# Appendix of Functions

## Exp_out

```
#quickly expoentiate and compute CI's for glm object


exp_out <-function(model_object){
  out<-matrix(nrow=1,ncol=3)
    out[1,1] <-round(exp(summary(model_object)$coefficients[2,1]),2)
    out[1,2] <-round(exp(summary(model_object)$coefficients[2,1] - 1.96*summary(model_object)$coefficients[2,2]),2)
    out[1,3] <-round(exp(summary(model_object)$coefficients[2,1] + 1.96*summary(model_object)$coefficients[2,2]),2)
  colnames(out) <-c('OR','Lower','Upper')
  rownames(out)<-names(model_object$coefficients[2])
  print(out)
}
```

## G Computation with Covariate Adjustment

```
#define inverse logit
expit <-function(x){
  exp(x)/(1+exp(x))
}

#out_type is linear, binary, or count
#ate_type is risk_difference, odds_ratio, or rate_ratio

U.DR = function(est,A,Y,X,OR.formula,out_type="linear",ate_type="difference"){
  data = data.frame(A,Y,X)

  Y_AX = model.matrix(data,object=OR.formula)

  par.Y_AX = est[1:ncol(Y_AX)]
  ATE = est[ncol(Y_AX)+1]

  data1 = data.frame(A=1,Y,X)
  data0 = data.frame(A=0,Y,X)

  #### setup predicted outcomes

  if (out_type=="binary") {

    m1 = expit(model.matrix(data1,object=OR.formula)%*%par.Y_AX)
    m0 = expit(model.matrix(data0,object=OR.formula)%*%par.Y_AX)

    L1 <-mean(m1)
    L0 <-mean(m0)

    U.DR.Y_AX = c(Y-expit(Y_AX%*%par.Y_AX)) *(Y_AX)

  } else if (out_type=="count") {

    m1 = exp(model.matrix(data1,object=OR.formula)%*%par.Y_AX)
    m0 = exp(model.matrix(data0,object=OR.formula)%*%par.Y_AX)

    L1 <-mean(m1)
    L0 <-mean(m0)

    U.DR.Y_AX = c(Y-exp(Y_AX%*%par.Y_AX)) *(Y_AX)

  } else if(out_type=="linear"){
    m1 = model.matrix(data1,object=OR.formula)%*%par.Y_AX
```

```
    m0 = model.matrix(data0,object=OR.formula)%*%par.Y_AX

    U.DR.Y_AX = c(Y-(Y_AX%*%par.Y_AX)) *(Y_AX)
  }

  ###construct estimating equations

  if(ate_type=="odds_ratio"){

    U.DR.ATE = ATE -log((L1/(1-L1))/(L0/(1-L0)))
  } else   if(ate_type=="rate_ratio"){

    U.DR.ATE = ATE -log(L1/L0)
  } else   if(ate_type=="difference"){

    U.DR.ATE = ATE -(m1-m0)
  }


  return(cbind(OR=U.DR.Y_AX,ATE=U.DR.ATE))
}

G = function(est,A,Y,X,OR.formula,out_type,ate_type){
  return(
    apply(U.DR(est,A,Y,X,OR.formula,out_type,ate_type),2,sum)
  )
}


return.CI <-function(est,A,Y,X,OR.formula,out_type="linear",ate_type="difference"){

  meat.half=U.DR(est,A,Y,X,OR.formula,out_type = out_type,ate_type = ate_type)

  bread<-numDeriv::jacobian(func=G,x=est, A=A,Y=Y,X=X, OR.formula=OR.formula,out_type = out_type,ate_t
ype = ate_type)

  IF = meat.half%*%t(solve(-bread))
  ### ATE is the last element of est
  ATE.var = sum(IF[,ncol(IF)]^2)

  out<-matrix(nrow=1,ncol=3)
  if(ate_type=="difference"){
    out[1,1] <-round(ATE,2)
    out[1,2] <-round(ATE-qnorm(0.975)*sqrt(ATE.var),2)
```

```
    out[1,3] <-round(ATE+qnorm(0.975)*sqrt(ATE.var),2)
  } else {
    out[1,1] <-round(exp(ATE),2)
    out[1,2] <-round(exp(ATE-qnorm(0.975)*sqrt(ATE.var)),2)
    out[1,3] <-round(exp(ATE+qnorm(0.975)*sqrt(ATE.var)),2)
  }
  colnames(out) <-c('ATE estimate','CI Lower','CI Upper')
  print(out)

}
```

# G Computation with Spline Propensity Score Adjustment

```
#define inverse logit
expit <-function(x){
  exp(x)/(1+exp(x))
}

#out_type is linear, binary, or count
#ate_type is risk_difference, odds_ratio, or rate_ratio

U.DR_ps = function(est,A,Y,X,PS,PS.formula,OR.formula,out_type="linear",ate_type="difference"){
