folder 'init code'

#set your working directory here
setwd("C:\\Users\\dbishop\\Dropbox\\Journals and Reviewing\\peerj\\Frederickson critique\\") #set working directory
options(scipen=999) #turn off sci notation
options(digits=4)
library(readr)#added by DB
library(data.table)
library(car)
library(mvtnorm)
library(lmPerm)
library(nlme)
library(lme4)
library(reshape2)
library(doBy)
# plot libraries
library(ggplot2)
library(showtext) # needed for eps fonts to add Arial to .eps
font.add('Arial',regular='Arial.ttf')
library(gridExtra)
library(pls)
library(MRCE) # multivariate regression with correlated error
library(gee) # generalized estimating equations
library(geepack) #ditto
library(GlobalAncova) # GlobalAncova -not available R3.3
library(limma) # Roast not available R3.3
library(CCA) # canonical correlation Analysis
library(MASS) #for generating multivariate normal data
# lmperm downloaded from https://github.com/kabacoff/RiA2/tree/master/lmPerm
# using R-Studio chose Tools > Install Packages and the Install From > Package Archive pop-up menu.


#-----------------------------------------------------
# create dummy dataset
#-----------------------------------------------------
ndata=100 #N simulated rows of data (subjects)
ntype=c(2,10,4) #Ns for independent variables (eg hedonia),
#dependent variables (eg  expression values for each gene)
 #and covariates (e.g. gender)
nvar=sum(ntype)
thistype=c('IV','DV','COV')
mycovs=c(.7,.5,.2)
 #covariance within ivs, dvs, covariates
```

```
#NB assume no significant associations between iv, dv and covs but could change this by modifying cov matrix

#create cov matrix for simulated data
mymean=rep(0,nvar)
mycov=diag(nvar) #extra variable is for group
mYcols=rep('v',nvar) #initialise names
startn=1
endn=ntype[1]
mycounter=0
for (myloop in 1:3){
  if(myloop>1) {startn=startn+ntype[myloop-1]
  endn=endn+ntype[myloop]}
for (j in startn:endn-1) {
  for (k in (j+1):endn){
    mycov[j,k]=mycovs[myloop]
    mycov[k,j]=mycovs[myloop]
  }

}
  for (j in startn:endn)
  {mYcols[j]=paste(thistype[myloop],j-mycounter,sep="")
  }
  mycounter=mycounter+ntype[myloop]}

mydata = data.table(mvrnorm(ndata,mymean,mycov )) #creates a set of ndata rows with zscores with covariances as in mycov
colnames(mydata)=mYcols
Xcols=mYcols[1:ntype[1]] #column names for ivs
Ycols=mYcols[(ntype[1]+1):(ntype[1]+ntype[2])] #col names for dvs
Zcols=mYcols[(ntype[1]+ntype[2]+1):(ntype[1]+ntype[2]+ntype[3])] #col names for covariates
regressors=c(Xcols,Zcols)
#NB uppercase used here to avoid confusion with different usage by Walker


attach(mydata)
write.csv(mydata, file="mydata.csv", row.names=TRUE) #if you want to compare results with other packages
#you can read mydata.csv in to SPSS for instance
#-----------------------------------------------------
# Call statistical functions - need thousands of iterations for proper usage, but set to small N for testing, variable niter specifies this
# N.B. need to run functions before running script (see below)

niter=20; initer=20
myoutfile='myoutput' #summary output will be written to file with this header name
```

```
btable=permutation_test_anderson(mydata,Xcols,Ycols,Zcols,niter,write_it=TRUE,myoutfile
,multi=FALSE)
   #generates file coeff.myoutput.txt and t.myoutput.txt
   # These are standardized partial regression coeffs and t-test values you get
   # if you put all IV + covariates in as predictors with each DV as predicted var


   bootstrap_test(niter,mydata,regressors,Ycols,write_it=TRUE,
myfile=paste(myoutfile,'.bootstrap.txt',sep=''))
   #generates myoutput.bootstrap.txt

   bootstrap_models (mydata, Xcols,Ycols,Zcols,myoutfile, tests=c('mv','gls','gee'), niter)
   #generates output such as myoutput.TQYO.bootstrap.gls.table, where the 4 letters are
random identifiers
   #Outputs for gee and gls


   bootstrap_obrien(mydata,regressors,Ycols, myoutfile, niter,initer=niter) #has additional
iteration within function which is initer - can be difft from niter
  #generates output myoutput.boot.txt

   permutation_obrien(mydata,Xcols,Ycols,Zcols,myoutfile, niter,initer)
   #generates myoutput.perm.txt

detach(mydata)