  data = data.frame(A,Y,X,ps)
  A_X = model.matrix(data=data,object=PS.formula)

  Y_AX = model.matrix(data,object=OR.formula)

  par.A_X = est[1:ncol(A_X)]
  par.Y_AX = est[ncol(A_X)+ 1:ncol(Y_AX)]
  ATE = est[ncol(A_X)+ncol(Y_AX)+1]

  data1 = data.frame(A=1,Y,X,ps)
  data0 = data.frame(A=0,Y,X,ps)

  #### setup predicted outcomes

  if (out_type=="binary") {

    m1 = expit(model.matrix(data1,object=OR.formula)%*%par.Y_AX)
    m0 = expit(model.matrix(data0,object=OR.formula)%*%par.Y_AX)

    L1 <-mean(m1)
    L0 <-mean(m0)

    U.DR.A_X = c(A-expit(A_X%*%par.A_X)) *(A_X)
    U.DR.Y_AX = c(Y-expit(Y_AX%*%par.Y_AX)) *(Y_AX)

  } else if (out_type=="count") {

    m1 = exp(model.matrix(data1,object=OR.formula)%*%par.Y_AX)
    m0 = exp(model.matrix(data0,object=OR.formula)%*%par.Y_AX)

    L1 <-mean(m1)
    L0 <-mean(m0)

    U.DR.A_X = c(A-exp(A_X%*%par.A_X)) *(A_X)
```

```
    U.DR.Y_AX = c(Y-exp(Y_AX%*%par.Y_AX)) *(Y_AX)

  } else if(out_type=="linear"){
    m1 = model.matrix(data1,object=OR.formula)%*%par.Y_AX
    m0 = model.matrix(data0,object=OR.formula)%*%par.Y_AX

    U.DR.A_X = c(A-(A_X%*%par.A_X)) *(A_X)
    U.DR.Y_AX = c(Y-(Y_AX%*%par.Y_AX)) *(Y_AX)
  }

  ###construct estimating equations

  if(ate_type=="odds_ratio"){

    U.DR.ATE = ATE -log((L1/(1-L1))/(L0/(1-L0)))
  } else   if(ate_type=="rate_ratio"){

    U.DR.ATE = ATE -log(L1/L0)
  } else   if(ate_type=="difference"){

    U.DR.ATE = ATE -(m1-m0)
  }


  return(cbind(PS=U.DR.A_X,OR=U.DR.Y_AX,ATE=U.DR.ATE))
}

G_ps = function(est,A,Y,X,PS,PS.formula,OR.formula,out_type,ate_type){
  return(
    apply(U.DR_ps(est,A,Y,X,PS,PS.formula,OR.formula,out_type,ate_type),2,sum)
  )
}


return.CI_ps <-function(est,A,Y,X,PS,PS.formula,OR.formula,out_type="linear",ate_type="difference"){

  meat.half=U.DR_ps(est,A,Y,X,PS,PS.formula,OR.formula,out_type = out_type,ate_type = ate_type)

  bread<-numDeriv::jacobian(func=G_ps,x=est, A=A,Y=Y,X=X,PS=PS, PS.formula=PS.formula, OR.formula=OR.f
ormula,out_type = out_type,ate_type = ate_type)

  IF = meat.half%*%t(solve(-bread))
  ### ATE is the last element of est
  ATE.var = sum(IF[,ncol(IF)]^2)
```

```
  out<-matrix(nrow=1,ncol=3)
  if(ate_type=="difference"){
    out[1,1] <-round(ATE,2)
    out[1,2] <-round(ATE-qnorm(0.975)*sqrt(ATE.var),2)
    out[1,3] <-round(ATE+qnorm(0.975)*sqrt(ATE.var),2)
  } else {
    out[1,1] <-round(exp(ATE),2)
    out[1,2] <-round(exp(ATE-qnorm(0.975)*sqrt(ATE.var)),2)
    out[1,3] <-round(exp(ATE+qnorm(0.975)*sqrt(ATE.var)),2)
  }
  colnames(out) <-c('ATE estimate','CI Lower','CI Upper')
  print(out)

}
```

# Weighted RMST

```r
#Original Code by Sarah C. Conner
# see  https://github.com/s-conner/akm-rmst
# --- RMST Using Adjusted KM ---
# Time is the time to event
# Status is 0 if censored, 1 if event
# Group should be a factor variable
# Weights can be obtained separately, ie through logistic models
# Tau is a user-specified truncation point.
# If not specified, the default will be the minimum of the each groups' last event time

akm_rmst <- function(time, status, group, weight=NULL, tau=NULL, alpha=.05,
                     xaxismin=0, xaxismax=max(time),plot=FALSE){
  if(sum(time<0)>0){print("Error: times must be positive.")
  }else{
    if(sum(weight<=0)>0){print("Error: weights must be greater than 0.")
    }else{
      if(sum(status!=0 & status!=1)>0){print("Error: status must be a vector of 0s and/or 1s.")
      }else{

        if(is.null(weight)){weight <- rep(1, length(time))}
        data <- data.frame(time, status, group, weight)
        data <- data[!is.na(data$group) & !is.na(data$time),]
        data <- data[order(group),]

        #--- If tau not specified, use minimum tau from all groups ---
        j=length(unique(data$group))

        if(is.null(tau)){
          taui = rep(999, 3)
          for (i in (1:j)){
            groupval <- (levels(data$group)[i])
            dat_group <- data[which(data$group==(groupval)),]
            taui[i] <- max(dat_group$time[dat_group$status==1])
          }
          tau <- min(taui)
        }

        #--- Calculate AKM RMST in each group ---
        rmst <- rep(999, length(1:j))
        groupval <- rep(999, length(1:j))
        rmst_var <- rep(999, length(1:j))
        rmst_se <- rep(999, length(1:j))
        if(plot==TRUE){
```

```
          plot(NULL, xlim=c(xaxismin, xaxismax), ylim=c(0,1), xlab='Time',ylab='Adjusted Survival Prob
ability')
          title(main='Adjusted Kaplan-Meier')
        }

        for (i in 1:j){
          groupval[i] <- (levels(data$group)[i])
          dat_group <- data[which(data$group==(groupval[i])),]

          #--- AKM ---
          # Based on 'adjusted.KM' function from {IPWsurvival} package
          # Author: F. Le Borgne and Y. Foucher
          tj <- c(0,sort(unique(dat_group$time[dat_group$status==1])))
          dj <- sapply(tj, function(x){sum(dat_group$weight[dat_group$time==x & dat_group$status==1
])})
          yj <- sapply(tj, function(x){sum(dat_group$weight[dat_group$time>=x])})
          st <- cumprod(1-(dj/yj))
          m <- sapply(tj, function(x){sum((dat_group$weight[dat_group$time>=x])^2)})
          mj <- ((yj^2)/m)
          #ft <- data.frame(time=tj, n_risk=yj, n_event=dj, survival=st, variable=i, m=mj)
          ft <- data.frame(tj, yj, dj, st, i, mj)

          #--- RMST ---
          # Based on 'rmst1 function' from {survRM2} package
          # Author: Hajime Uno, Lu Tian, Angel Cronin, Chakib Battioui, Miki Horiguchi
          rtime <- ft$tj<=tau
          tj_r <- sort(c(ft$tj[rtime],tau))
          st_r <- ft$st[rtime]
          yj_r <- ft$yj[rtime]
          dj_r <- ft$dj[rtime]
          time_diff <- diff(c(0, tj_r))
          areas <- time_diff * c(1, st_r)
          rmst[i] <- sum(areas)

          #--- Variance ---
          mj_r <- ft$mj[rtime]
          #var_r <- ifelse((yj_r-dj_r)==0, 0, dj_r /(mj_r *(yj_r - dj_r)))
          var_r <- ifelse((yj_r-dj_r)==0, 0, dj_r /(yj_r *(yj_r - dj_r)))
          var_r <- c(var_r,0)
          rmst_var[i] <- sum(cumsum(rev(areas[-1]))^2 * rev(var_r)[-1])
          rmst_se[i] <- sqrt(rmst_var[i])

          #--- Plot AKM ---
          if(plot==TRUE){
```

```
            lines(ft$tj, ft$st,type="s", col=(i+2), lwd=2)
          }

        }
      }
    }
  }

  #--- Add legend and tau to plot ---
  if(plot==TRUE){
    abline(v=tau, col=1, lty=3, lwd=2)
    legend('bottomleft', paste("Group", groupval), lty=rep(1, j), lwd=rep(2, j), col=3:(j+2),
           cex=.75, bty ="n", inset = c(0, 0))
  }



  #--- Compare RMST between groups and compile output---
  results <- data.frame(groupval,rmst,rmst_var,rmst_se,tau)
  pwc <- ((j^2)-j)/2   #number of pairwise comparisons

  label_diff <- rep(999,(pwc))
  rmst_diff <- rep(999,(pwc))
  rmst_diff_se <- rep(999,(pwc))
  rmst_diff_low <- rep(999,(pwc))
  rmst_diff_upp <- rep(999,(pwc))
  rmst_diff_pval <- rep(999,(pwc))

  label_ratio <- rep(999,(pwc))
  rmst_logratio <- rep(999,(pwc))
  rmst_logratio_se <- rep(999,(pwc))
  rmst_ratio <- rep(999,(pwc))
  rmst_ratio_low <- rep(999,(pwc))
  rmst_ratio_upp <- rep(999,(pwc))
  rmst_logratio_pval <- rep(999,(pwc))

  output_diff <- data.frame(label_diff,rmst_diff,rmst_diff_se,rmst_diff_low,rmst_diff_upp,rmst_diff_pv
al)
  output_ratio <- data.frame(label_ratio,rmst_logratio,rmst_logratio_se,rmst_ratio,rmst_ratio_low,rmst
_ratio_upp,rmst_logratio_pval)
  l <- 1

  for (i in 1:(j-1)){
    for (j in (i+1):j){
      # Based on 'rmst2 function' from {survRM2} package
```

```
      # Author: Hajime Uno, Lu Tian, Angel Cronin, Chakib Battioui, Miki Horiguchi

      #--- RMST Difference ---
      output_diff[l,]$label_diff <- paste('Groups',results[j,]$groupval,'vs.',results[i,]$groupval,' '
)
      output_diff[l,]$rmst_diff <- (results[j,]$rmst - results[i,]$rmst)
      output_diff[l,]$rmst_diff_se <- sqrt(results[j,]$rmst_var + results[i,]$rmst_var)
      output_diff[l,]$rmst_diff_low <- output_diff[l,]$rmst_diff - qnorm(1-alpha/2)*output_diff[l,]$rm
st_diff_se
      output_diff[l,]$rmst_diff_upp <- output_diff[l,]$rmst_diff + qnorm(1-alpha/2)*output_diff[l,]$rm
st_diff_se
      output_diff[l,]$rmst_diff_pval <- 2*(1-pnorm(abs(output_diff[l,]$rmst_diff)/output_diff[l,]$rmst
_diff_se))

      #--- RMST Ratio ---
      output_ratio[l,]$label_ratio <- paste('Groups',results[j,]$groupval,'vs.',results[i,]$groupval,'
')