#############################################################################
#       FUNCTIONS
#############################################################################


read_permutation.gls_list <- function(myfile){
  the_list <- as.character(read.table(myfile)[,1])
  res <- data.table(NULL)
  for(i in 1:length(the_list)){
    sfn <- the_list[i]
    res <- rbind(res,read.table(sfn,header=TRUE))
  }

  # check for multiple rows with perm=obs
  obs_rows <- which(res[,permutation]=='obs')
  exc <- setdiff(obs_rows,obs_rows[1])
  if(length(exc>=1)){res <- res[-exc,]}
  return(res)
}
#############################################################################
read_bootstrap.gls_list <- function(myfile){
  the_list <- as.character(read.table(myfile)[,1])
```

```
  res <- data.table(NULL)
  for(i in 1:length(the_list)){
    sfn <- the_list[i]
    res <- rbind(res,read.table(sfn,header=TRUE))
  }

  # check for multiple rows with perm=obs
  obs_rows <- which(res[,samp]=='obs')
  exc <- setdiff(obs_rows,obs_rows[1])
  if(length(exc>=1)){res <- res[-exc,]}
  return(res)
}


#############################################################################
round_dt <- function(df, digits) {
  df <- data.frame(df)
  nums <- vapply(df, is.numeric, FUN.VALUE = logical(1))
  df[,nums] <- (round(df[,nums], digits = digits))
  df <- data.table(df)
  (df)
}
#############################################################################
round_df <- function(df, digits) {
  nums <- vapply(df, is.numeric, FUN.VALUE = logical(1))
  df[,nums] <- round(df[,nums], digits = digits)
  (df)
}



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

permutation_test_anderson <- function(dt,Xcols,Ycols,Zcols,niter, write_it=FALSE,myfile,
multi=FALSE){
  # permutation test based on Anderson and Robinson 2001
  # dt is a data.table with the X IVs and Y DVs and Z covariates
  # myfile is the file name to write to
  # multi: if TRUE then do GLH tests
  # notes: the expected association between permuted IVs and DVs is zero so the expected
delta is zero E(E(b.h)-E(b.e))=0-0.

  # returns two different results
  # bmatrix is a matrix of the coefficients for each permuation (observed in row I=1)


  p <- length(Ycols)
  Y <- scale(as.matrix(dt[, .SD, .SDcols=Ycols]))
  dt[,IV1:=scale(IV1)]
  dt[,IV2:=scale(IV2)]
```

```
  # coefficients (save t-value in addition to coefficients)
  coeffs.IV1 <- matrix(0,nrow=niter,ncol=p)
  coeffs.IV2 <- matrix(0,nrow=niter,ncol=p)
  colnames(coeffs.IV1) <- Ycols
  colnames(coeffs.IV2) <- Ycols
  t.IV1 <- matrix(0,nrow=niter,ncol=p)
  t.IV2 <- matrix(0,nrow=niter,ncol=p)
  colnames(t.IV1) <- Ycols
  colnames(t.IV2) <- Ycols


  # Get residuals from covariates ( excluding IV1 and IV2)
  form <- formula(paste('Y',paste(Zcols,collapse='+'),sep='~'))
  fit.obs <- lm(form, data=dt)#These are b coeffs using gene values as d.v. and covariates as
predictors
  yhat <- predict(fit.obs) # Yhat = aX #equiv to fit.obs$fitted.values
  rows <- 1:nrow(dt) # observed on first iter and permuted after
  for(iter in 1:niter){
    e <- residuals(fit.obs)[rows,] # permuted Yhat - aX
    # yhat is that predicted by covariates, e contains the residual or unpredicted or what is left
to be predicted by IV1 and IV2.
    # So permute e and there should be no expected correlation between IVs and DVs
    Ypi <- yhat + e # if iter=1 then ypi=y, otherwise permuted
    form <- formula(paste('Ypi',paste(c(Xcols,Zcols),collapse='+'),sep='~'))
    fitmv.pi <- lm(form, data=dt)


    ss <- summary(fitmv.pi)
    for(j in 1:p){ # save t-values instead of coefficients
      coeffs.IV1[iter,j] <- ss[[paste('Response ',Ycols[j],sep='')]]$coefficients['IV1', "Estimate"]
      coeffs.IV2[iter,j] <- ss[[paste('Response ',Ycols[j],sep='')]]$coefficients['IV2', "Estimate"]
      t.IV1[iter,j] <- ss[[paste('Response ',Ycols[j],sep='')]]$coefficients['IV1', "t value"]
      t.IV2[iter,j] <- ss[[paste('Response ',Ycols[j],sep='')]]$coefficients['IV2', "t value"]
    }