      output_ratio[l,]$rmst_logratio <- (log(results[j,]$rmst) - log(results[i,]$rmst))
      output_ratio[l,]$rmst_logratio_se <- sqrt(results[j,]$rmst_var/(results[j,]$rmst^2) + results
[i,]$rmst_var/(results[i,]$rmst^2))
      output_ratio[l,]$rmst_ratio <- exp(output_ratio[l,]$rmst_logratio)
      output_ratio[l,]$rmst_ratio_low <- exp(output_ratio[l,]$rmst_logratio - qnorm(1-alpha/2)*output_
ratio[l,]$rmst_logratio_se)
      output_ratio[l,]$rmst_ratio_upp <- exp(output_ratio[l,]$rmst_logratio + qnorm(1-alpha/2)*output_
ratio[l,]$rmst_logratio_se)
      output_ratio[l,]$rmst_logratio_pval <- 2*(1-pnorm(abs(output_ratio[l,]$rmst_logratio)/output_rat
io[l,]$rmst_logratio_se))

      l <- l+1 #move to next row
    }
  }

  #cat("\n\n\n")
  #cat(paste('RMST calculated up to tau =',round(results$tau[1],3)))
  #cat("\n\n\n")

  #cat ("Restricted Mean Survival Time (RMST) per Group \n\n")
  #colnames(results) <- c("Group", "RMST", "Var", "SE", "Tau")
  #rownames(results) <- c(paste("Group", results$Group,' '))
  #print(round(results[c(2,4)],3))
  #cat("\n\n")

  #cat ("Restricted Mean Survival Time (RMST) Differences \n\n")
  #colnames(output_diff) <- c("Groups", "Est.", "SE", "CIL", "CIU", "p")
```

```
  #rownames(output_diff) <- c(output_diff$Groups)
  #print(round(output_diff[c(2,3,4,5,6)],3))
  #cat("\n\n")

  #cat ("Restricted Mean Survival Time (RMST) Ratios \n\n")
  #colnames(output_ratio) <- c("Groups", "Log Est.", "SE", "Est.", "CIL", "CIU", "p")
  #rownames(output_ratio) <- c(output_ratio$Groups)
  #print(round(output_ratio[c(2,3,4,5,6,7)],3))
  return(output_diff)
}
```

# Sensitivy Analysis - Vibration of Effects

```
###Original Code from  Chirag Patel chirag@hms.harvard.edu
# see https://github.com/chiragjp/voe/tree/gh-pages/vibration
###Modified for Propensity score applications
library(survival, quietly=T)
library(EValue)
library(survey)

run_model <- function(form, data, family='gaussian', weights=NULL,...) {
  args <- list(form, data = data, ...)
  if(family == 'gaussian') {
    return(do.call(lm, args))
  }
  if(family == 'cox') {
    return(do.call(coxph, args))
  }
  if(family == 'binomial') {
    args <- list(form, data, family=binomial(),weights=weights, ...)
    return(do.call(glm, args))
  }
  if(family == 'poisson') {
    args <- list(form, data, family=poisson(), ...)
    return(do.call(glm, args))
  }
  if(family=='match'){
    args <- list(form, data, method = "nearest",caliper=.2,ratio=4, ...)
    return(do.call(matchit, args))
  }
  if(family == 'quasibinomial') {
    args <- list(form, design=data, family=binomial(link='logit'), ...)
    return(do.call(svyglm, args))
  }
}



conductVibrationForK_ps <- function(base_formula,base_out_formula,dataFrame,adjustby,k=1,family=c('gau
ssian', 'binomial', 'cox','poisson'), print_progress=T, ...) {
  initFrame <- function(nrows,ncols) {
    matrix(NA,nrows,ncols)
  }



  addToBase <- function(base_formula, adjustingVariables) {
    form <- base_formula
```

```
    if(length(adjustingVariables)) {
      addStr <- as.formula(sprintf('~ . + %s', paste(adjustingVariables, collapse='+')))
      form <- update.formula(base_formula, addStr)
    }
    return(form)
  }


  variablename <- attr(terms(base_formula), 'term.labels')[1]
  varname <-'treatment' #all.vars(as.formula(sprintf('~%s', variablename)))
  if(print_progress) print(varname);

  if(class(adjustby)=='formula') {
    adjustby <- attr(terms(adjustby), 'term.labels')
  }
  n <- length(adjustby)
  varComb <- combn(n, k)
  retFrame <- NULL
  retFrameCounter <- 1
  bicFrame <- NULL
  for(ii in 1:ncol(varComb)) { # cycle through each possibility
    if(print_progress) cat(sprintf('%i/%i\n',ii, ncol(varComb)));

    adjustingVariables <- adjustby[varComb[, ii]]
    strComb <- paste(sort(varComb[, ii]), collapse=',')
    form <- addToBase(base_formula,adjustingVariables)
    if(print_progress) print(form);
    ps <-run_model(form,dataFrame,family='binomial')


    #MODIFY SECTION BASED ON PROPENSITY METHOD
   dataFrame$pr_score <- predict(ps, type="response")

   dataFrame$pr_score_trim <-ifelse(dataFrame$pr_score<.01,.01,dataFrame$pr_score)
   dataFrame$pr_score_trim <-ifelse(dataFrame$pr_score>.99,.99,dataFrame$pr_score_trim)

   #IPTW
   #dataFrame$IPTW <-dataFrame$treatment/dataFrame$pr_score_trim + (1-dataFrame$treatment)/(1-dataFram
e$pr_score_trim)

   #design.ps <- svydesign(ids=~1, weights=~IPTW, data=dataFrame)

 #Matched
   #matched <-run_model(form,dataFrame,family='match')
```

```
  #matched_data <- match.data(matched)
    ## run the model

    est <- tryCatch(
      #matched
      #run_model(base_out_formula,matched_data,family='binomial',weights =matched_data$weights ),
      #spline
      run_model(base_out_formula,dataFrame,family='binomial'),
      #IPTW
      #run_model(base_out_formula,design.ps,family='quasibinomial' ),
      error=function(err) {
        message(err)
        return(NULL)
      }
    )