    # permute rows
    rows <- sample(1:nrow(dt))
  }

  # matrix of regression coefficients for each DV
  b_table <- data.table(I=rep(1:niter,2), data=rep(c('obs',rep('perm',niter-1)),2),
Type=rep(c('IV1','IV2'),each=niter), rbind(coeffs.IV1,coeffs.IV2))
  # matrix of regression t stats for each DV
  t_table <- data.table(I=rep(1:niter,2), data=rep(c('obs',rep('perm',niter-1)),2),
Type=rep(c('IV1','IV2'),each=niter), rbind(t.IV1,t.IV2))

  fn1 <- paste('coeff.',myfile,'.txt',sep='')
  if(write_it==TRUE){write.table(b_table,fn1,quote=FALSE,sep='\t',row.names=FALSE)}
  fn2 <- paste('t.',myfile,'.txt',sep='')
```

```
  if(write_it==TRUE){write.table(t_table,fn2,quote=FALSE,sep='\t',row.names=FALSE)}

 return(NULL)
}

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

  bootstrap_test<-function(niter,dt,txcols,Ycols,write_it=TRUE, myfile){

  # resamples dt and computes the standardized beta of IV1 and IV2
  # first row is observed data
  # dt is the data.table of IVs, DVs and covariates
  # res is the resulting coefficients of the regression
  # txcols are the independent variables + covariates
  # Ycols are the dependent variables

  # myfile is the base file name to write to
  p <- length(Ycols)
  IV1_matrix <- matrix(0,nrow=niter,ncol=p) # matrix of coefficients
  IV2_matrix <- matrix(0,nrow=niter,ncol=p) # matrix of coefficients
  colnames(IV1_matrix) <- Ycols
  colnames(IV2_matrix) <- Ycols

  rows <- 1:nrow(dt) # observed on first iter and permuted after
  for(iter in 1:niter){
    Y <- scale(as.matrix(dt[rows, .SD, .SDcols=Ycols]))
    dts <- dt[rows,]
    dts[,IV1:=scale(IV1)]
    dts[,IV2:=scale(IV2)]
    # if too few cases with smoke=1 then drop it
    form <- formula(paste('Y',paste(txcols,collapse='+'),sep='~'))
    fitmv <- lm(form, data=dts[,.SD,.SDcols=txcols])
    IV1_matrix[iter,] <- coefficients(fitmv)['IV1',]
    IV2_matrix[iter,] <- coefficients(fitmv)['IV2',]
    rows <- sample(1:nrow(dt),replace=TRUE)
  }
  b_matrix <- data.table(I=rep(1:niter,2), data=rep(c('obs',rep('resamp',niter-1)),2),
Type=rep(c('IV1','IV2'),each=niter), rbind(IV1_matrix,IV2_matrix))


if(write_it==TRUE){write.table(b_matrix,myfile,quote=FALSE,sep='\t',row.names=FALSE)
}
  return(NULL)
}
#-------------------------------------------------------------------------------------------


do_gls_tests <- function(dt,which_file,method='lme',niter=101){
  # dt is a data.table of the responses and regressors
  # which_file is the file FRED13 or FRED15 Fredrickson et. al.
```

```
  # exclude_smoke=TRUE, see below)

  # get xcols and Ycols
  if('IL6' %in% colnames(dt)){year <- 2013}else{year <- 2015}
  Ycols <- c(pro_inflam_genes(year),antibody_genes(),ifn_genes())
  xcols <- get_xcols()

  #gls permutation
  zcols <- c('IV1','IV2')

permutation_gls(dt,xcols=xcols,Ycols=Ycols,zcols=zcols,method=method,niter=niter,do_obs
=TRUE,write_it=TRUE,myfile=paste(which_file,'.permutation.',method,sep=''))

  #gls permutation on individual measures of happiness
  #  zcols <- c('IV1','IV2')
  #  happy <- 'IV2' # limit analysis to zcols
  #  red_xcols <- setdiff(xcols,setdiff(zcols,happy)) # remove the other happy from xcols
  #
permutation_gls(dt,xcols=red_xcols,Ycols=Ycols,zcols=happy,niter=80,do_obs=FALSE,writ
e_it=TRUE,myfile=paste(paste(happy,which_file,sep=''),'.permutation.gls',sep=''))

}


#-------------------------------------------------------------------------------------------------
coeff_vector <- function(dt,Xcols,Ycols){
  # Frederickson et al computation of the regression coefficient of each gene expression on
the set of regressors
  # dt is a data.table
  # xcols are the cols of dt that are the regressors
  # Ycols are the cols of dt that are the univariate response
  p <- length(Ycols)
  b.IV1 <- numeric(p)
  b.IV2 <- numeric(p)
  se.IV1 <- numeric(p) # doing this just to get the vector of se's for modeling
  for(j in 1:p){
    form <- formula(paste(Ycols[j],paste(xcols,collapse='+'),sep='~'))
    fit <- lm(form, data=dt)
    b.IV1[j] <- coefficients(fit)['IV1']
    b.IV2[j] <- coefficients(fit)['IV2']
    se.IV1[j] <- coefficients(summary(fit))['IV1','Std. Error']
  }

  # check ... check!
  #  Y <- as.matrix(dt[,.SD,.SDcols=Ycols])
  #  form <- formula(paste('Y',paste(xcols,collapse='+'),sep='~'))
  #  fitmv <- lm(form, data=dt)
  #  b.IV1m <- coefficients(fitmv)['IV1',]
  #  b.IV2m <- coefficients(fitmv)['IV2',]
  #  data.table(uni=b.IV1,multi=b.IV1m)
```

```
  coeff_table <- data.frame(b.IV1=b.IV1,b.IV2=b.IV2)
  rownames(coeff_table) <- Ycols
  return(coeff_table)
}
#-------------------------------------------------------------------------------
get_ols_table <- function(cc, which_file){
  # cc input are contrast coefficients
  # returns mean beta and SE and CIs

  b_table <- copy(cc)
  b_table[, beta:=apply(.SD,1,mean),.SDcols=Ycols]
  means <- cbind(b_table[Type=='IV1',.(data,IV1=beta)], b_table[Type=='IV2',.(IV2=beta)])
  means[, delta:=IV1-IV2]
  beta_bar <- apply(means[data=='obs', .SD, .SDcols=c('IV1','IV2','delta')],2,mean) # using
apply because it returns the correct format
  beta_se <- apply(means[, .SD, .SDcols=c('IV1','IV2','delta')],2,sd)
  beta_ci <- apply(means[, .SD, .SDcols=c('IV1','IV2','delta')],2,quantile,
probs=c(0.025,0.975))
  ols_table <- data.table(
    Type=c('IV1','IV2','delta'),
    Data=which_file,
    beta=beta_bar,
    SE=beta_se,
    lwr=beta_ci['2.5%',],
    upr=beta_ci['97.5%',]
  )
  return(ols_table)
}

#-------------------------------------------------------------------------------------------
bootstrap_obrien <- function(dt,txcols,Ycols,myfile, niter,initer){
  #initer is the inner bootstrap iteration to get the error on the correlation among the beta
coefficients
  # iter is the outer iteration for bootstrap
#txcols are all regressors, ie IVs and covariates

  N <- nrow(dt)
  p <- length(Ycols)
  J <- matrix(1,nrow=p,ncol=1)
  tJ <- t(J)
  BIV1 <- matrix(0,nrow=initer,ncol=p)
  BIV2 <- matrix(0,nrow=initer,ncol=p)
  Bdelta <- matrix(0,nrow=initer,ncol=p)
  Ttable <- data.table(NULL)
  Y <- dt[,.SD, .SDcols=Ycols] # Y is rescaled in the loop
  rows <- 1:N # observed on first iter and permuted after
  for(iter in 1:niter){
    # don't need to rescale these at this point because they will be rescaled in the inner loop
    Ysamp <- Y[rows,]
```

```
    dt.samp <- dt[rows,]
    # beta for each iteration is the first row in the inner loop
    irows <- 1:nrow(dt) # observed on first iter and permuted after
    # now resample dt and Ypi many time
    for(iiter in 1:initer){
      Y.insamp <- scale(Ysamp[irows,])
      dt.insamp <- dt.samp[irows,]
      dt.insamp[, IV1:=scale(IV1)]
      dt.insamp[, IV2:=scale(IV2)]

      form <- formula(paste('Y.insamp',paste(txcols,collapse='+'),sep='~'))
      fitmv.pi <- lm(form, data=dt.insamp)
      # these need to be the t-value not the coefficient
      BIV1[iiter,] <- coefficients(fitmv.pi)['IV1',]
      BIV2[iiter,] <- coefficients(fitmv.pi)['IV2',]
      Bdelta[iiter,] <- coefficients(fitmv.pi)['IV1',] - coefficients(fitmv.pi)['IV2',]
      irows <- sample(1:N, replace=TRUE)
    }
    R <- cor(BIV1)
    num <- tJ%*%BIV1[1,]
    denom <- c(sqrt(tJ%*%R%*%J))
    TIV1 <- c(num/denom)
    R <- cor(BIV2)
    num <- tJ%*%BIV2[1,]
    denom <- c(sqrt(tJ%*%R%*%J))
    TIV2 <- c(num/denom)
    R <- cor(Bdelta)
    num <- tJ%*%Bdelta[1,]
    denom <- c(sqrt(tJ%*%R%*%J))
    Tdelta <- c(num/denom)
    Ttable <- rbind(Ttable,data.table(TIV1=TIV1,TIV2=TIV2,Tdelta=Tdelta))
    rows <- sample(1:N, replace=TRUE)
  }
  code <- sample(LETTERS,4,replace=TRUE)
  code <- 'boot'
  fn_full <- paste(myfile,'.',paste(code,collapse=''),'.txt',sep='')
  write.table(Ttable,fn_full,quote=FALSE,sep='\t',row.names=FALSE)
}
#-----------------------------------------------------------------------------------------------------
permutation_obrien <- function(dt,Xcols,Ycols,Zcols,myfile, niter,initer){
  # returns the null distribution for O'Brien's T
  #initer is the inner bootstrap iteration to get the error on the correlation among the beta
coefficients
  # iter is the outer iteration for permutation