    if(!is.null(est)) {
      ## collect the result
      ## do unweightedEst here...

      frm <- coef(summary(est))
      bicMod <- getBIC(est) # do BIC
      ## modify the above...
      ### need to get nlevels of variable
      rowIndex <- grep(varname, rownames(frm))
      nLevels <- length(rowIndex)

      if(length(rowIndex) & is.null(retFrame)) {
        nrows <- ncol(varComb) * nLevels
        ncols <- ncol(frm)
        retFrame <- initFrame(nrows,ncols+2) ## need to add 2 columns for the combination and factor_l
evel
        bicFrame <- initFrame(ncol(varComb), 3) #
        colnames(retFrame) <- c(colnames(frm), 'combination_index', 'factor_level')
        colnames(bicFrame) <- c('edf', 'bic', 'combination_index')
      }

      bicFrame[ii, 'combination_index'] <- ii
      bicFrame[ii, 'edf'] <- bicMod[1]
      bicFrame[ii, 'bic'] <- bicMod[2]

      for(jj in 1:length(rowIndex)) {
        retFrame[retFrameCounter, 1:ncol(frm)] <- frm[rowIndex[jj], ]
```

```
        retFrame[retFrameCounter, ncol(frm)+1] <- ii
        retFrame[retFrameCounter, ncol(frm)+2] <- jj
        retFrameCounter <- retFrameCounter+1
      }

    }

  }
  return(list(vibration=retFrame,bic=bicFrame, k=k,combinations=varComb, family=family, base_formula=b
ase_formula, adjust=adjustby))
}

getBIC <- function(mod) {
  return(extractAIC(mod)) # do BIC
}

recomputePvalue <- function(allData, zStatColName, pValColName) {
  ### some pvalues estimated at 0 because test statistics so large; recompute their pvalues
  zeroPval <- !is.na(allData[,pValColName]) & (allData[,pValColName] == 0)
  allData[zeroPval, pValColName] <- pnorm(abs(allData[zeroPval, zStatColName]), lower.tail=F)*2 #two s
ided pvalue
  return(allData)
}

conductVibration_ps <- function(base_formula,base_out_formula,dataFrame,adjustby,family=c('gaussian',
'binomial', 'cox','poisson'), kMin=NULL, kMax=NULL, print_progress=T, ...) {
  if(is.null(kMin)) {
    kMin <- 1
  }
  if(is.null(kMax)) {
    n <- length(attr(terms(adjustby), 'term.labels'))
    kMax <- n - 1
  }
  cat(sprintf('running models; k start:%i, k stop:%i\n', kMin, kMax))
  retFrame <- list()
  ii <- 1
  for(k in kMin:kMax) {
    vib <- conductVibrationForK_ps(base_formula, base_out_formula,dataFrame, adjustby, k, family, prin
t_progress, ...)
    retFrame[[ii]] <- vib
    ii <- ii + 1
  }
  ret <- gatherFrames(retFrame)
  return(ret)
```

```
}

gatherVibration <- function(returnFrames) {
  ## gathers up results from multiple runs; see conductVibration
  nrows <- c()
  for(ii in 1:length(returnFrames)) {
    nrows <- c(nrows, nrow(returnFrames[[ii]]$vibration))
  }

  retFrame <- matrix(nrow=sum(nrows), ncol=ncol(returnFrames[[1]]$vibration)+1)
  colnames(retFrame) <- c(colnames(returnFrames[[1]]$vibration), 'k')

  startIndex <- 1
  for(ii in 1:length(returnFrames)) {
    ncols <- ncol(returnFrames[[ii]]$vibration)
    retFrame[startIndex:(startIndex+nrows[ii]-1), 1:ncols] <- returnFrames[[ii]]$vibration
    retFrame[startIndex:(startIndex+nrows[ii]-1), ncols+1] <- returnFrames[[ii]]$k
    startIndex <- startIndex+nrows[ii]
  }
  return(retFrame)
}

gatherVibrationBIC <- function(returnFrames) {
  nrows <- c()
  for(ii in 1:length(returnFrames)) {
    nrows <- c(nrows, nrow(returnFrames[[ii]]$bic))
  }

  retFrame <- matrix(nrow=sum(nrows), ncol=ncol(returnFrames[[1]]$bic)+1)
  colnames(retFrame) <- c(colnames(returnFrames[[1]]$bic), 'k')

  startIndex <- 1
  for(ii in 1:length(returnFrames)) {
    ncols <- ncol(returnFrames[[ii]]$bic)
    retFrame[startIndex:(startIndex+nrows[ii]-1), 1:ncols] <- returnFrames[[ii]]$bic
    retFrame[startIndex:(startIndex+nrows[ii]-1), ncols+1] <- returnFrames[[ii]]$k
    startIndex <- startIndex+nrows[ii]
  }
  return(retFrame)
}

column_headers <- function(vibFrame, family) {
  existingColnames <- colnames(vibFrame)
  newColnames <- NULL
```

```
  if(family == 'cox') {
    isRobust <- grep('robust', existingColnames)
    # if(isRobust) {
    #return(c('estimate', 'HR', 'se', 'robust_se', 'z', 'pvalue', 'combination_index', 'factor_level',
'k'))
    # } else {
    return(c('estimate', 'OR', 'se', 'z', 'pvalue', 'combination_index', 'factor_level', 'k'))
    #}
  } else if(family == 'gaussian') {
    ## to do
    existingColnames[1] <- 'estimate'