  N <- nrow(dt)
  p <- length(Ycols)
  J <- matrix(1,nrow=p,ncol=1)
  tJ <- t(J)
  BIV1 <- matrix(0,nrow=initer,ncol=p)
```

```
  BIV2 <- matrix(0,nrow=initer,ncol=p)
  Bdelta <- matrix(0,nrow=initer,ncol=p)
  t.IV1 <- numeric(p)
  t.IV2 <- numeric(p)
  t.delta <- numeric(p)
  Ttable <- data.table(NULL)
  Y <- scale(dt[,.SD, .SDcols=Ycols])
  dts <- copy(dt)
  dts[, IV1:=scale(IV1)]
  dts[, IV2:=scale(IV2)]
  # predicted and residuals for all covariates
  form <- formula(paste('Y',paste(Zcols,collapse='+'),sep='~'))
  fit.obs <- lm(form, data=dts)
  yhat <- predict(fit.obs) # Yhat = aX
  rows <- 1:N # observed on first iter and permuted after
  for(iter in 1:niter){
    e <- residuals(fit.obs)[rows,] # permuted Yhat - aX
    # yhat is that predicted by covariates, e contains the residual or unpredicted or what is left
to be predicted by IV1 and IV2. So permute e and there should be no expected correlation
between happiness and ypi.
    Ypi <- yhat + e # if iter=1 then ypi=y, otherwise permuted
    irows <- 1:nrow(dt) # observed on first iter and permuted after
    # now resample dt and Ypi many time
    for(iiter in 1:initer){
      Yp.samp <- scale(Ypi[irows,])
      dt.samp <- dts[irows,]
      dt.samp[, IV1:=scale(IV1)]
      dt.samp[, IV2:=scale(IV2)]

      form <- formula(paste('Yp.samp',paste(c(Xcols,Zcols),collapse='+'),sep='~'))
      fitmv.pi <- lm(form, data=dt.samp)
      BIV1[iiter,] <- coefficients(fitmv.pi)['IV1',]
      BIV2[iiter,] <- coefficients(fitmv.pi)['IV2',]
      Bdelta[iiter,] <- coefficients(fitmv.pi)['IV1',] - coefficients(fitmv.pi)['IV2',]
      if(iiter==1){
        ss <- summary(fitmv.pi)
        for(j in 1:p){ # save t-values instead of coefficients
          t.IV1[j] <- ss[[paste('Response ',Ycols[j],sep='')]]$coefficients['IV1', "t value"]
          t.IV2[j] <- ss[[paste('Response ',Ycols[j],sep='')]]$coefficients['IV2', "t value"]
          t.delta[j] <- ss[[paste('Response ',Ycols[j],sep='')]]$coefficients['IV1', "t value"] -
ss[[paste('Response ',Ycols[j],sep='')]]$coefficients['IV2', "t value"]
        }
      }
      irows <- sample(1:N, replace=TRUE)
    }
    R <- cor(BIV1)
    num <- sum(t.IV1)
    denom <- sqrt(sum(R)) # c(sqrt(tJ%*%R%*%J))
    TIV1 <- c(num/denom)
    R <- cor(BIV2)
```

```
    num <- sum(t.IV2)
    denom <- sqrt(sum(R)) # c(sqrt(tJ%*%R%*%J))
    TIV2 <- c(num/denom)
    R <- cor(Bdelta)
    num <- sum(t.delta)
    denom <- sqrt(sum(R)) # c(sqrt(tJ%*%R%*%J))
    Tdelta <- c(num/denom)
    Ttable <- rbind(Ttable,data.table(TIV1=TIV1,TIV2=TIV2,Tdelta=Tdelta))
    rows <- sample(1:N, replace=TRUE)
  }
  code <- sample(LETTERS,4,replace=TRUE)
  code <- 'perm'
  fn_full <- paste(myfile,'.',paste(code,collapse=''),'.txt',sep='')
  write.table(Ttable,fn_full,quote=FALSE,sep='\t',row.names=FALSE)
}