    existingColnames[length(existingColnames) - 4] <- 'pvalue'
    return(existingColnames)
  } else if(family == 'binomial' | family == 'poisson') {
    ## to do
    existingColnames[1] <- 'estimate'
    existingColnames[length(existingColnames) - 4] <- 'pvalue'
    return(existingColnames)
  }
  ### fill in the rest later for other families
  return(existingColnames)
}

harmonizeFrame <- function(vibFrame, family) {
  vibFrame <- as.data.frame(vibFrame)
  colnames(vibFrame) <- column_headers(vibFrame, family)
  if(family %in% c('binomial','poisson')) {
    vibFrame$HR <- exp(vibFrame$estimate)
  }
  return(vibFrame)
}

gatherFrames <- function(returnFrames) {
  bic <- gatherVibrationBIC(returnFrames)
  vibration <- gatherVibration(returnFrames)
  combinations <- list()
  for(ii in 1:length(returnFrames)) {
    combinations[[ii]] <- returnFrames[[ii]]$combinations
  }
  family <- returnFrames[[1]]$family
  base_formula <- returnFrames[[1]]$base_formula
  adjust <- returnFrames[[1]]$adjust

  vibration <- harmonizeFrame(vibration, family)
```

```
  #change based on method
  vibration <- recomputePvalue(vibration, 'Pr(>|z|)','pvalue')
  return(list(vibFrame=vibration, bicFrame=bic, combinations=combinations, adjust=adjust, family=famil
y, base_formula=base_formula))
}


## vibration of effects
## plot the VoE distribution
## Chirag Patel cjp@stanford.edu
## 07/05/13


library(ggplot2)
library(RColorBrewer)
CBB_PALETTE <- c("#000000", "#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2", "#D55E00", "#CC79A
7")


vib2d_cox <- function(vibObj, factor_num=1,nbins=20) {
  vibFrame <- vibObj$vibFrame

  nFactor <- length(unique(vibFrame$factor_level))
  yRange <- range(c(-log10(.05), -log10(vibFrame$'Pr(>|z|)')), na.rm=T)
  xRange <- range(vibFrame$HR, na.rm=T)
  probs <- c( 0.5)
  hProbs <- c(0.5)
  subFrame <- subset(vibFrame, factor_level == factor_num)
  subFrame$factor_level <- NULL
  subFrame$pvalue <-subFrame$`Pr(>|t|)`
  estLevel <- statPerK(subFrame)
  estLevel$HR <- exp(estLevel$estimate)
  pQuant <- quantilesPvalue(subFrame, probs)
  hQuant <- quantilesHR(subFrame, hProbs)

  #RHR <- round(hQuant[3,'HR']/hQuant[1,'HR'], 2)
  #RPvalue <- round(-log10(pQuant[1,'pvalue']) + log10(pQuant[3,'pvalue']), 2)
  p <- ggplot(subFrame, aes(HR, -log10(pvalue)))
  if(sum(colnames(subFrame) == 'has_variable')) {
    p <- p + geom_hex(aes(colour=factor(has_variable)),bins=nbins) + scale_fill_gradientn(name='', col
ours=c('blue','yellow'))
  } else {
    p <- p + geom_hex(bins=nbins) + scale_fill_gradientn(name='', colours=c('blue','yellow'))
  }
```

```
  p <- p + geom_point(data=estLevel, color='red', shape=1) + geom_line(data=estLevel, color='red')
  p <- p + geom_text(aes(HR, -log10(pvalue), label=k,vjust=-1), data=estLevel, color='black')
  pQuant$x <- max(subFrame$HR)
  p <- p + geom_hline(aes(yintercept=-log10(pvalue), alpha=.4), linetype='dashed', data=pQuant)
  p <- p + geom_text(aes(x=x, y=-log10(pvalue), label=round(probs*100, 2), vjust=-.2), data=pQuant)

  hQuant$y <- max(c(-log10(subFrame$pvalue), -log10(0.05)))
  p <- p + geom_vline(aes(xintercept=HR, alpha=.4), linetype='dashed', data=hQuant)
  p <- p + geom_text(aes(x=HR, y=y, label=round(probs*100, 2), hjust=-.1, vjust=-.1), data=hQuant)
  p <- p + geom_hline(yintercept=-log10(0.05))
  p <- p + scale_x_continuous(limits=xRange) + scale_y_continuous(limits=yRange)
  #p <- p + ggtitle(sprintf('RHR = %.02f\nRP = %.02f', RHR, RPvalue))
  p <- p + xlab('ATE (Odds Ratio Scale)') + ylab('-log10(pvalue)') + theme_bw()
  return(p)
}


vibcontour_cox <- function(vibObj, factor_num=1, alpha=1) {
  vibFrame <- vibObj$vibFrame
  subFrame <- subset(vibObj$vibFrame, factor_level == factor_num)
  subFrame$pvalue <-subFrame$`Pr(>|t|)`
  contourData <- getContoursForPctile(subFrame)
  subFrame$factor_level <- NULL
  yRange <- range(c(-log10(.05), -log10(vibFrame$'Pr(>|t|)')), na.rm=T)
  xRange <- range(vibFrame$HR, na.rm=T)
  probs <- c(0.1,0.5,0.9)
  hProbs <- c(0.1,0.5,0.9)
  estLevel <- statPerK(subFrame)
  estLevel$HR <- exp(estLevel$estimate)
  pQuant <- quantilesPvalue(subFrame, probs)
  hQuant <- quantilesHR(subFrame, hProbs)
  maxk <-max(vibFrame$k)
  rowid<- with(vibFrame[vibFrame$k==maxk,], which(HR == quantile(HR, .5, type = 1,na.rm=TRUE)))
  medianOR <-vibFrame$HR[rowid]
  loor <-exp(log(medianOR) -1.96*vibFrame$`Std. Error`[rowid])
  upor <-exp(log(medianOR) +1.96*vibFrame$`Std. Error`[rowid])
  Evalues <-evalues.RR(medianOR,loor,upor)
  if(loor<=1 & upor>=1){
    FEvalue <-as.character(c(1,1,1))
  } else {
    FEvalue <-as.character(round(Evalues[2,],2))
  }
  #RHR <- round(hQuant[3,'HR']/hQuant[1,'HR'], 2)
  #RPvalue <- round(-log10(pQuant[1,'pvalue']) + log10(pQuant[3,'pvalue']), 2)
```

```
  p <- ggplot(subFrame, aes(x=HR, y=-log10(pvalue)))
  if(sum(colnames(subFrame) == 'has_variable')) {
    p <- p + geom_point(aes(colour=factor(has_variable)), alpha=alpha) + scale_colour_manual(values=CB
B_PALETTE)
  } else {
    p <- p + geom_point(alpha=alpha,color="grey")
  }
  p <- p + geom_contour(data=contourData$densityData, aes(x=x,y=y,z=z), breaks=contourData$levels, siz
e=.3,color="navy",alpha=alpha)
  p <- p + geom_point(data=estLevel, aes(color=k))+ scale_color_gradient(low="yellow", high="red") + g
eom_line(data=estLevel, color='darkorange')
  #p <- p + geom_text(aes(HR, -log10(pvalue), label=k,vjust=-1), data=estLevel, color='red4')
  pQuant$x <- max(subFrame$HR)
  p <- p + geom_hline(aes(yintercept=-log10(pvalue)), linetype='dashed', data=pQuant,alpha=0.3)
  p <- p + geom_text(aes(x=x, y=-log10(pvalue), label=round(probs*100, 2), vjust=-.2), data=pQuant)

  hQuant$y <- max(c(-log10(subFrame$pvalue), -log10(0.05)))
  p <- p + geom_vline(aes(xintercept=HR), linetype='dashed', data=hQuant,alpha=0.3)
  p <- p + geom_text(aes(x=HR, y=y, label=round(probs*100, 2), hjust=-.1, vjust=-.1), data=hQuant)

  p <- p + geom_hline(yintercept=-log10(0.05),color="darkmagenta",size=1.1)
  p <- p + scale_x_continuous(limits=xRange) + scale_y_continuous(limits=yRange)
  #grob <- grobTree(textGrob(paste0('Evalue for Full PS Model:', FEvalue[1]), x=0.05,  y=0.9, hjust=0,
                            #gp=gpar(col="black", fontsize=12, fontface="italic")))
  #p <- p  + annotation_custom(grob) # paste0('Evalue',": ", FEvalue[1]," ","(",FEvalue[2],",", FEvalu
e[3],")"),

 # p <- p + ggtitle("Oral Therapy, IPTW")
  p <- p + labs(color="K",
                y='PValue (-log10 scale)',
                caption=paste0('Evalue for Full PS Model:', FEvalue[1])) + theme_bw() + theme(legend.p
osition = "none")
  return(p)
}

find_adjustment_variable <- function(vibObj, adjustment_num=1) {
  vibFrame <- vibObj$vibFrame
  combinations <- vibObj$combinations
  ks <- unique(vibFrame$k)
  vibFrame[, 'has_variable'] <- 0
  for(ii in 1:length(ks)) {
    k <- ks[ii]
```

```
    adjusters <- combinations[[ii]]
    combIndex <- which(apply(adjusters, 2, function(arr) {sum(arr==adjustment_num)})==1)  ## gives col
umn
    if(length(combIndex)) {
      vibFrame[vibFrame$k == k & (vibFrame$combination_index %in% combIndex), 'has_variable'] <- 1
    }
  }
  vibObj$vibFrame <- vibFrame
  return(vibObj)
}

plot_vibration_cox <- function(vibObj, type=c('bin', 'contour'), factor_num=1, adjustment_num=NA, ...)
{
  ### plots the vibration of effects for a cox model
  if(length(type)) {
    type <- type[1]
  }

  if(!is.na(adjustment_num)) {
    vibObj <- find_adjustment_variable(vibObj, adjustment_num)
  }

  if(type == 'bin') {
    return(vib2d_cox(vibObj, factor_num, ...))
  } else if(type == 'contour') {
    return(vibcontour_cox(vibObj, factor_num, ...))
  }

  return(NULL)
}



#other called functions
## Chirag Patel
## 4/18/2013
### functions to post processes a vibFrame

library(MASS)

meanEstimate <- function(subFrame) {
  pval <- median(subFrame$pvalue)
  hr <- median(subFrame$estimate)
  return(data.frame(estimate=hr, pvalue=pval))
}
```

```
mean_manhattan <- function(arr) {
  ### computes a manhattan distance (pairwise differences)
  ### then computes the relative distance and means it
  dd <- as.matrix(dist(arr, method='manhattan'))
  dd <- dd / abs(arr)
  mean(dd[upper.tri(dd)])*100
}

cdfPerPvalue <- function(subFrame, pvalues=c(10^(-10:-2), .02, .03, .04, .05, .06, .07, .08, .09, .1))
{
  Fn <- ecdf(subFrame$pvalue)
  data.frame(pvalue=pvalues,cdf=Fn(pvalues), number=Fn(pvalues)*nrow(subFrame))
}

quantilesPvalue <- function(subFrame, probs=c(0.01, .25, 0.5, .75, 0.99)) {
  qs <- quantile(subFrame$pvalue, probs)
  data.frame(probs=probs, pvalue=qs)
}