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

wald_test <- function(x,se){
  pchisq(x^2/se^2, 1, ncp = 0, lower.tail = FALSE, log.p = FALSE)
}

gee_delta <- function(gee_res){
  # given the two estimates in est and the SEs of the estimates in se,
  # this computes the SE, Wald, and p-value of the difference in the estimates
  delta <- gee_res["IV1",'Estimate'] - gee_res["IV2",'Estimate']
  delta.se <- sqrt(sum(gee_res[,'Std.err']^2))
  delta.wald <- delta^2/delta.se^2
  delta.p <- pchisq(delta.wald, 1, ncp = 0, lower.tail = FALSE, log.p = FALSE)
  gee_res <- rbind(gee_res,c(delta,delta.se,delta.wald,delta.p))
  row.names(gee_res)[3] <- 'delta'
  return(gee_res)
}

#-------------------------------------------------------------------------------
glh.p.values <- function(glh_mod){
  tol <- 1e-12
  p <- nrow(glh_mod$SSPH)
  lambda.h <- eigen(glh_mod$SSPH)$values
  lambda.h <- lambda.h[lambda.h>tol]
  h <- length(lambda.h)
  lambda.e <- eigen(glh_mod$SSPE)$values
  lambda.e <- lambda.e[lambda.e>tol]
  e <- length(lambda.e)
  lambda.he <- eigen(glh_mod$SSPE+glh_mod$SSPE)$values
  lambda.he <- lambda.he[lambda.he>tol]
  he <- length(lambda.he)
  wilks <- det(glh_mod$SSPE)/det(glh_mod$SSPE + glh_mod$SSPH)
  pillai <- sum(diag(glh_mod$SSPH%*%solve(glh_mod$SSPE + glh_mod$SSPH)))
  s <- min(p,h)
```

```
  m <- (abs(p-h)-1)/2
  n <- (e-p-1)/2
  ndf <- s*(2*m+s+1)
  ddf <- s*(2*n+s+1)
  pillaiF <- (2*n+s+1)*pillai/((2*m+s+1)*(s-pillai))
  pillaiP <- pf(pillaiF,ndf,ddf,lower.tail=FALSE)
  return(pillaiP)
}
#---------------------------------------------------
t.test.p.value <- function(x){
  # usage:  t <- apply(res, 2, t.test.p.value)
  return(t.test(x)$p.value)
}
#---------------------------------------------------
t.test.statistic <- function(x){
  # usage:  t <- apply(res, 2, t.test.statistic)
  return(t.test(x)$statistic)
}
#---------------------------------------------------
permutation.p.value <- function(x){
  # obs statistic must be in first cell
  # for two sided, x must be abs(x)
  # usage:  t <- apply(res, 2, permutation.p.value)
  return(length(which(x >= x[1]))/length(x))
}
#---------------------------------------------------
permutation.gls.p.value <- function(res, statistic='t',do_ci=FALSE){
  # res has columns for coefficient and t stat
  # read permuation.gls moved permutation=obs to 1st row and deleted additional
permuation=obs
  if(statistic=='t'){
    inc <- which(substr(colnames(res),1,1)=='t')
    # get p-value for delta
    t.value <- res[,.SD,.SDcols=colnames(res)[inc]]
    inc <- which(substr(colnames(res),1,1)=='c')
    coeff.value <- res[,.SD,.SDcols=colnames(res)[inc]]
    # delta.se <- sqrt(se[Type=='IV1',se]^2 + se[Type=='IV2',se]^2)
    se.value <- coeff.value/t.value
    delta.se <- sqrt(apply(se.value^2,1,sum))
    delta.t <- (coeff.value[,coeff.IV1] - coeff.value[,coeff.IV2])/delta.se
    t.value <- cbind(t.value, delta=delta.t)
    p.value <- apply(abs(t.value),2,permutation.p.value)

    # bootstrap CIs of p-value
    if(do_ci==TRUE){
      t.obs <- t.value[1,]
      niter <- 2000
      bootp <- matrix(0,nrow=niter,ncol=3)
      for(iter in 1:niter){
        rows <- sample(2:nrow(res),(nrow(res)-1),replace=TRUE)
```