quantilesHR <- function(subFrame, probs=c()) {
  ### change this to estimate.
  qs <- quantile(subFrame$estimate, probs)
  data.frame(probs=probs, HR=exp(qs))
}

quantilesEstimate <- function(subFrame, probs) {
  qs <- quantile(subFrame$estimate, probs)
  data.frame(probs=probs, estimate=qs)
}



statPerK <- function(vibFrame) {
  ### computes a mean HR and median p-value for each k and vibration for each k
  estLevel <- data.frame()
  ks <- sort(unique(vibFrame$k))
  levs <- unique(vibFrame$factor_level)
  for(ii in ks) {
    subFrame <- subset(vibFrame, k==ii)
    mn <- meanEstimate(subFrame)
    estLevel <- rbind(estLevel, data.frame(k=ii, estimate=mn$estimate, pvalue=mn$pvalue))
  }
  estLevel
}
```

```
statPerKandFactor <- function(vibFrame) {
  levs <- unique(vibFrame$factor_level)
  estLevels <- data.frame()
  for(ii in levs) {
    subFrame <- subset(vibFrame, factor_level == ii)
    estLevel <- statPerK(subFrame)
    estLevel$factor_level <- ii
    estLevels <- rbind(estLevels, estLevel)
  }
  return(estLevels)
}


summary.vibration <- function(vibFrame, bicFrame=NULL) {
  ### this is for cox model.

  ### take in a data.frame and compute all summary stats
  ## do per factor? -- yes.
  # HR 99 and HR 1 -- if sign change for the 99 vs. 1?
  # P 99 and P 1; how many < thresholds
  # HR 99/ HR 1
  # -log10P1 + log10P99
  # stat per K (mean HR/ median p per K/factor)
  levs <- unique(vibFrame$factor_level)
  summaryFrame <- data.frame()
  pvalue_cdf <- data.frame()
  bestMod <- NULL;
  if(!is.null(bicFrame)) {
    combInd <- bicFrame[which.min(bicFrame[,2]), 3]
    bestK <- bicFrame[which.min(bicFrame[,2]), 4]
    bestMod <- subset(vibFrame, k == bestK & combination_index == combInd)
  }
  for(ii in levs) {
    subFrame <- subset(vibFrame, factor_level == ii)

    hrs <- quantilesHR(subFrame, probs=c(.01,.5, .99))
    ps <- quantilesPvalue(subFrame, probs=c(.01,.5, .99))
    hr_01 <- hrs[1, 'HR']
    hr_50 <- hrs[2, 'HR']
    hr_99 <- hrs[3, 'HR']
    p_01 <- ps[1, 'pvalue']
    p_50 <- ps[2, 'pvalue']
    p_99 <- ps[3, 'pvalue']
```

```
    RHR <- hr_99/hr_01
    vibP <- -log10(p_01) + log10(p_99)
    frm <- data.frame(HR_01=hr_01, HR_50=hr_50, HR_99=hr_99,pvalue_01=p_01,pvalue_50=p_50, pvalue_99=p
_99, rHR=RHR, rPvalue=vibP, factor_level=ii)
    if(!is.null(bestMod)) {
      bestSub <- subset(bestMod, factor_level == ii)
      frm$HR_bic <- bestSub[1, 'HR']
      frm$pvalue_bic <- bestSub[1, 'pvalue']
    }

    summaryFrame <- rbind(summaryFrame, frm)
    cdfPerP <- cdfPerPvalue(subFrame)
    cdfPerP$factor_level <- ii
    pvalue_cdf <- rbind(pvalue_cdf, cdfPerP)
  }

  perK <- statPerKandFactor(vibFrame)

  return(list(summary=summaryFrame, pvalue_cdf = pvalue_cdf, summary_per_k=perK))
}
summary.vibration.stratum <- function(vibFrame) {
  ## gets a summary per stratum
  ## for cox model.
  strata <- unique(vibFrame$stratum)
  summaryFrame <- data.frame()
  summary_per_k <- data.frame()
  pvalue_cdf <- data.frame()
  for(ii in strata) {
    perStrat <- summary.vibration(subset(vibFrame, stratum == ii))
    perStrat$summary[, 'stratum'] <- ii
    perStrat$summary_per_k[, 'stratum'] <- ii
    perStrat$pvalue_cdf[, 'stratum'] <- ii
    summaryFrame <- rbind(summaryFrame, perStrat$summary)
    summary_per_k <- rbind(summary_per_k, perStrat$summary_per_k)
    pvalue_cdf <- rbind(pvalue_cdf, perStrat$pvalue_cdf)
  }

  return(list(summary=summaryFrame, pvalue_cdf=pvalue_cdf, summary_per_k=summary_per_k))
}

getContoursForPctile <- function(vib, pctiles=seq(.05, .95, by=.05)) {
  dens <- kde2d(vib$HR, -log10(vib$pvalue), n = 200)
  ### this is from http://stackoverflow.com/questions/16225530/contours-of-percentiles-on-level-plot/1
6228938#16228938
```

```
  ### HPDRegionplot code in the emdbook package
  dx <- diff(dens$x[1:2])
  dy <- diff(dens$y[1:2])
  sz <- sort(dens$z)
  c1 <- cumsum(sz) * dx * dy
  levels <- sapply(pctiles, function(x) {
    approx(c1, sz, xout = 1 - x)$y
  })
  densityData <- data.frame(expand.grid(x = dens$x, y = dens$y), z = as.vector(dens$z))
  return(list(levels=levels, densityData=densityData))
}
```