```
      inc <- which(substr(colnames(res),1,1)=='t')
      t.value <- res[rows,.SD,.SDcols=colnames(res)[inc]]
      inc <- which(substr(colnames(res),1,1)=='c')
      coeff.value <- res[rows,.SD,.SDcols=colnames(res)[inc]]
      se.value <- coeff.value/t.value
      delta.se <- sqrt(apply(se.value^2,1,sum))
      delta.t <- (coeff.value[,coeff.IV1] - coeff.value[,coeff.IV2])/delta.se
      t.value <- rbind(t.obs,cbind(t.value, delta=delta.t))
      bootp[iter,] <- apply(abs(t.value),2,permutation.p.value)
     }
     ci <- apply(bootp,2,quantile, probs=c(0.025,0.975))
     p.value <- rbind(p.value,ci)
     p.value <- data.table(stat=row.names(p.value),p.value)
    }
  }
  if(statistic=='c'){
    inc <- which(substr(colnames(res),1,1)=='c')
    p.value <- apply(abs(res[,.SD,.SDcols=colnames(res)[inc]]),2,permutation.p.value)
    p.value <- c(p.value, delta=permutation.p.value(abs(res[,coeff.IV1]-res[,coeff.IV2])))
  }
  return(p.value)
}

#-----------------------------------------------------
type.1.error <- function(x, alpha=0.05){
  # obs statistic must be in first cell
  # usage:  t <- apply(res, 2, type.1.error)
  return(length(which(x <= alpha))/length(x))
}
#-----------------------------------------------------
effect_size <- function(dt){
  # fit the UNstandardized Y
  # change in mean expression give 4sd change in IV1 score
  if('IL6' %in% colnames(dt)){year <- 2013}else{year <- 2015}

  Y <- as.matrix(dt[, .SD, .SDcols=Ycols])
  form <- formula(paste('Y',paste(xcols,collapse='+'),sep='~'))
  fitmv <- lm(form, data=dt[,.SD,.SDcols=xcols])
  coeffs <- data.table(IV1=coefficients(fitmv)['IV1',], IV2=coefficients(fitmv)['IV2',])
  obs2 <- (2^(4*apply(coeffs,2,mean)) - 1)*100
  return(obs2)
}

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



#-------------------------------------------------------------------------
smart_t_stats <- function(cc){
  # cc must be the contrast coefficients and not raw regression coefficients
```

```
  cc[,ME:=apply(.SD,1,mean),.SDcols=Ycols, by=Type]
  # create new table with IV1 and IV2 mean coefficients in columns
  beta <- data.table(IV1=cc[Type=='IV1',ME], IV2=cc[Type=='IV2',ME])
  beta[, delta:=IV1-IV2]
  means <- unlist(beta[1,])
  se <- apply(beta,2,sd)
  t <- abs(means/se)
  m <- length(Ycols)
  prob <- 2*pt(t,df=(m-1),lower.tail = FALSE)
  return(prob)
}

#-----------------------------------------------------------------------
permutation_t <- function(cc,statistic='t'){
  # cc must be the contrast coefficients and not raw regression coefficients

  p <- length(Ycols)
  cc[,ME:=apply(.SD,1,mean),.SDcols=Ycols, by=Type]
  cc[,SD:=apply(.SD,1,sd),.SDcols=Ycols, by=Type]
  cc[,SE:=SD/sqrt(p), by=Type]
  cc[,t:=ME/SE, by=Type]

  delta <- cc[Type=='IV1',ME] - cc[Type=='IV2',ME]

  if(statistic=='mean'){
    theta <- data.table(
      IV1=cc[Type=='IV1',ME],
      IV2=cc[Type=='IV2',ME],
      delta=delta
    )
  }
  if(statistic=='t'){
    theta <- data.table(
      IV1=cc[Type=='IV1',t],
      IV2=cc[Type=='IV2',t],
      delta=delta/sqrt(cc[Type=='IV1',SD]^2/p + cc[Type=='IV2',SD]^2/p)
    )
  }

  ols.perm.p <- apply(abs(theta),2,permutation.p.value)
  return(ols.perm.p)
}
#-------------------------------------------------------------------------
permutation_lambda <- function(res){
  # res is the table of Wilk's lambda or Pillai's trace for each iteration
  niter <- nrow(res)
  p_matrix <- data.table(p.IV1=NA,p.IV2=NA,p.delta=NA)

  # is IV1 up regulated? This is two-sided despite the question
  IV1 <- length(which(res$IV1.0 >= res$IV1.0[1]))/niter # permutation p
```

```
  # is eudamonic down regulated?
  IV2 <- length(which(res$IV2.0 >= res$IV2.0[1]))/niter # permutation p
  # is there a difference in regulation?
  delta <- length(which(res$delta.0 >= res$delta.0[1]))/niter # permutation p
  res <- c(IV1,IV2,delta)
  names(res) <- c('IV1','IV2','delta')
  return(res)
}

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

correlation_p.value <- function(dt1, dt2){
  # dt must be the contrast coefficients and not raw regression coefficients
  Ycols <- c(pro_inflam_genes(year=2015),antibody_genes(),ifn_genes())
  r <- numeric(nrow(dt1))
  for(i in 1:nrow(dt1)){
    y1 <- as.numeric(dt1[i,.SD, .SDcols=Ycols])
    y2 <- as.numeric(dt2[i,.SD, .SDcols=Ycols])
    y1s <- y1/sqrt(sum(y1^2))
    y2s <- y2/sqrt(sum(y2^2))
    # sum(y1s^2) # check
    # sum(y2s^2) # check
    r[i] <- as.numeric(t(y1s)%*%y2s)
  }
  niter <- nrow(dt1)/2
  res.table <- data.table(I=dt1[,I],Type=dt1[,Type], r=r)


}
#-----------------------------------------------------------------------------------------
bootstrap_models <- function(dt, Xcols,Ycols,Zcols,which_file, tests, niter){
  # bootstrap estimates using multivariate regression, glm, and gee models

  Y <- as.matrix(dt[, .SD, .SDcols=Ycols])

  rows <- 1:nrow(dt)
  mv_matrix <- matrix(0,nrow=niter,ncol=2)
  colnames(mv_matrix) <- Xcols
  gee_table <- data.table(NULL)
  gls_table <- data.table(NULL)
  code <- paste(sample(LETTERS,4,replace=TRUE),collapse='') #?add 4 random letters to
output?
  gee.out <- paste(which_file,code,'bootstrap.gee.table','txt',sep='.')
  gls.out <- paste(which_file,code,'bootstrap.gls.table','txt',sep='.')
  samp <- 'obs'
  for(iter in 1:niter){
    if('mv' %in% tests){
      Y.samp <- scale(Y[rows,])
      form <- formula(paste('Y.samp~',paste(c(Xcols,Zcols),collapse='+'),sep=''))
      fit <- lm(form,data=dt[rows,])
```

```
      mv_matrix[iter,] <- apply(coefficients(fit)[Xcols,], 1, mean)
    }
    if('gls' %in% tests){
      dtlong <- get_dtlong(dt[rows,],Xcols,Ycols,Zcols,center_Y=TRUE) # returns contrast
coefficients
      dtlong <- orderBy(~subject + DV, dtlong)
      form <- formula(paste('expression~',paste(c('DV',Xcols,Zcols),collapse='+'),sep=''))
      fit.gls <- gls(form, data=dtlong, method='ML', correlation=corCompSymm(form = ~ 1 |
subject), weights=varIdent(form = ~1|DV), control=lmeControl(msMaxIter = 500,
msVerbose = FALSE))
      estimate <- summary(fit.gls)$tTable[Xcols,'Value']
      names(estimate) <- Xcols
      gls_table <- rbind(gls_table,data.table(samp=samp,t(estimate)))
      write.table(gls_table,gls.out,quote=FALSE,row.names=FALSE,sep='\t')
    }
    if('gee' %in% tests){
      dtlong <- get_dtlong(dt[rows,],Xcols,Ycols,Zcols,center_Y=TRUE) # returns contrast
coefficients
      dtlong <- orderBy(~subject + DV, dtlong)
      form <- formula(paste('expression~',paste(c('DV',c(Xcols,Zcols)),collapse='+'),sep=''))
      fit.geeglm <- geeglm(form, family=gaussian, data=dtlong,id=subject,waves=DV,
corstr='exchangeable', std.err="san.se")

      estimate <- summary(fit.geeglm)$coefficients[Xcols,'Estimate']
      names(estimate) <- Xcols
      gee_table <- rbind(gee_table,data.table(samp=samp,t(estimate)))
      write.table(gee_table,gee.out,quote=FALSE,row.names=FALSE,sep='\t')
    }
    rows <- sample(1:nrow(dt),replace=TRUE)
    samp <- 'resample'
  }

  return(NULL)
}

#------------------------------------------------------------
get_dtlong <- function(dt,Xcols,Ycols,Zcols,center_Y=TRUE,factor2numeric=FALSE){
  # dt is a raw data set
  # returns scaled contrast coefficients
  regressors=c(Xcols,Zcols)
  # create matrix with scaled y
  #y.scale <- scale(dt[,.SD,.SDcols=Ycols])
  Y <- scale(dt[,.SD,.SDcols=Ycols],center=center_Y)
  dts <- cbind(dt[,.SD,.SDcols=regressors], Y)
  dts[,subject:=factor(.I)]

  # wide to long
  dtlong <-
melt(dts,id.vars=c('subject',regressors),variable.name='DV',value.name='expression')
  dtlong[,DV:=factor(DV)]
```

```
  return(dtlong)
}
```